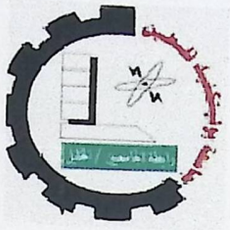
Palestine Polytechnic University



College Of Engineering and Technology
Electrical and Computer Engineering Department

Graduation Project

Educational USB Data Acquisition Card

Project Team

Raghda S. Al_swiety

Razan S. Qabaja

Wala M. Dandis

Project Supervisor

Eng.Mazen Zallom

Hebron-Palestine
Jun, 2009

جامعة بوليتكنك فلسطين

كلية الهندسة والتكنولوجيا

دائرة الهندسة الكهربائية والحاسوب

اسم المشروع

Educational USB Data Acquisition Card

أسماء الطلبة

رزان سعدي قباجة **رغده سلامة السويطي**

ولاء محمد سالم دنديس

بناءً على نظام كلية الهندسة والتكنولوجيا وإشراف ومتابعة المشرف المباشر على المشروع وموافقة أعضاء اللجنة الممتحنة تم تقديم هذا المشروع إلى دائرة الهندسة الكهربائية والحاسوب، وذلك للوفاء بمتطلبات درجة البكالوريوس في الهندسة تخصص هندسة أنظمة الحاسوب .

توقيع المشرف

.....

توقيع اللجنة الممتحنة

....

توقيع رئيس الدائرة

.....

Dedication

To our families and friends we dedicate this humble effort

Acknowledgment

We would like to seize this opportunity to extend best wishes and gratitude to well-deserved supervisor Eng. Mazen Zallom, along with respectable Eng. Sami Al_salamen & Mr. Rasmi Saed Ahmed, and all who have been so forthcoming and instrumental in helping us reach our goals.

Abstract

This project aims to design and implement a general purpose educational data acquisition card. The card is interconnected with computer through the popular Universal Serial Bus (USB), which has become an important widely used tool in science and technology. This card was intended to be used to control many applications which enable users, including teachers and students, to perform a variety of experiments; four applications have been implemented and tested which are: fan, heater, stepper motor, and alarm system.

The microcontroller PIC18f4550 was used in building the data acquisition card because of its capability to provide the entire necessary control signal for the applications presented.

Visual C++ 2008 software was used as a tool for the graphical user interface, to enable users to interact with the system in efficient manner.

الملخص

يهدف هذا المشروع إلى بناء كرت تعليمي (Educational USB Data Acquisition Card), بحيث يتم توصيل هذا الكرت مع جهاز الكمبيوتر عن طريق(USB port) , يتحكم الكرت بالعديد من التطبيقات ,في هذا المشروع تم اختيار عدة أمثلة عمليه ليتم التحكم بها, وهي(Alarm system, Fan ,Heater, stepper motor) ليتسنى للمستخدم التعامل مع النظام بشكل فعال وإجراء التجارب على التطبيقات التي يدعمها النظام.

للتحكم بالإشارات تم اختيار(microcontroller PIC18f4550), لما له من امتيازات وخصائص تمكنه من التعامل مع الإشارات بالطريقة المطلوبه , ولمعالجة وتمثيل الإشارات على جهاز الكمبيوتر يعتمد المشروع استخدام (Visual C++ 2008) لعرض الإشارات والبيانات المطلوبة على واجهة المستخدم.

List of Contents

Dedication.....II
Acknowledgments.....III
Abstract.....IV
الملخص.....V
List of Contents.....VI
List of Tables.....IX
List of Figures.....IX
List of abbreviations.....X

Chapter 1: Introduction 1

1.1 Preface 2
1.2 Project Importance.....3
1.3 Project Objectives.....3
1.4 Literature Review.....4
1.5 Requirements.....5
1.5.1 User requirements.....5
1.5.2 System requirements.....5
1.5.2.1 Functional requirements.....5
1.5.2.2 Non- functional requirements.....6
1.6 TimePlane.....7
1.7 Estimated Cost.....8
1.8 Project Risks.....10
1.9 Project Road Map.....10

Chapter 2: Theoretical Background 12

2.1 Universal Serial Bus(USB)..... 13

2.2 Benefits of the Universal Serial Bus (USB) for Data Acquisition.....13
2.3 Main components of USB system.....16
2.4The USB Protocol.....18
2.5 Transfer Basics.....19
2.5.1 Elements of a Transfer.....20
2.5.2 USB Transfer Types.....22
2.6 USB Pipes.....30
2.7 USB Cables and Ports.....31
2.8 USB Versions.....33
2.9 Enumeration of Devices.....33
2.10 The Data Acquisition System.....36
2.11 Hardware Components of the Project.....37
2.11.1 Microcontroller 37
2.11.2 Sensors 38
2.11.3 Signal Conditioning 39

Chapter 3: Conceptual Design 40

3.1Preface.....41
3.2 Detailed Project Objectives.....41
3.3 Design Options.....42
3.3.1 Hardware Design Options..... 42
3.3.1.1 Computer Interface Options.....42
3.3.1.2 Control Unit Design Options.....43
3.2.2 Software Design Options.....47

3.2.2.1 Microsoft Visual Studio 2008 Express.....47
3.2.2.2 VB.NET.....47
3.4 Realization Approach.....48
3.5 General Block Diagram.....48
3.5.1 Detailed Block Diagram.....49
3.6 The system work.....50

Chapter 4: Detailed System Design 51

4.1 Preface.....52
4.2 Project Phases.....52
4.3 Schematic design.....53
4.3.1Microcontroller PIC 18F4550.....54
4.3.2LM35TemperatureSensor.....55
4.3.3 Conditioning Circuits.....56
4.3.4 Stepper Motor.....57
4.4 General Schematic Design for USB Board.....59
4.5 Ports Assignment.....61
4.6 Applications presented.....62
4.6.1Alarm system using simple buzzer.....62
4.6.2 Heater Circuit.....62
4.6.3 Fan Circuit.....63
4.6.4 Stepper Motor Circuit.....64
4.7 Overall system schematic design.....64

Chapter 5: Conceptual Design 66

5.1 Preface.....67
5.2 USB Class Drivers.....67
5.2.1 Human Interface Devices (HID).....68
5.2.2 USB Communications Device Class (USB-CDC).....68
5.2.3 Microchip General Purpose (Custom Class) USB Driver 69
5.3 USB library 69
5.3.1 Libusb-win32.....70
5.3.2 MPUSBAPI Library and DLL Source.....70
5.4 MPLAB IDE version 8.0 71
5.5 Microsoft Visual C++ 2008 Express 72
5.6 Enumeration about Devices 72
5.7 System Flowcharts.....74
5.8 Graphical User Interface.....88

Chapter 6: **Software System Design** **94**

6.2 Preface 95
6.2 Implementation..... 95
6.3Project components testing.....96
6.3.1 Subsystem testing.....96
6.3.2 Integrated System Testing.....100

Chapter 7: **Software System Design** **101**

7.1 Introduction.....102
7.2 Conclusion.....102
7.3 Problems 103
7.3.1 Hardware problems.....103
7.3.2 Software Problems.....104
7.4 Future work 104

References 106

List Of Tables

Table (1.1) Time Plan(First Semester).....7
Table (1.2) Time Plan(Second Semester).....8
Table (1.3) Estimated Hardware Cost.....9
Table (1.4) Estimated software Cost Table.....9
Table (2.1) The transactions source and content.....21
Table (2.2) comparison between USB versions.....32
Table (4.1) USB Cable 60
Table (4.2) Summary of the I/O ports on PIC18F4550.....61

List Of Figures

Figure (2.1) Control Transfer Setup Stage.....23
Figure (2.2) Control Transfer Data Stage.....23
Figure (2.3) Control Transfer IN Status Stage.....25
Figure (2.4) Control Transfer OUT Status Stage.....25
Figure(2.5) Interrupt IN and Interrupt OUT transaction.....28
Figure (2.6) Isochronous IN and OUT transaction.....29
Figure (2.7) Bulk IN and OUT Transaction.....29
Figure (2.8) USB Cable 33
Figure (3.1) FPGA.....44
Figure (3.2) Microprocessor.....45
Figure (3.3) General Block Diagram.....48
Figure (3.4) Detailed Block Diagram.....49
Figure (4.1) PIC18f4550.....54
Figure (4.2)block diagram of PICF4550.....55
Figure (4.3) LM35 Temperature.....56
Figure (4.4) conditioning circuit.....56
Figure (4.5) Stepper Motor.....57
Figure (4.6) ULN2803.....58
Figure (4.7) Main USB Board.....59
Figure (4.8) Heater Circuit.....63
Figure (4.9) Fan Circuit 63

Figure (4.10) Stepper Motor Circuit.....64
Figure (5.1) MPLAB IDE work space.....71
Figure (5.2)Flowchart for installing driver of DAQ.....75
Figure (5.2)Firmware Flowchart.....77
Figure (5.4) Loading MPUSBAPI library Flowchart.....79
Figure (5.5) Handling the command Flowchart.....81
Figure (5.6) Digital Input case Flowchart.....82
Figure (5.7)Digital Output case flowchart.....83
Figure (5.8) Analog Output case flowchart.....84
Figure (5.9) Analog Input case flowchart.....85
Figure (5.10) sensor application flowchart.....86
Figure(5.11) Stepper Motor application flowchart.....87
Figure (5.12) Main GUI form of the system.....88
Figure (5.13) General USB data acquisition card form.....89
Figure (5.14) Buzzer application form.....89
Figure (5.15) Switch application form.....90
Figure (5.16) Sensor application form.....91
Figure (5.17) Fan application form.....92
Figure (5.18) Heater application form.....92
Figure (5.19) Stepper motor form.....93
Figure (6.1) PIC18F4450 Circuit with PIKIT2 Programmer before Testing.....97
Figure (6.2) PIC18F4450 Circuit after Successful Test.....97
Figure (6.3) Buzzer testing.....98
Figure (6.4) Heater circuit before test.....98
Figure (6.5) Heater circuit after test.....99
Figure (6.6) Fan Circuit Testing.....99
Figure (6.7) Stepper Motor Circuit Testing.....100

List of abbreviations

ADC	Analog to Digital Converter
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
CDC	Communication Device Class
CMOS	Complementary Metal Oxide Semiconductor
CRC	Cyclic Redundancy Check
DAQ	Data Acquisition
DWG	Device Working Group
FIFO	First In First Out
FPGA	Field-Programmable Gate Array
GB	Giga Byte
Gbps	Giga bit per second
GHz	Giga Hertz
GUI	Graphical User Interface
HID	Human Interface Device
IDE	Integrated Development Environment
LSB	Least Significant Bit
MIPS	Million Instructions Per Second
MSC	Mass Storage Device
SDK	Software Development Kit
SOF	Start Of Frame

PCI	Peripheral Component Interconnect
PIC	Programmable Interface Controller
USB	Universal Serial Bus

Chapter

1 *INTRODUCTION*

1.1 Preface.

1.2 Project Importance.

1.3 Project Objectives.

1.4 Literature Review.

1.5 Requirements.

1.6 Time Plan.

1.7 Expected Cost.

1.8 Project Risks.

1.9 Road Map.

1.1 Preface

The Universal Serial Bus (USB) is a fast and flexible interface for connecting devices to computers. Every new PC has at least a couple of USB ports. The interface is versatile enough to use with standard peripherals like keyboards and disk drives as well as more specialized devices.

USB is designed to be plug and play interface for almost all the standard peripherals; in short, USB is very different from the legacy interfaces it is replacing. On attaching to a PC through USB, a device must respond to a series of requests that enable the PC to learn about the device and establish communications with it.

Developing a USB device and the software that communicates with it require knowing something about how USB works and how the PC's operating system implements the interface.

In addition, the right choice of Controller chip, device class, tools, and techniques can go a long way in avoiding snags and simplifying what needs to be done.

To communicate with a USB peripheral, a PC needs two things: a device driver that knows how to communicate with the PC's USB drivers, and an application that knows how to talk to the device driver.

This project is based on building a data acquisition card that is completely compatible with the Universal Serial Bus standards. The acquisition system communicates and controls the signals for the applications supported by the system which are: alarm, conditioning system (heater and fan), and stepper motor system, also Visual C++ Express will be used as a tool to interface between the user and the computer system.

Every USB peripheral must contain an intelligent controller. There are dozens of controller chips designed for use in USB peripherals. This project uses PIC18F4550 microcontroller because of its ability to handle most of USB characteristics.

1.2 Project Importance

The importance of this project stems from its dependency on The USB technology for acquiring data. The reason for choosing USB is that it is extremely convenient for data acquisition for several reasons that will be discussed later in our project, also since data acquisition cards play an important role in many projects, that is a large number of projects depends on acquiring data, so we attempt to implement and to make use of USB in order to facilitate the process of dealing with data acquisition process.

This project aims to give both educators and students a deep understanding to deal with USB data acquisition, through building an educational USB data acquisition card.

1.3 Project Objectives

The main objectives of our project are:

1- To support users with important information and features about the USB technology.

2- To design and implement a data acquisition card system that is capable to be remote controlled with command messages that are sent through USB.

3- To design a USB 2.0 interface that can be generalized for various custom applications, providing a platform that can be easily adapted for new applications technology.

4- To use USB technology in order to communicate with and control various applications.

5- To help both educators and students to become more familiar with USB DAQ.

6- To implement and design a friendly system that would be easily understood and handheld by end user.

7- To provide the system with real applications, supported by a GUI that is easy to be used and understood by students and end users.

1.4 Literature review

Literature review related to Universal Serial Bus Data Acquisition Card

'USB Data Acquisition Card', this project was implemented on Oct, 2006 by Joe Evans and Bernhard Mayr; they built a data acquisition module that is compatible with the Universal Serial Bus standard. When the module is connected to a PC through a normal USB-cable, it should be able to deliver any data it has acquired to the PC, using AVR Atmega32 microcontroller.

'USB Tenki - Environmental Data Acquisition', granted on January. 2008, by Raphael Assenat .This is an electronic project to interface sensors to an USB port with

a multi channel data acquisition system for various types of sensors. It's been developed to record temperature, humidity and atmospheric pressure.

'Remote Sensor - Wireless Data Acquisition ', by Objective Development Software GmbH. This is a wireless thermo- and hygrometer with an USB port on the receiver. It can draw fancy diagrams of temperature and humidity for up to 16 wireless sensors. The system utilizes an interrupt endpoint to indicate to the host when new data arrived, demonstrating how AVR-USB handles a second endpoint. The receiver part is basically a serial to USB converter with special processing for the wireless protocol such as data integrity checks and data block buffering.

1.5 Requirements

1.5.1 User requirements:

- The system should provide a graphical User Interface (GUI) in order to help users to interact with the system simply and easily.
- The GUI should provide the user with the results and their explanations.
- The GUI should provide the user with many options to help him to communicate with the system in appropriate manner.

1.5.2 System requirements:

1.5.2.1 Functional requirements

- The DAQ system should be able to interpret signals detected by the sensors and input circuits.

- The DAQ system should be able to handle all USB specifications to send the acquired data to PC.
- The system should analyze the signals and provide the output results.
- The system should be able to control and communicate with the predetermined applications.

1.5.2.2 Non- functional requirements

- Usability: the system must be easy to use.
- Speed: the interface shouldn't become bottlenecked because of slow communications.
- Reliability: the system must be reliable under different circumstances and conditions.
- Low cost: so users don't balk at the price.
- Performance: the system should work with good performance measurements.

1.6 Time Plan

The time plan and task description for first semester is shown in table (1.1)

Table (1.1) Time Plan (first semester)

Week / Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Data gathering & analyzing																
Requirement analysis																
Studying PIC18f4550																
Studying USB																
Studying DAQ																
Project Design																
Documentation																

The time plan and task description for second semester is shown in table (1.2)

Table (1.2) Time Plan (second semester)

Week / ask	17	28	19	20	21	22	23	24	25	26	27	28	29	30	31	3
Hardware Design																
System Software																
System testing																
Documentation																

1.7 Estimated cost:

The project needs both hardware equipments and software programming that run on the computer, so this section shows the estimated cost.

Table (1.3) Estimated Hardware Cost

Component	Number of components	Estimated cost(NIS)
PIC18F4550	1	20
Adapter	1	20
Transistor	2	6
Resistors	20	5
Led	10	5
Diode	5	5
Switch	1	5
Breadboard	5	75
Wires	–	5
Buzzer	1	5
Crystal	1	5
opt coupler	2	6
Capacitor	1	2
	Total	164

Table (1.4) Estimated software Cost Table

Component	Estimated Cost (NIS)
Windows XP	1200
Visual C++ 2008 Express	Free from Microsoft
MPLAB IDE program	Free from lab
	Total: 1200

1.8 Project Risks

The project may face some problems and risks that we have to declare in the early time of the project designing and manipulation, and the project must avoid those problems to work in its high efficiency, so when we find a risk we try to solve it without effect on the total project as much as we can. The risks such as:

1. Illness or absence of a team member
2. Technology specifications change.
3. Unavailability of some project needed components under the pressure of political environments.
4. Some of project components not work correctly, it will break the whole system
5. Lack of experience in using lab Instruments.
6. Latency of some components arrival.

1.9 Project Road Map

The project consists of six chapters; each chapter describes a specific area of the project the contents of each chapter are as follows:

Chapter One "Introduction"

This chapter gives a general idea about the project, explaining the project importance, and main objectives; also it contains literature reviews of the studies related to our project, besides it includes requirements, the time plan, road map, and the total cost of the project.

Chapter Two "Theoretical Background"

This chapter gives a clear picture about the system theoretical background related to the main features of USB technology, data acquisition system and basic system components.

Chapter Three "Conceptual Design"

This chapter represents the system block diagram, flow charts, design options for the system, and the interfacing of components with each others.

Chapter Four "Detailed system design"

This chapter represents system circuits and the detailed block diagrams that describe the system process.

Chapter Five "Software System Design"

This chapter describes the software program, code and algorithms that describe the system processes.

Chapter Six "System Implementation and Testing"

This chapter describes the actual project testing and implementation in details.

Chapter Seven "Conclusion and Future Work"

This chapter presents the conclusions, the future works and recommendations, it lists results achieved and how this project could be improved in future, in addition of some suggestions for the future system developing.

Chapter

2 *THEORETICAL BACKGROUND*

2.1 Universal Serial Bus (USB).

2.2 Benefits of the Universal Serial Bus (USB) for Data Acquisition.

2.3 Main components of USB system.

2.4 The USB Protocol.

2.5 Transfer Basics.

2.6 USB Pipes.

2.7 USB Cables and Ports.

2.8 USB Versions.

2.9 Enumeration of Devices.

2.10 The Data Acquisition System.

2.11 Hardware Components of the project.

Chapter Two

Theoretical Background

This chapter provides an illustrative background for the project related topics, and components.

2.1 Universal Serial Bus (USB)

Universal Serial Bus (USB) is a set of connectivity specifications developed by Intel in collaboration with industry leaders. USB allows high-speed, easy connection of peripherals to a PC. When plugged in, everything configures automatically USB is designed from the ground up to be easy for end users, with no user configuring required in hardware or software.

USB can be considered as the most successful interconnect in the history of computing, it has numerous advantages over older ways of connecting peripherals to PCs, it is very different from the legacy interfaces it's replacing, below we list some of the advantages that encourage us to go for USB technology in acquiring data.

2.2 Benefits of the Universal Serial Bus (USB) for Data Acquisition

USB provides users with a simple alternative for developing test and measurement application, in fact employing USB features for data acquisition has great advantages by offering the following services:

- **Easy Installation**

Ease of use was a key feature in the creation of the USB, many features included to make USB devices easy to install. USB devices are true "plug-and-play". There is no setting of address lines, interrupts. You can simply connect your data acquisition module to the USB port of your PC using a standard, low-cost cable. The host PC automatically identifies the module when it is plugged in, and installs the software necessary to operate it. Simply connect your sensors to the module and within minutes, you're acquiring data: temperature, pressure, and sound - whatever you need.

- **Less Noise Sensitivity from the PC**

Since USB data acquisition modules are external to the computer this offers performance benefits for noise-sensitive measurements. The USB cable is typically 1 to 5 meters long, the circuit is located further away from the computer's noisy motherboard and power supplies, and closer to the sensors they will be measuring so that it will give more precise measures.

- **Cost Savings**

Although USB is more complex than earlier interfaces, its components and cables are inexpensive.

Many USB data acquisition modules include removable terminal blocks and connectors that conveniently handle all user I/O connections. This design is not only convenient, but cost-effective, since you don't have to purchase optional screw terminal accessories.

- **Full and High-Speed Transfer Rates**

Computers configured with USB 1.1 ports can transfer data to and from a USB data acquisition module at up to 12Mbits/second - this full-speed rate is useful for data streaming applications and supports data acquisition rates of up to 400 kHz. For high-performance applications, your PC must have a high-speed USB2.0 port. With USB 2.0, you can transfer data between the PC and your USB data acquisition module at up to 480Mbits/second.

- **Portability**

USB data acquisition modules are compact and portable. Because USB devices can work directly with most laptops, even the most sophisticated data acquisition applications come out of the lab and into the field

- **Hot-Swappable**

USB devices are designed to be installed or removed while the computer is running. You need on y to plug the device in, use it, and then remove it when done.

- **Expandability**

Using low-cost expansion hubs and USB cables, you can connect up to 127 data acquisition modules to a single USB port.

- **Simple Power Connections**

USB data acquisition modules can either be powered by the USB directly or from an external power source.

2.3 Main components of USB system

A typical USB system consists of:

- **One host**

There is only one host in the USB system; its main function is to handle the whole complexity of the USB protocol. The host controls the media access no one can access the bus unless it got an approval required from the host.

The host controller serves both the USB and the host and has the same functionality in every USB system [1].

Following are some of the host controller functions:

- Frame generation: The host controller is responsible to partition the USB time into time units, in a way that each time unit is 1msec and is called a "frame". The host controller issues, periodically, SOF (Start Of Frame) packet every 1msec (after the transmission of the SOF the HC can transmit any other transaction for the rest of the frame period). The SOF contains the current frame number, which is maintained by the host controller.
- Data Processing: The host controller handles the requests for data to and from the host.
- Protocol Engine: Handling the USB protocol level interface
- Error handling: The host controller handles errors such as:

1. Timeout: the function in the device is not responding.

2. CRC error.
3. Remote wakeup: The host controller is able to enter the USB into a suspend state, and to detect a remote wakeup signaling on the bus.

- **Hub**

The hub provides an interconnect point, which enables many devices to connect to a single USB port. The logical topology of the USB is a star structure; all the devices are connected (logically) directly to the host. It is totally transparent to the device what is its' hub tier (the number of hubs the data has to flow through). The hub is connected to the USB host in the upstream direction (data flows up to the host) and is connected to the USB device in the downstream direction (data flows down from the host to the device) [1].

Here is the hubs' main functionality:

- The responsibility of detecting an attachment and detachment of devices
- Handling the power management for devices that are bus-powered.
- Bus error detection and recovery.
- Manage both full and low speed devices. When a device is attached to the system the hub detects the speed, which the device operates in, and through the whole communication on the bus prevents from full speed traffic to reach low seed device and vice versa .

- **Device**

Everything in the USB system, which is not a host, is a device (including hubs). A device provides one or more USB functions. Devices may be self powered (device

which supplies its own power), or bus powered devices (a device that gets its power from the bus) [1].

There are three kinds of devices:

- Low-speed devices operate in 1.5Mbps.
- Full-speed devices operate in 12Mbps
- High-speed devices operate in 480 Mbps

2.4 The USB Protocol

As we mentioned previously the USB host handles most of the complexity of the USB protocol, which makes the peripherals design simple and low cost. Data flow can be from host to device and from device to host.

USB transactions are done through packets. Each transaction is composed usually from three phases:

1- Token phase - the host initiates token indicating the future transaction type.

2- Data phase - the actual data is transmitted through packet. The data direction matches the direction indicated by the token that was transmitted previously.

3- Handshake phase - (optional) - handshake packet is sent, indicating the success or failure of the transaction.

The USB uses a polling protocol. Whenever the host whishes to receive data from the device it issues a token packet addressed to that specific device. If the device has data to send it sends it after receiving the token and the host (if the handshake phase is included on the transfer) will respond with handshake packet. It the device doesn't have anything to send the host issues the token to the next device. If on the other hand, the host whishes to send data to the device, it will send the appropriate token and data

packet following it. The device will response by a handshake packet (if exist). When the host finishes transmitting data to that device; it issues a new token to the next one.

2.5 Transfer Basics

USB communications are divided into two categories, depending on whether they're used in configuring and setting up the device, or in the applications that carry out the device's purpose.

In configuration communications, the host learns about the device and prepares it for exchanging data. Most of these communications take place when the host enumerates the device on power up or attachment. Application communications occur when the host exchanges data for use with applications. These are the communications that performs the functions the device is designed for. For example, a data acquisition card, the application communications are the sending of signals acquired to the host to tell an application to display the result as specified in the user interface.

- **Configuration Communications**

During enumeration process (which will be discussed later), the device's firmware responds to a series of standard requests from the host.
On PCs, Windows performs the enumeration, so there's no user programming involved. However, to complete the enumeration, Windows must have two files available: an INF file (that is a text file that you can usually adapt if needed from an example provided by

the driver's provider) identifies the filename and location of the device's driver, and the device driver itself. The enumeration process is invisible to users [1].

- **Application Communications**

After the host has exchanged enumeration information with the device and a device driver has been assigned and loaded, the application communications can be fairly straightforward. At the host, applications can use standard Windows API functions to read and write to the device. At the device, transferring data typically requires placing data to send in the USB controller's transmit buffer, reading received data from the receive buffer, and on completing a transfer, ensuring that the device is ready for the next transfer.

Most devices also require additional firmware support for handling errors and other events [1].

2.5.1 Elements of a Transfer

- **Device Endpoints**

All transmissions travel to or from a device endpoint. The specification defines a device endpoint as "a uniquely addressable portion of a USB device that is the source or sink of information in a communication flow between the host and device [6].

This suggests that an endpoint carries data in one direction only however, a control endpoint is a special case that is bidirectional.

The data stored at an endpoint may be received data, or data waiting to transmit. The host also has buffers for received data and for data ready to transmit, but the host

doesn't have endpoints but it serves as the starting point for communicating with the device endpoints.

The unique address required for each endpoint consists of an endpoint number (which may range from 0 to 15) and direction. The direction is from the host's perspective: IN is toward the host and OUT is away from the host. An endpoint configured to do control transfers must transfer data in both directions.
Every device must have Endpoint 0 configured as a control endpoint (bidirectional). The other transfer types send data in one direction only.

Every transaction on the bus includes an endpoint number and a code that indicates the direction of data flow and whether or not the transaction is initiating a control transfer. The codes are IN, OUT, and Setup (see table 2.1)

Table (2.1) The Transactions Source and Content

Transaction Type	Source of data	Types of transfers that use this transaction type	contents
IN	Device	all	Generic data
OUT	Host	all	Generic data
Setup	host	control	A request

- **Pipes: Connecting Endpoints to the Host**

A USB pipe is a logical association between a device's endpoint and the host controller's software.

Before a transfer can occur the host establishes pipes shortly after system power-up or device attachment, on requesting configuration information from the device. If the device is removed from the bus, the host removes the no-longer-needed pipes.

2.5.2 USB Transfer Types

Data transfer takes place between the memory buffer on the host computer and an endpoint on the universal serial bus (USB) device. Data is organized into packets before it is transferred. The transfer type that is used depends on the pipe on which the transfer is being issued, which is determined by the USB device on your system. There are four main types [11]:

1- Control Transfer:

Control transfers are bidirectional transfers that are used by the USB system software mainly to query, configure, and issue certain generic commands to USB devices. Control transfers typically take place between the host computer and the USB device's endpoint 0.

A control transfer can have up to three stages.

1- **Setup Stage** where the request is sent consists of three packets.

- The setup token is sent first which contains the address and endpoint number.
- The data packet is sent next and always has a packet identifier (PID) type of data0 and includes a setup packet which details the type of request.
- The last packet is a handshake used for acknowledging successful receipt or to indicate an error. If the function successfully receives the setup data (CRC and PID etc OK) it responds with ACK, otherwise it ignores the data and doesn't send a handshake packet. Functions cannot issue a STALL or NAK packet in response to a setup packet.

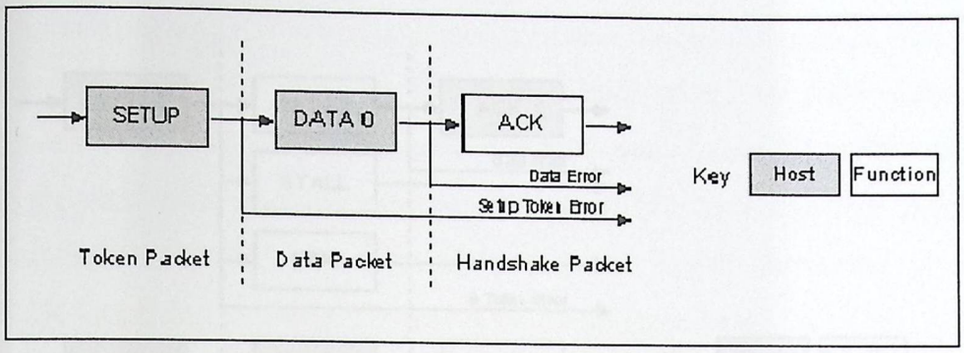


Figure (2.1) Control Transfer Setup Stage [11]

2- **Data Stage** (optional): consists of one or multiple IN or OUT transfers. The setup request indicates the amount of data to be transmitted in this stage. If it exceeds the maximum packet size, data will be sent in multiple transfers each being the maximum packet length except for the last packet.

The data stage has two different scenarios depending upon the direction of data transfer.

- **IN:** When the host is ready to receive control data it issues an IN Token. If the function receives the IN token with an error, then it ignores the packet. If the token was received correctly, the device can either reply with a DATA packet containing the control data to be sent, a STALL packet indicates the endpoint has had a error or a NAK packet indicating to the host that the endpoint is working, but temporary has no data to send.

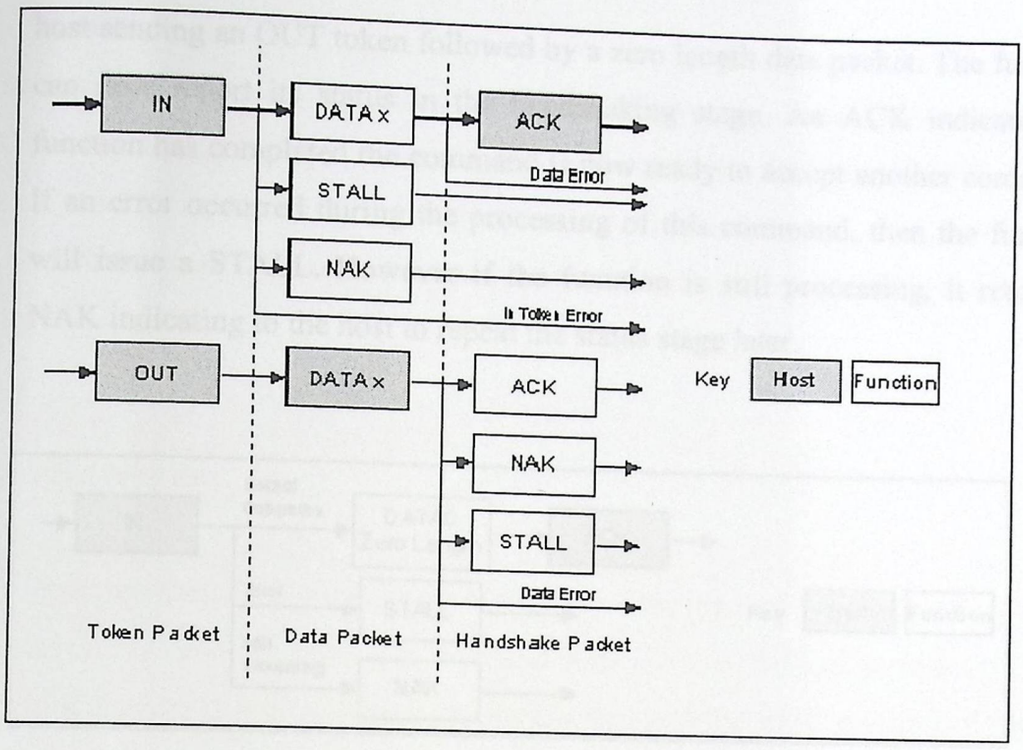


Figure (2.2) Control Transfer Data Stage [11]

- **OUT:** When the host needs to send the device a control data packet, it issues an OUT token followed by a data packet containing the control data as the payload. If any part of the OUT token or data packet is corrupt then the function ignores the packet. If the function's endpoint buffer was empty and it has clocked the data into the endpoint buffer it issues an ACK informing the host it has successfully received the data. If the endpoint buffer is not empty due to processing of the previous packet, then the function returns a NAK. However if the endpoint has had a error and its halt bit has been set, it returns a STALL.

3- Status Stage reports the status of the overall request and this once again varies due to direction of transfer. Status reporting is always performed by the function.

- **IN:** If the host sent IN token(s) during the data stage to receive data, then the host must acknowledge the successful receipt of this data. This is done by the

host sending an OUT token followed by a zero length data packet. The function can now report its status in the handshaking stage. An ACK indicates the function has completed the command is now ready to accept another command. If an error occurred during the processing of this command, then the function will issue a STALL. However if the function is still processing, it returns a NAK indicating to the host to repeat the status stage later.

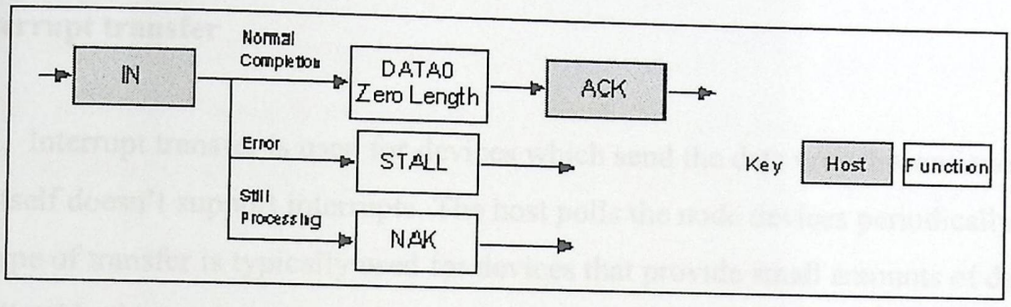


Figure (2.3) Control Transfer IN Status Stage [11]

- **OUT:** If the host sent OUT token(s) during the data stage to transmit data, the function will acknowledge the successful receipt of data by sending a zero length packet in response to an IN token. However if an error occurred, it should issue a STALL or if it is still busy processing data, it should issue a NAK asking the host to retry the status phase later.

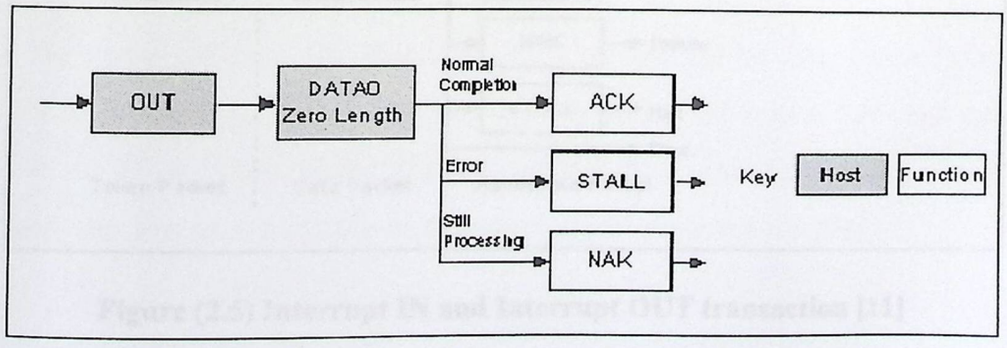


Figure (2.4) Control Transfer OUT Status Stage [11]

- **Data Size**

The maximum size of the data packet in the Data stage varies with the device's speed. For low-speed devices, the maximum is 8 bytes. For full speed, the maximum may be 8, 16, 32, or 64 bytes. For high speed, the maximum must be 64 bytes. These bytes include only the information transferred in the data packet, excluding the PID and CRC bits.

2- Interrupt transfer

Interrupt transfer is used for devices which send the data very discontinuous. USB itself doesn't support interrupts. The host polls the node devices periodically. This type of transfer is typically used for devices that provide small amounts of data at unpredictable times.

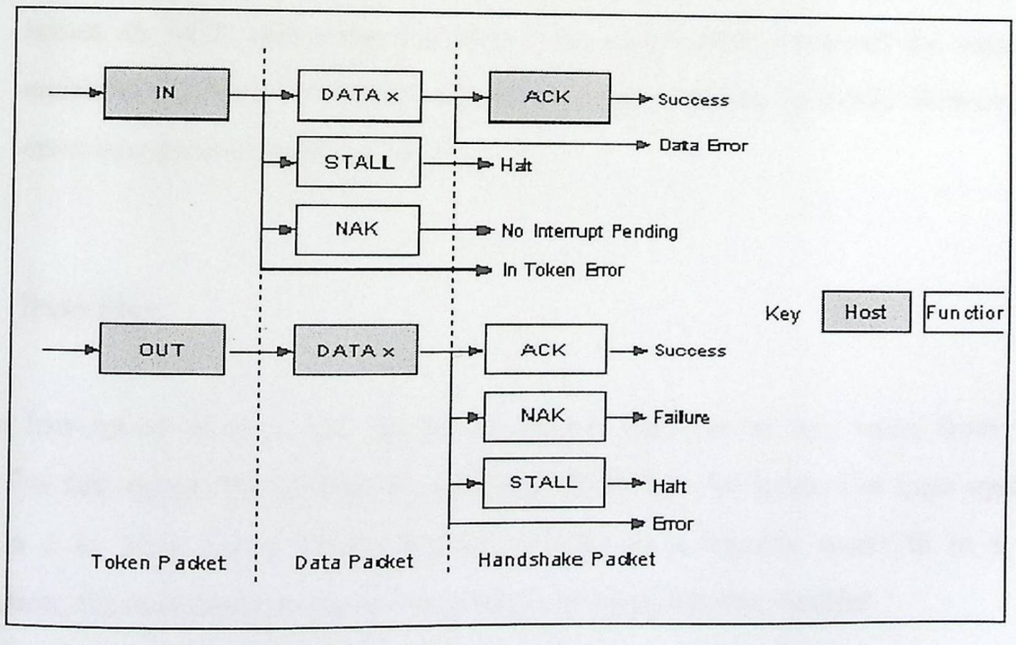


Figure (2.5) Interrupt IN and Interrupt OUT transaction [11]

The above diagram shows the format of an Interrupt IN and Interrupt OUT transaction.

- **IN:** The host will periodically poll the interrupt endpoint. This rate of polling is specified in the endpoint descriptor (which will be discussed later) each poll will involve the host sending an IN Token.

If an interrupt has been queued by the device, the function will send a data packet containing data relevant to the interrupt when it receives the IN Token.

Upon success the host will return an ACK. However if the data is corrupted, the host will return no status. If an interrupt condition was not present, then the function signals this state by sending a NAK. If an error has occurred on this endpoint, a STALL is sent.

- **OUT:** the host issues OUT token followed by a data packet containing the interrupt data. If any part of the OUT token or data packet is corrupted then the function ignores the packet. If the function's endpoint buffer was empty buffer it issues an ACK informing the host it has successfully received the data. If the endpoint buffer is not empty, then the function returns an NAK. However if an error occurred it returns a STALL.

- **Data Size**

For low-speed devices, the maximum packet size can be any value from 1 to 8 bytes. For full speed, the maximum can range from 1 to 64 bytes. For high speed, the range is 1 to 1024 bytes. If the amount of data in a transfer won't fit in a single transaction, the host uses multiple transactions to complete the transfer

.

3- Isochronous transfer

Isochronous transfers provide guaranteed amounts of bandwidth and latency. They are used for streaming data that is time-critical and error-tolerant or for real-time applications that require a constant data transfer rate. For isochronous transfers, timely data delivery is much more important than perfectly accurate or complete data transfer. The Disadvantage of this mode is that it doesn't use error detection 2009 [12]

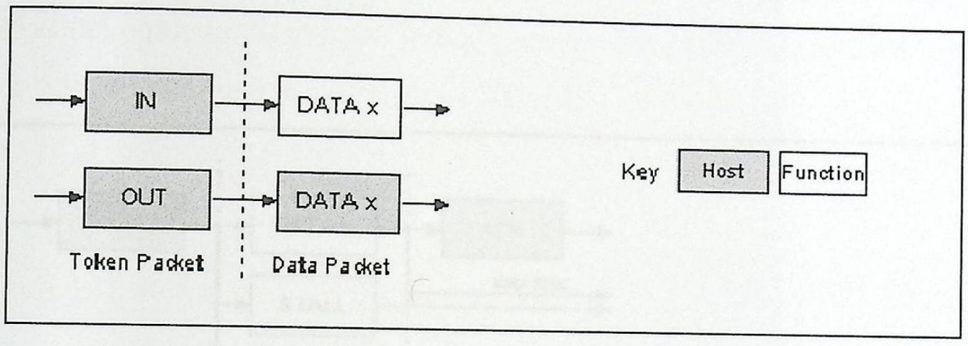


Figure (2.6) Isochronous IN and OUT transaction [11]

The above diagram shows the format of an Isochronous IN and OUT transaction.

- **Data Size**

For full-speed endpoints, the maximum packet size can range from 0 to 1023 data bytes. High-speed endpoints can have a maximum packet size up to 1024 bytes. If the amount of data won't fit in a single packet, the host completes the transfer in multiple transactions.

4- Bulk transfer:

Bulk transfers are for devices that have large amounts of data to transmit or receive and that require guaranteed delivery, but do not have any specific bandwidth or latency requirements.

It is a fast and secure data transfer with CRC check. Very slow or greatly delayed transfers can be acceptable for these types of device, as long as all of the data is delivered eventually. Bulk transfers are only supported by full and high speed devices.

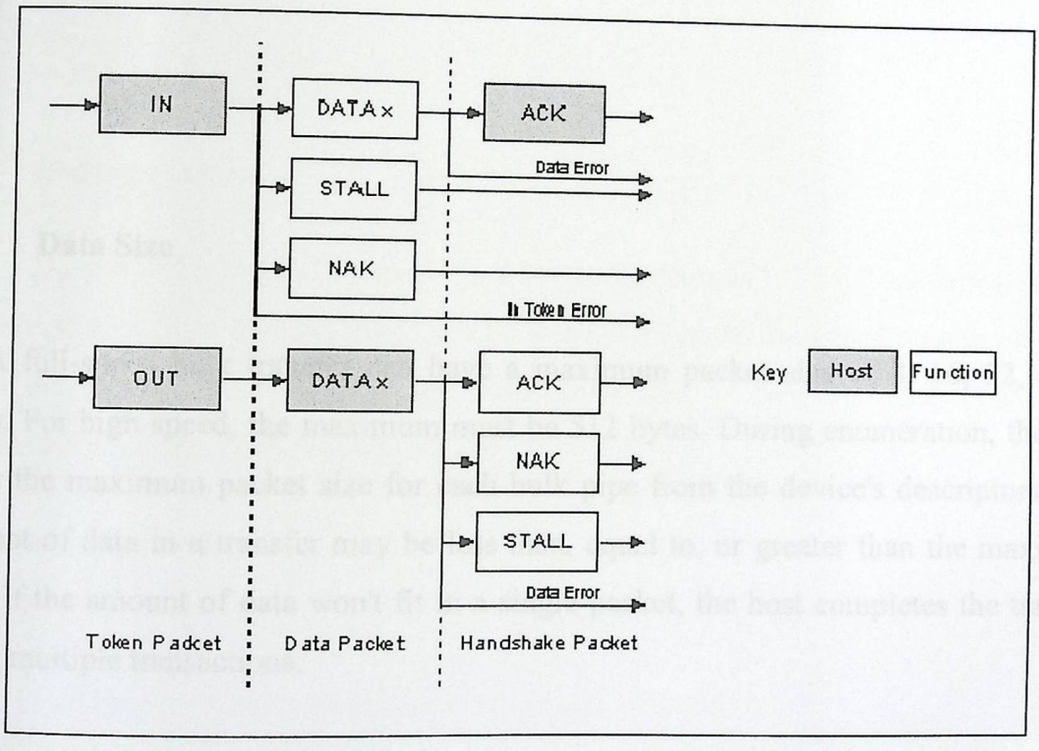


Figure (2.7) Bulk IN and OUT Transaction [11]

The above diagram shows the format of a bulk IN and OUT transaction.

- **IN:** When the host is ready to receive bulk data it issues an IN Token. If the function receives the IN token with an error, it ignores the packet. If the token was received correctly, the function can either reply with a DATA packet

containing the bulk data to be sent, or a stall packet indicating the endpoint has had a error or a NAK packet indicating to the host that the endpoint is working, but temporary has no data to send.

- **OUT:** When the host wants to send the function a bulk data packet, it issues an OUT token followed by a data packet containing the bulk data. If any part of the OUT token or data packet is corrupt then the function ignores the packet. If the function's endpoint buffer was empty and it has clocked the data into the endpoint buffer it issues an ACK informing the host it has successfully received the data. If the endpoint buffer is not empty due to processing a previous packet, then the function returns an NAK. However if the endpoint has had an error and its halt bit has been set, it returns a STALL.

- **Data Size**

A full-speed bulk transfer can have a maximum packet size of 8, 16, 32, or 64 bytes. For high speed, the maximum must be 512 bytes. During enumeration, the host reads the maximum packet size for each bulk pipe from the device's descriptors. The amount of data in a transfer may be less than, equal to, or greater than the maximum size. If the amount of data won't fit in a single packet, the host completes the transfer using multiple transactions.

2.6 USB Pipes:

While the device sends and receives data on a series of endpoints, the client software transfers data through pipes. A pipe is a logical connection between the host and endpoint(s). Pipes will also have a set of parameters associated with them such as how much bandwidth is allocated to it, what transfer type it uses, a direction of data

flow and maximum packet/buffer sizes. For example the default pipe is a bi-directional pipe made up of endpoint zero in and endpoint zero out with a control transfer type [11].

USB defines two types of pipes:

- **Stream Pipes**

Have no defined USB format that is you can send any type of data down a stream pipe and can retrieve the data out the other end. Data flows sequentially and has a pre-defined direction, either in or out. Stream pipes will support bulk, isochronous and interrupt transfer types. Stream pipes can either be controlled by the host or device.

- **Message Pipes**

Have a defined USB format. They are host controlled, which are initiated by a request sent from the host. Data is then transferred in the desired direction, dictated by the request. Therefore message pipes allow data to flow in both directions but will only support control transfers.

2.7 USB Cables and Ports

- Two types of cables can be used with USB devices

- The Series A connector: used with high speed (12 Mbps) devices, and can be up to 5 meters long.
- Series B cables: limited to 3 meters in length and are for use with low-speed (1.5 Mbps) USB peripherals such as keyboards and mice.

The USB cable is 4 wire cables, figure [2.8], signaling on the bus is done by signaling over two wires. There is a D_+ wire and a D_- wire, in a way that if we want to transmit 0 over the bus we will keep D_+ low and D_- high and vice versa to transmit 1 we need to keep D_- low and D_+ high. The other two cables are Vbus (+5v) and GND .T o deliver power to the device. Bits are sent into the bus LSB first.

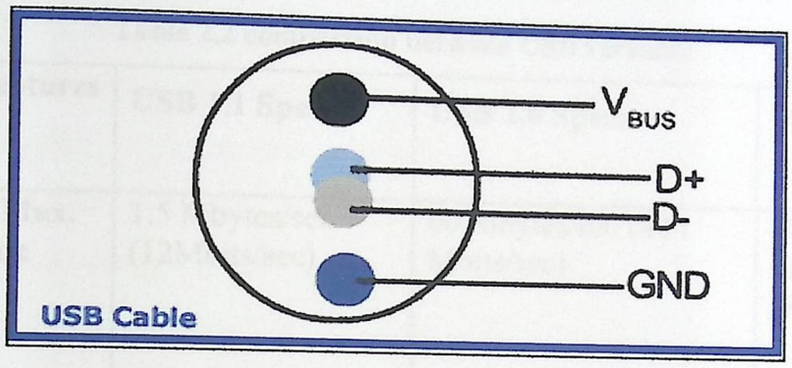


Figure (2.8) USB Cable [13]

- **Special signals in the bus:**

- Reset signaling (REST): The host can reset the device. This is done by signaling SE0 (single ended zero - D_+ and D_- are kept low) for more than 2.5 sec.
- Suspend signaling: the host can enter the device into a suspend mode, in which the device won't respond to the USB traffic. A device will begin the transition to a suspend mode whenever it will recognize an idle state on the bus for more than 3msec, the device will actually be suspended no more than 10msec bus inactivity. Recognizing signaling on its upstream ports will take the device out from the suspend mode
- Resume signaling: A device, which in suspended state, will resume its operation whenever it will recognize a K signaling (differential "0" for full speed devices and differential "1" for low speed devices) on the bus. Whenever the host wishes to wakeup the device it sends RESUME signaling for at least 20msec. A device can also wakeup itself - we call that feature "remote wakeup capability",

which allows the devices, which in suspend mode, start sending K signaling on the bus and resume its own activity **[13].**

2.8 USB Versions

Main USB version and their properties are shown below in table 2.2

Table 2.2 comparison between USB versions

Features	USB 1.1 Specs	USB 2.0 Specs	USB ٣.0 Specs
Theoretical Max. Transfer Rate	1.5 Mbytes/sec (12Mbits/sec)	60 Mbytes/sec (480 Mbits/sec)	USB 3.0 raises the data rate to 4.8 Gbps
Advantages	low cost, ideal for portable systems, hot swapping/plug & play, up to 127 devices via 1 port	All the advantages of USB plus significantly higher speeds making it compatible with high-speed peripherals such as data drives and video cameras	Unlike USB 2.0, USB 3.0 does not continuously poll the devices that are attached, thereby reducing overhead on the CPU.
Disadvantages	slower than PCI and other plug-in busses (such as Fire wire), not compatible with older peripherals	Not compatible with older peripherals, still slower than PCI	Will be obvious later

2.9 Enumeration of Devices

To communicate with a device, the host needs to learn about the device and assign a device driver. Enumeration is the initial exchange of information that accomplishes this. The process includes assigning an address to the device, reading data structures from the device, assigning and loading a device driver, and selecting a configuration

from the options presented in the retrieved data. This process allows the Host to identify and to manage the device [1].

- **Device identification:**

The Host sends standard device requests on the default control endpoint in order to identify the device and then to load the appropriate driver. The device answers to each request with the corresponding descriptor tables. The descriptor tables contain all the information relating to the device: characteristics of the device and number and characteristics of each configuration, interface and endpoint
The most common descriptors are:

- **Device Descriptors**

The device descriptor of a USB device represents the entire device. As a result a USB device can only have one device descriptor. It specifies some basic, yet important information about the device such as the supported USB version, maximum packet size, vendor and product IDs and the number of possible configurations the device can have [15].

- **Configuration Descriptor**

A USB device can have several different configurations although the majority of devices are simple and only have one. The configuration descriptor specifies how the device is powered, what the maximum power consumption is, the number of interfaces it has. Therefore it is possible to have two configurations, one for when the device is bus powered and another when it is mains powered. As this is a "header" to the Interface descriptors, it is also feasible to have one configuration using a different transfer mode to that of another configuration [15].

- **Interface Descriptors**

The interface descriptor could be seen as a header or grouping of the endpoints into a functional group performing a single feature of the device [15].

- **Endpoint Descriptors**

Endpoint descriptors are used to describe endpoints other than endpoint zero. Endpoint zero is always assumed to be a control endpoint and is configured before any descriptors are even requested. The host will use the information returned from these descriptors to determine the bandwidth requirements of the bus [15].

- **String Descriptors**

String descriptors provide human readable information and are optional. If they are not used, any string index fields of descriptors must be set to zero indicating there is no string descriptor available [15].

2.10 The Data Acquisition System

Data acquisition systems can be defined as a collection of software and hardware that connects you to the physical world. Acquired data are displayed, analyzed, and stored on a computer and control can be developed using various general purpose programming languages

A typical data acquisition system consists of these components:

- **Data acquisition hardware**

At the heart of any data acquisition system lies the data acquisition hardware. The main function of this hardware is to convert analog signals to digital signals, and vice versa

- **Sensors and actuators (transducers)**

Sensors and actuators can both be transducers. A transducer is a device that converts input property or physical phenomenon of one form into output of another form (mainly to its corresponding measurable electrical signal).

- **Signal conditioning hardware**

Sensor signals are often incompatible with data acquisition hardware. To overcome this problem, the signal must be conditioned. Conditioning an input signal may be by amplifying, it or by removing unwanted frequency components. Output signals may need conditioning as well.

- **The computer**

The computer provides a processor, a system clock, a bus to transfer data, memory, and disk space to store data.

- **Software**

Data acquisition software allows you to exchange information between the computer and the hardware. For example, typical software allows you to configure the sampling rate of your board, and acquire a predefined amount of data.

2.11 Hardware Components of the Project

2.11.3 Microcontroller

A microcontroller is an integrated chip that is often part of an embedded system. The microcontroller consist of the Central Processing Unit (CPU), Random Access Memory (RAM), Read Only Memory (ROM), Input/Output ports (I/O ports), timers like a standard computer and other peripheral such as analog to digital (A/D) converter and to analog (D/A) converter, they are much smaller and simplified so that they can include all the functions required on a single chip.

- **PIC 18f4550 microcontroller**

This subsection provides a theoretical background about the PIC18F4550 by mentioning .different tasks and features related to this microcontroller, that illustrate the reasons for choosing this PIC to handle USB specifications in data acquisition in this project [6].

- **Features of PIC18F4550 related to USB**

  - USB V2.0 Compliant.
  - Low Speed (1.5 Mb/s) and Full Speed (12 Mb/s).
  - Supports Control, Interrupt, Isochronous and Bulk Transfers.
  - Supports up to 32 endpoints (16 bidirectional).

- 1-Kbyte dual access RAM for USB.
- On-chip USB transceiver with on-chip voltage regulator.
- Interface for off-chip USB transceiver.

- **Features of PIC18F4550 related to Data Acquisition**

  - 10-bit, up to 13-channels Analog-to-Digital Converter module (A/D) with programmable acquisition time.

  - Dual analog comparators with input multiplexing Special.
  - Three external interrupts Four Timer modules (Timer0 to Timer3)

2.11.1 Sensors

In order to sense and measure physical variables such as temperature, pressure, flow and motion, it is necessary to use sensors (transducers) until to convert physical variables into electrical signal and transmit these signals to a signal conditioning device, so in our project we use the LM35 temperature to do this sense.

- **LM35 temperature sensor**

The LM35 is a very standard part for measuring temperature. It is highly stable and outputs a value proportional to temperature over a large scale [16].

- **Features:**

  - Calibrated directly in ° Celsius (Centigrade).

- Linear + 10.0mV/°C scale factor.
- 0.5°C accuracy guarantee able (at +25°C).
- Rated for full −55° to +150°C range.
- Low cost due to wafer-level trimming.
- Operates from 4 to 30 volts.
- Less than 60 μA current drain.
- Low self-heating, 0.08°C in still air.
- Nonlinearity only ±1⁄4°C typical.
- Low impedance output, 0.1 W for 1 mA load.

2.11.2 Signal Conditioning:

Signal conditioning maximizes the accuracy of a system, allows sensors to operate properly, and guarantees safety by many applications as amplifiers, filters the sensor signal, bridge completion, and provides appropriate output signal which is compatible and easy to capture with an analog input board.

- **USB Connector**

The USB Connector is a standard "type B" connector. You can plug any "A-to-B" type cable into it, with other end (the flat connector) going to your computer. As we mentioned there are four connections in a USB cable, two of which will supply power to the board, while the other two are the communications lines D+ and D-. This is how information is transferred between the host computer and the PIC18f4550 both when it is being programmed, and while your firmware sends or receives data with the computer if it is a Human Interface Device (HID) application.

Chapter

3 CONCEPTUAL DESIGN

3.1 Preface

3.2 Detailed Project Objectives

3.3 Design Options

3.4 Realization Approach.

3.5 General Block Diagram

3.6 The system work

Chapter Three

Conceptual Design

3.1 Preface

This chapter illustrates the detailed project objectives, design options suggested for the system, approaches and techniques to implement the system; also it includes a general block diagram for the overall system that gives the user a detailed description about the system components and architecture, in addition we will illustrate how the system works and the way in which components are interact and interface with each other and with the surroundings.

3.2 Detailed Project Objectives:

In chapter one, we have listed the general objectives of the system; here we list more detailed objectives that will clarify the general goal of the project:

1- Designing a data Acquisition system that will be able to be controlled with the command messages and signals that are sent through USB bus.

2- Designing a data Acquisition system that will be able to control many real applications, digital and analogue applications will be implemented, designed, and tested including: an alarm system, also a conditioning system (heater and cooler) that will work according to the temperature value that is read from the sensor, and a stepper motor, in order to make sense for users about the output from the system.

3- Visual C++ software will be used as a graphical interface between the user and the system to control its functionality and to enable users to practice the usage of USB, understand it efficiently and use it effectively, the interface should be flexible, and easy as possible to all end-users who need the system.

3.3 Design Options

3.3.1 Hardware Design Options

There are many issues that we must consider in order to build a functional, successful and reliable data acquisition card .There are many alternative options for implementing the system; this section outlines some of the available options for the user to design the system.

3.3.1.1 Computer Interface Options

- **Universal Serial Bus (USB)**

USB is a peripheral bus standard that allows you to connect a variety of peripheral devices to your computer.

Universal Serial Bus offers:

- Plug-and-play flexibility
- Automatic configuration of devices as soon as they are connected
- Hot swapping (the connecting and disconnecting of devices while the computer is on)
- Ability to have multiple devices run simultaneously on one computer, it can support up to 127 devices.
- USB supports three bus speeds: high speed at 480 Megabits per second, full Speed at 12 Megabits per second and low speed at 1.5 Megabits per second.

- Hi-Speed USB 2.0 uses a "Master-Slave" architecture in which the computer handles all arbitration functions and dictates data flow to, from and between the attached peripherals.

- **FireWire**

FireWire is an implementation of the high-speed serial data bus defined by IEEE Standard that can move large amounts of data between computers and peripheral devices. It features simplified cabling, hot swapping, and transfer speeds of up to 400 megabits per second.

FireWire uses a "Peer-to-Peer" architecture in which the peripherals are intelligent and can negotiate bus conflicts to determine which device can best control a data transfer .FireWire speeds up the movement of multimedia data and large files and enables the connection of digital consumer, directly to a personal computer.

Although FireWire external hard drive will provide good performance, but for convenience and compatibility between multiple computers USB would probably be the better choice (since practically every computer has a USB port), also people are familiar with USB more than FireWire. So the main idea of our project is to use USB.

3.3.1.2 Control Unit Design Options

- **Field-Programmable Gate Array** (FPGA)

A Field-Programmable Gate Array (FPGA) is a semiconductor device that can be configured by the customer or designer after manufacturing hence the name "field-programmable". FPGAs can be used to implement any logical function that an application-specific integrated circuit (ASIC) could perform, FPGAs contain

programmable logic components called "logic blocks", Logic blocks can be configured to perform complex combinational functions, or simple logic gates like AND gate and XOR gate with memory elements, which may be simple flip-flops or more complete blocks of memory as shown in Fig. 3.1 , FPGA has advantages such as:

- Ability to re-program in the field to fix bugs.
- They are available.
- Lower non-recurring engineering costs.

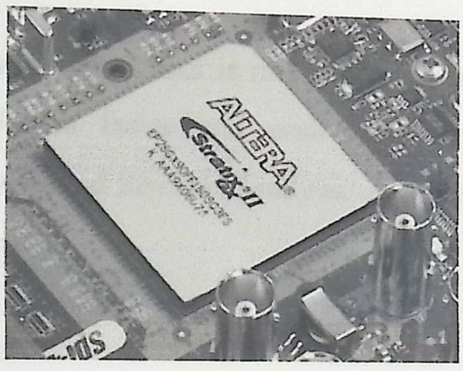


Figure (3.1) FPGA

But there are some disadvantages for FPGA such as:

- They are usually slower than their fixed application-specific integrated circuit (ASIC) counterparts.
- They need draw more power.
- Generally achieve less functionality using a given amount of circuit complexity.

- **Microprocessor**

A **microprocessor** is a computer processor on a microchip . A microprocessor is designed to perform arithmetic and logic operations that make use of small number-holding areas called registers. A microprocessor is always reliant on external devices like memory, clock oscillator, I/O interfaces, serial interfaces. Typical microprocessor operations include adding, subtracting, comparing two numbers, and fetching numbers from one area to another.

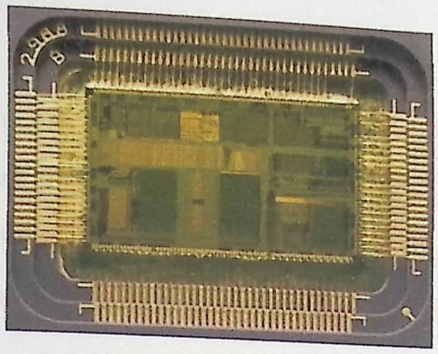


Figure (3.2) Microprocessor

The design process on microprocessor is much easier compared to FPGA. But the implementation with FPGA is faster and easier than microprocessor since the computation in FPGA is distributed instead of only done by the processor.

- **Microcontroller:**

A microcontroller is a computer on a single chip used to control electronic devices. It is a type of microprocessor emphasizing self-sufficiency and cost-effectiveness, in contrast to a general- purpose microprocessor. A typical microcontroller contains all the memory and interfaces needed for a simple application, where as a general purpose microprocessor requires additional chips to provide these functions.

Here we list two options of microcontrollers that can be used to implement this project: ATmega32 and PIC 18F4550.

- **ATmega32 :**

The ATmega32 is a low power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture, by executing powerful instructions in a single clock cycle , the ATmega32 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed[14].

1. On chip analog comparator.
2. 8-channel, 10-bit ADC.
3. Easy to build.

But the problem in choosing this PIC is the difficulty in programming because it needs a special external component as interface to USB, since it is not compatible with the USB so this component is not available and hard to programming

- **PIC 18F4550:**

This project uses PIC18F4550. Its main features were explained in chapter two; we noticed that they are very suitable and compatible with USB, the most important features which enthused us to choose PIC18F4550 are:

1. USB V2.0 Compliant
2. On-chip USB transceiver with on-chip voltage regulator.

3.3.2 Software Design Options

Here we list two alternative choices for the software to be used for the system, which are Visual C++, and VB.NET, with some of their features.

3.3.2.1 Microsoft Visual Studio 2008 Express

The Microsoft Visual Studio 2008 Express Editions with SP1 are a free set of tools that are simple, fun and easy to learn. Visual C++ 2008 Express Edition with SP1 is the development environment for creating native Windows applications that deliver the highest quality rich user experiences, some important features are:

- An easy installation of the Windows Platform SDK.
- It has tools for developing and debugging C++ code.
- Free and small size.
- Allows creating Win32 and Win64 applications.

3.3.2.2 VB.NET

We have an alternative choice to make interface between the user and the computer which is VB.NET ,it is considered as one of the most popular languages for interfacing applications with PC ,the most important features related to it are:

- Fast.
- Familiar with application.

In this project Visual C++ attended to be used, as it is easy to use and it is free available from Microsoft.

3.4 Realization Approach

As we mentioned, the main goal of this project is to design a USB 2.0 interface that can be generalized for various applications, providing a platform that can be easily adapted for users.

A general block diagram of the system is shown in figure (3.3) the block diagram depicts the overall system architecture.

The design approach for the platform divides the implementation into three primary elements: PIC18f4550 microcontroller, PC host, and applications part.

3.5 General Block diagram

Figure (3.3) is a general block diagram shows the main components of the system includes (PC ,USB port , PIC18F4550 , and the applications part which could be input or output circuits), and how they can interact with each others in a systematic way to achieve the main purpose of the system .

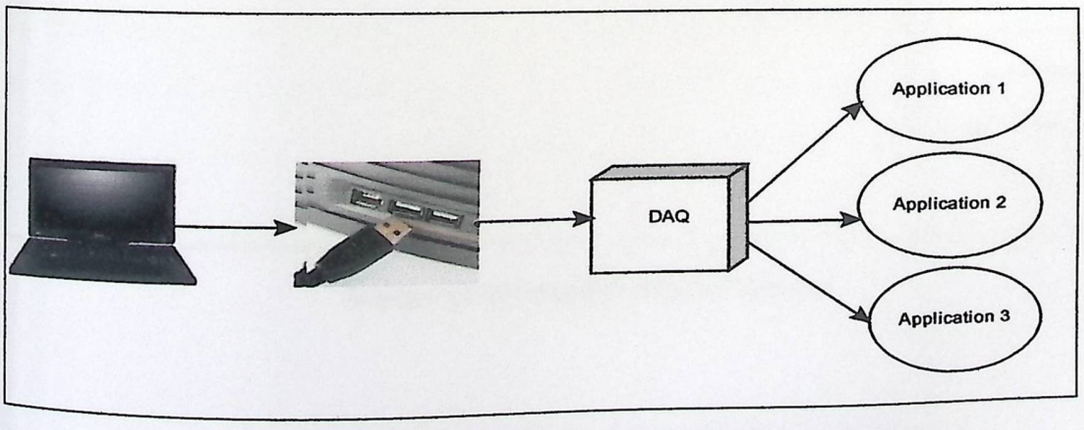


Figure (3.3) General Block Diagram

3.5.1 Detailed Block Diagram

The block diagram in figure(3.4) figures some details about the system which can follow the way the system works ,starting from sending user command by GUI through USB port to the PIC microcontroller inorder to control the functionality of various applications which is built using suitable conditioning and optocouplers circuits .

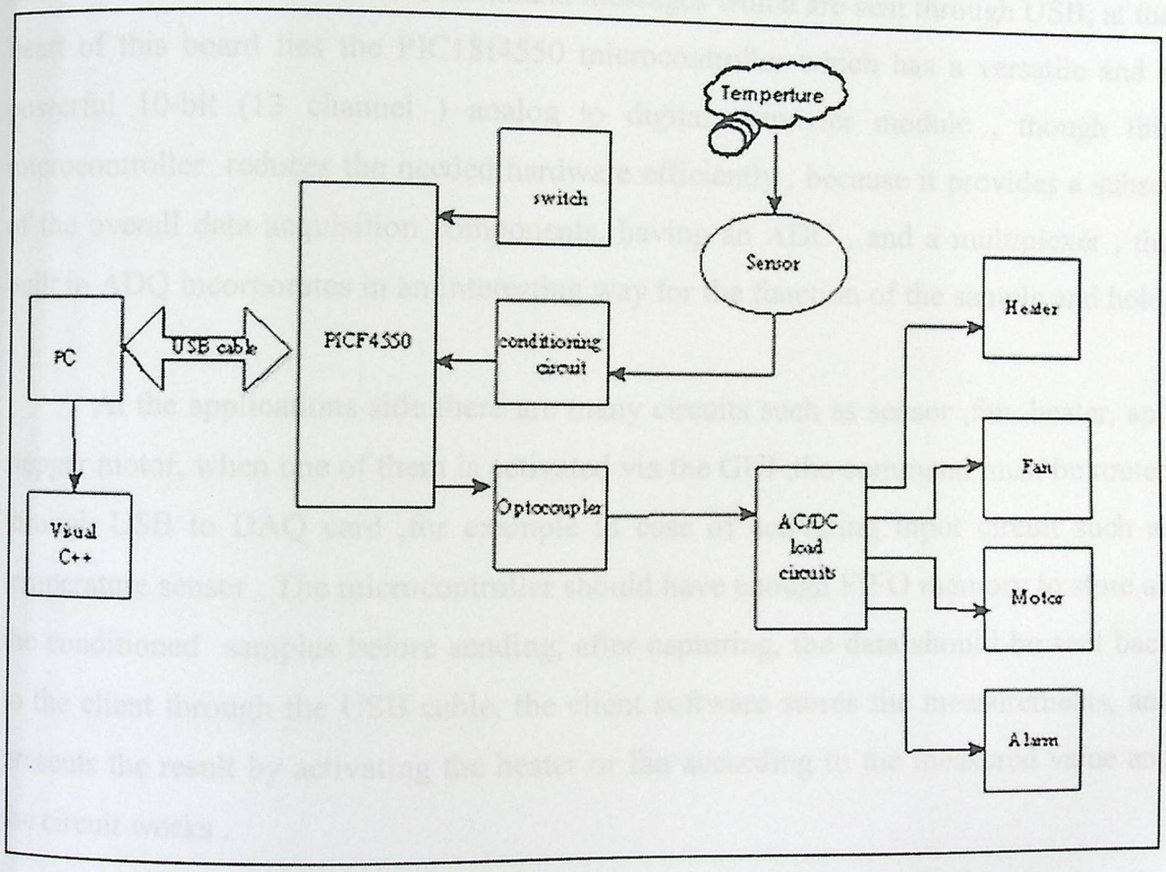


Figure (3.4) Detailed Block Diagram

3.6 The system work

The USB data acquisition card should be able to capture a fixed number of samples.

Since we aim to build a trainer data acquisition, so the system must be supported with a GUI that enable users to control the applications provided ,so a GUI was built with Visual C++ Express programming language , to enable the user to activate a certain application from the group of the applications presented, The card must be remote controlled with command messages which are sent through USB, at the heart of this board lies the PIC18f4550 microcontroller which has a versatile and a powerful 10-bit (13 channel) analog to digital converter module , though this microcontroller reduces the needed hardware efficiently , because it provides a subset of the overall data acquisition components, having an ADC , and a multiplexer , the built in ADQ incorporates in an interesting way for the function of the sample and hold.

At the applications side there are many circuits such as sensor ,fan ,heater, and stepper motor, when one of them is activated via the GUI ,the command must be routed through USB to DAQ card ,for example in case of activating input circuit such as temperature sensor , The microcontroller should have enough FIFO memory to store all the conditioned samples before sending, after capturing, the data should be sent back to the client through the USB cable, the client software stores the measurements, and presents the result by activating the heater or fan according to the measured value and the circuit works .

Chapter

4 DETAILD SYSTEM DESIGN

4.1 Preface.

4.2 Project Phases.

4.3 Schematic design.

4.4 General schematic design for USB board.

4.5 Applications presented.

4.6 Overall system schematic design.

Chapter Four
Detailed system design

4.1 Preface

This chapter describes the main hardware components that are used during the building of the system , also it represents block diagrams and general schematic diagrams which illustrate how these components are related to each others.

4.2 Project Phases:

- **Input Phase**

USB data acquisition card should be able to capture a fixed number of samples and passes the signals through USB cable to PC; this system has eight digital inputs and eight 10-bit analogue inputs, so both analogue and digital input circuits can be supported, this project includes switches, and the LM35 temperature sensor- that is used to capture temperature signals- as input circuits to the system, the command input can be send from GUI.

- **Processing Phase**

The signals that are captured from the input circuits pass through different stages to be processed, these stages include:

- Conditioning Circuits for amplifying and filtering sensor's signals, by using appropriate operational amplifiers.

- The input signals will be processed by PIC18f4550 microcontroller, which includes built in circuits for acquiring signals including ADC, multiplexers, sample and hold, etc …

- **Output Phase**

  - Signals are sent from the PIC18f4550 microcontroller through the USB cable to PC , the GUI using Visual C++ software will enable the user to control the functionality of the system and to choose the a specific application to turn on or off.
  - Since this data acquisition card has eight digital outputs, and two 10-bit analogue outputs, it can control many digital and analogue output circuits; here we presented the fan circuit, motor, alarm, and heater as examples of output applications.

4.3 Schematic design

This section illustrates a detailed schematic design for the major components that make up this project as well as the interfacing schematic design for all of these components.

The components are:

- PIC 18F4550 microcontroller.
- LM35 Temperature Sensor.
- Conditional circuit.
- Alarm system.
- Heater and fan
- USB cable.
- Stepper motor.
- ULN2803.

4.3.1 PIC 18f4550 Microcontroller

The project uses PIC18f4550 to control the functions of the system .This microcontroller is packed into a 40 pin package as shown in Figure (4.1).

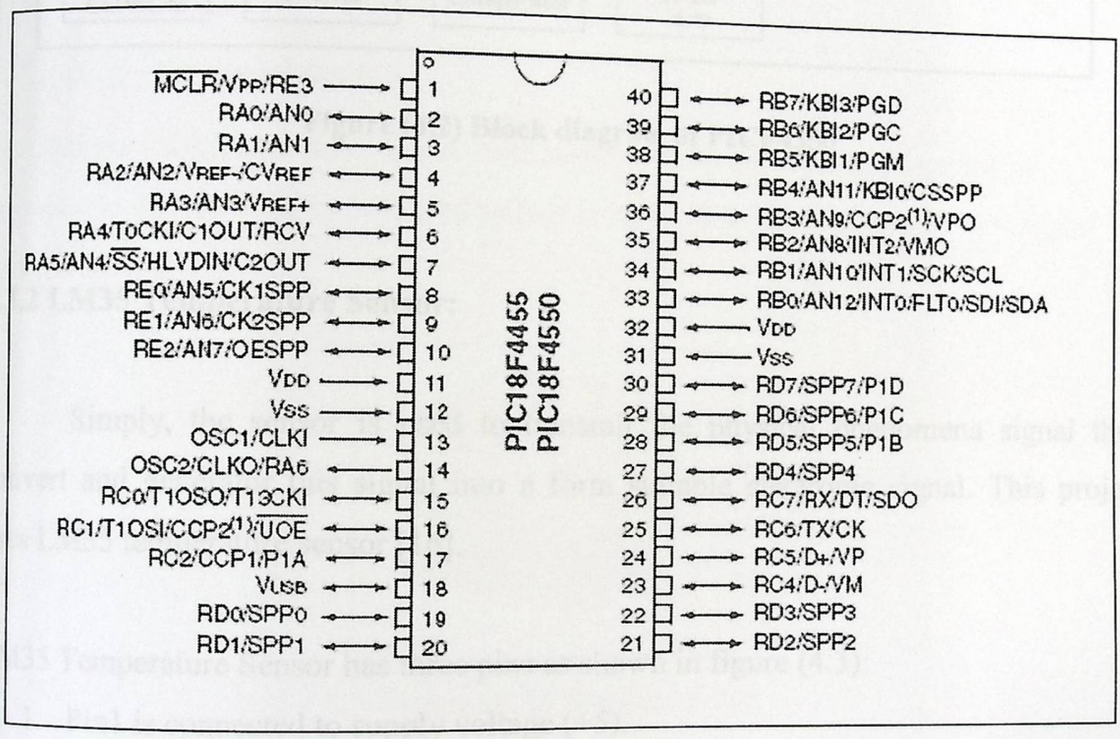


Figure (4.1) PIC18f4550

The microcontroller receives any signal from the input circuits, for example after the temperature sensor's signal is amplified, this signal is compared to a reference by comparator's microcontroller, if it is matched, then converted into a fraction which is then represented as a coded digital number and sent to the computer, until using the converted signal for controlling the alarm and heater circuits .Figure (4.2) shows the block diagram of PIC18f4550.

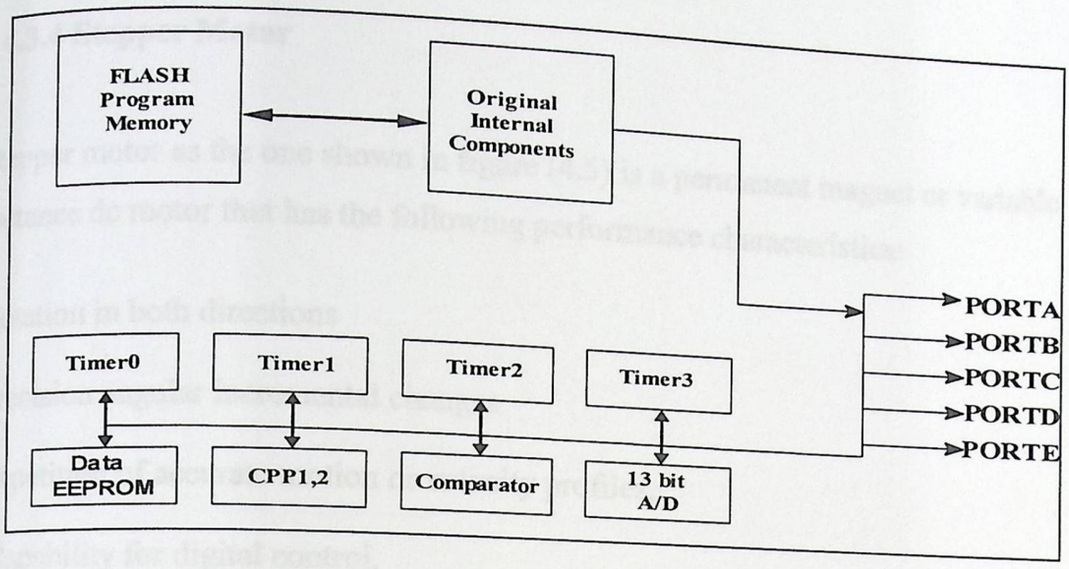


Figure (4.2) Block diagram of PICF4550

4.3.2 LM35 Temperature Sensor:

Simply, the sensor is used to transmit the physical phenomena signal then convert and generator this signal into a form suitable electronic signal. This project uses LM35 temperature sensor [16].

LM35 Temperature Sensor has three pins as shown in figure (4.3):

1. Pin1 is connected to supply voltage (+5).
2. Pin2 is connected to the ground.
3. Pin3 is a voltage output which connected with the input of conditional circuit.

4.3.4 Stepper Motor

A stepper motor as the one shown in figure (4.5) is a permanent magnet or variable reluctance dc motor that has the following performance characteristics:

1-Rotation in both directions

2-Precision angular incremental changes

3-Repetition of accurate motion or velocity profiles,

4- Capability for digital control.

5- A stepper motor can move in accurate angular increments known as steps in response to the application of digital pulses to an electric drive circuit from a digital controller.

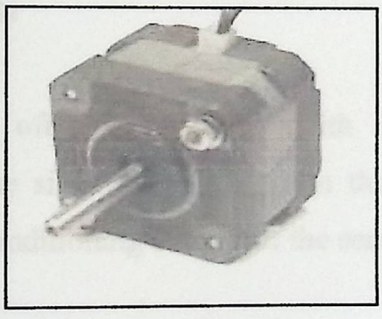


Figure (4. 5) Stepper Motor

- **ULN2803**

The ULN2803APG / AFWG Series are high voltage, high current Darlington drivers comprised of eight NPN Darlington pairs. All units feature integral clamp diodes for switching inductive loads. Applications include relay, hammer, lamp and display (LED) drivers. Figure (4.6) shows ULN2803 IC.

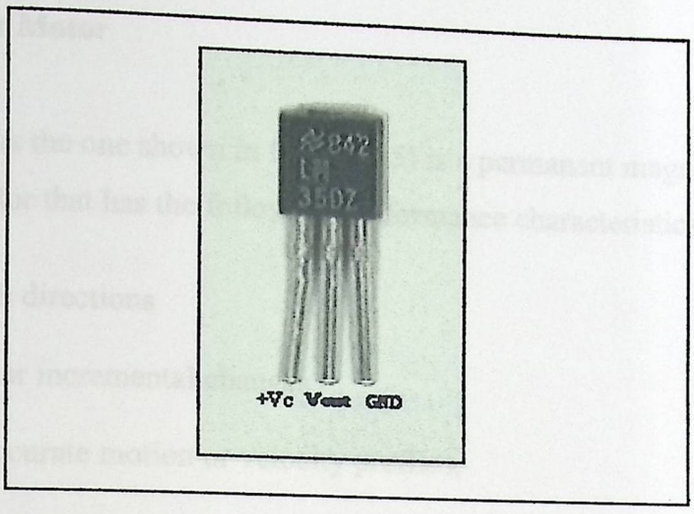


Figure (4.3) LM35 Temperature

4.3.3 Conditioning Circuits

Sensor signals are often incompatible with data acquisition hardware. To overcome this problem, the signals obtained from the sensors must be conditioned .Figure (4.4) displays the conditioning circuit for the sensor.

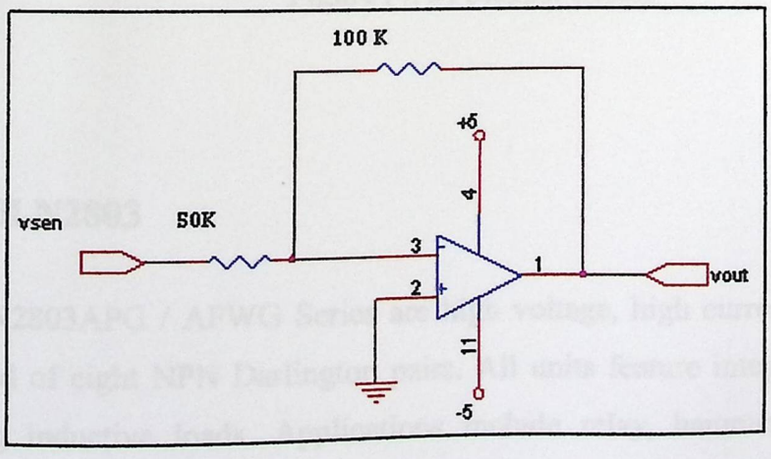


Figure (4.4) conditioning circuit

4.3.4 Stepper Motor

A stepper motor as the one shown in figure (4.5) is a permanent magnet or variable reluctance dc motor that has the following performance characteristics:

1-Rotation in both directions

2-Precision angular incremental changes

3-Repetition of accurate motion or velocity profiles,

4- Capability for digital control.

5- A stepper motor can move in accurate angular increments known as steps in response to the application of digital pulses to an electric drive circuit from a digital controller [24].

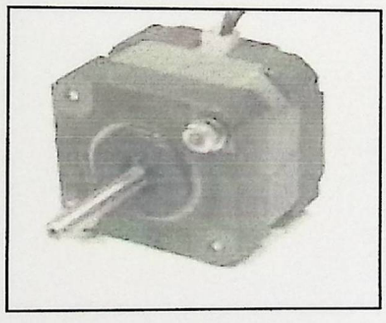


Figure (4. 5) Stepper Motor

- **ULN2803**

The ULN2803APG / AFWG Series are high voltage, high current Darlington drivers comprised of eight NPN Darlington pairs. All units feature integral clamp diodes for switching inductive loads. Applications include relay, hammer, lamp and display (LED) drivers. Figure (4.6) shows ULN2803 IC [23].

- **Features**

- Output current (single output) 500 mA (Max.)
- High sustaining voltage output 50 V (Min.)
- Output clamp diodes

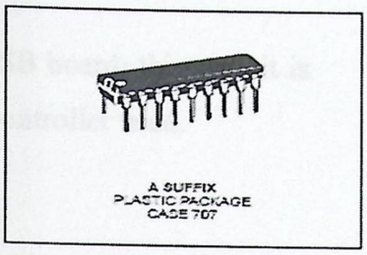


Figure (4.6) ULN2803

4.4 General Schematic Design for USB Board

Figure (4.7) shows the schematic design for building USB board; this circuit is general for all USB projects regardless of the type of the microcontroller used.

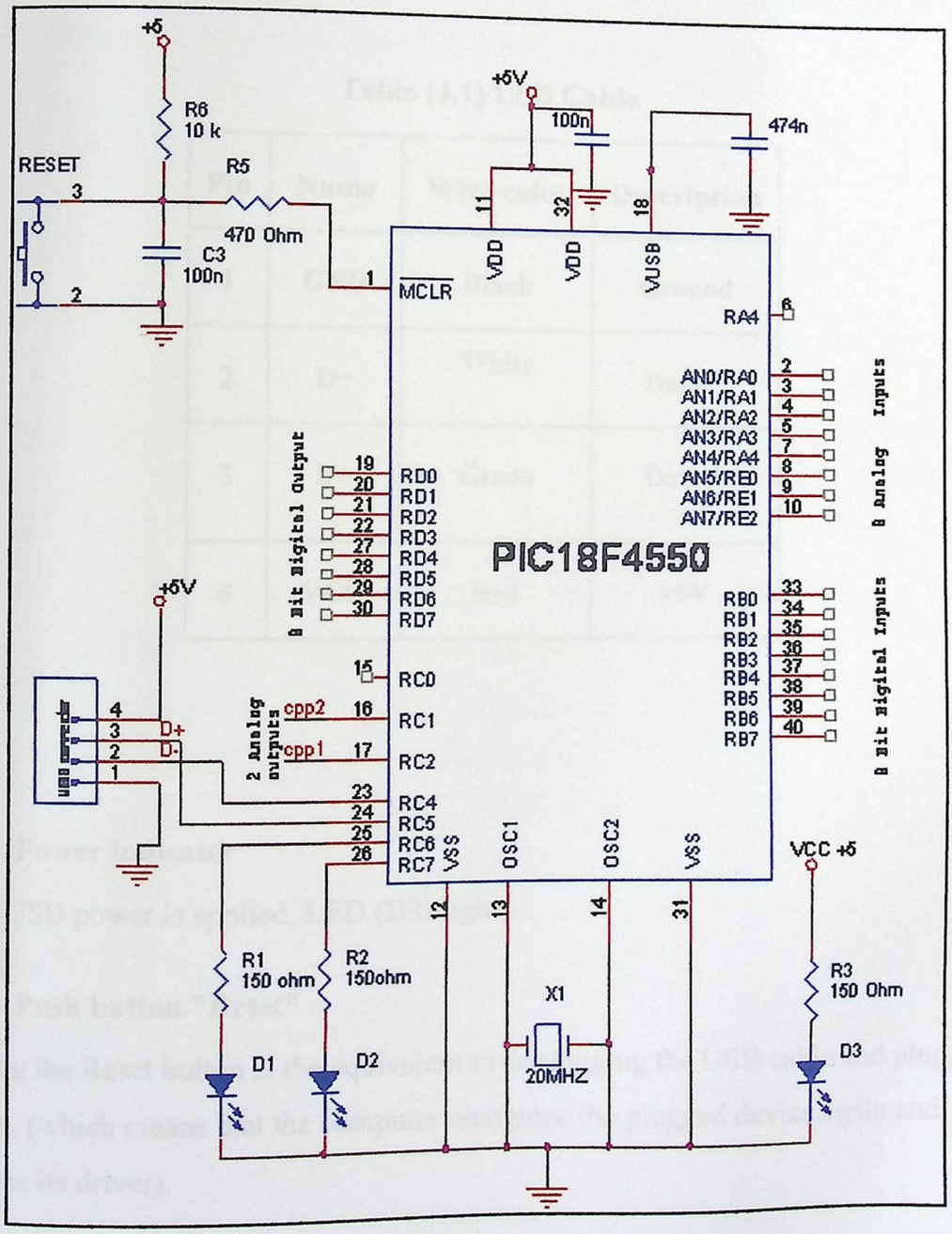


Figure (4.7) Main USB Board

- **Major components of the circuit:**

- **USB Connector**

The USB Connector is a standard "type B" connector. There are four connections in a USB cable; table (4.1) shows the internal structure and wires of USB cable.

Table (4.1) USB Cable

Pin	Name	Wire color	Description
1	GND	Black	Ground
2	D−	White	Data −
3	D+	Green	Data +
4	VCC	Red	+5V

- **Power indicator**

When USB power is applied, LED (D3) lights.

- **Push button "Reset"**

Pressing the Reset button is the equivalent to unplugging the USB cable and plugging it back in, (which means that the computer recognize the plugged device again and initialize its driver).

- **20MHz Crystal oscillator**

Provides a clock signal to the PIC, this frequency was chosen since it's the same as the evaluation board from Microchip.

- **USB state indicator**

Tow leds indicate the USB state, once the card is programmed and connected to the PC one of the leds (D1, D2) flashes while the other remains off.

4.5 Ports Assignment

Table (4.2) shows the I/O ports on PIC18F4550 and how they are configured for this project.

Table (4. 2) Summary of the I/O ports on PIC18F4550

RA0-RA7	8 bits for analog Input
RB0-RB7	8 bits for digital Input
RD0-RD7	8 bits for digital output
RC2	analog outputs (for PWM1)
RC4,RC5	Connected to USB D- ,D+ respectively

4.6 Applications presented

4.6.1 Alarm system using simple buzzer

As the name implies, it is a simple circuit that has only a simple buzzer connected with a transistor. This buzzer when triggered by PIC18F4550 gives a special sound. The buzzer has two edges, one is connected to Vcc (3V) and the other is connected to a transistor .this circuit is connected to one of the digital output bit of PORTB .The buzzer circuit is shown in figure (4.7).

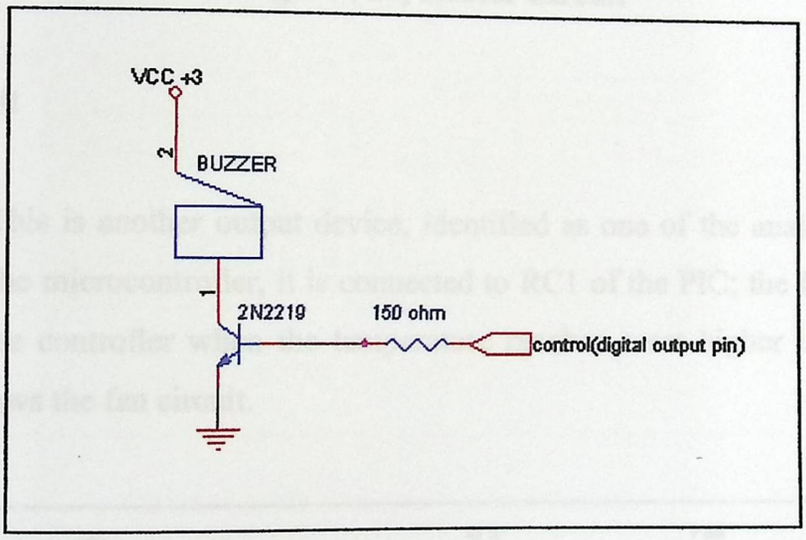


Figure (4.7) Simple Buzzer Circuit

4.6.2 Heater circuit

This is output device, identified as one of the output digital port of the microcontroller it is connected to one of the pins of PORTB; the heater is turned on by the controller when the temperature reaches a set lower limit.

Figure (4.8) shows the heater circuit:

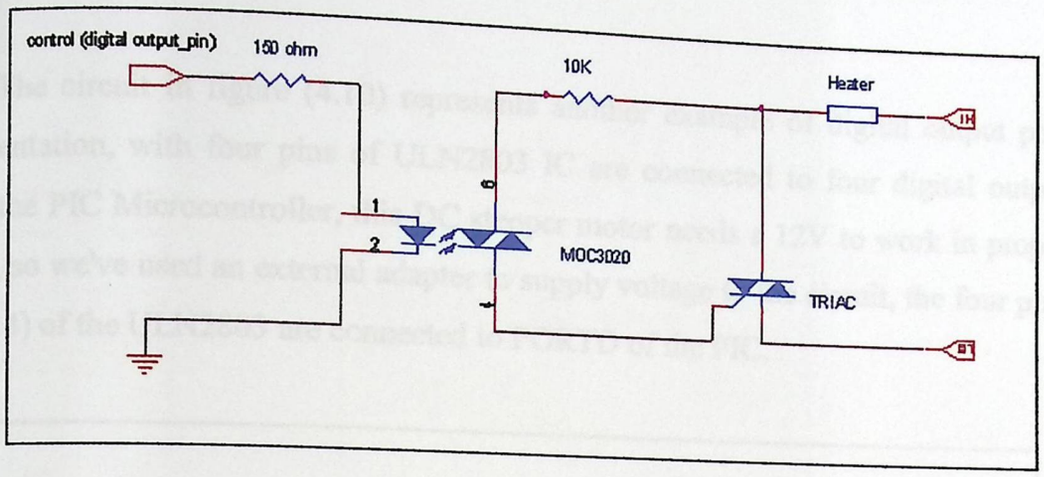


Figure (4.8) Heater Circuit

4.6.3 Fan circuit

This is another output device, identified as one of the analogue output port of the microcontroller, it is connected to RC1 of the PIC; the fan is turned on by the controller when the temperature reaches a set higher limit. Figure (4.9) shows the fan circuit.

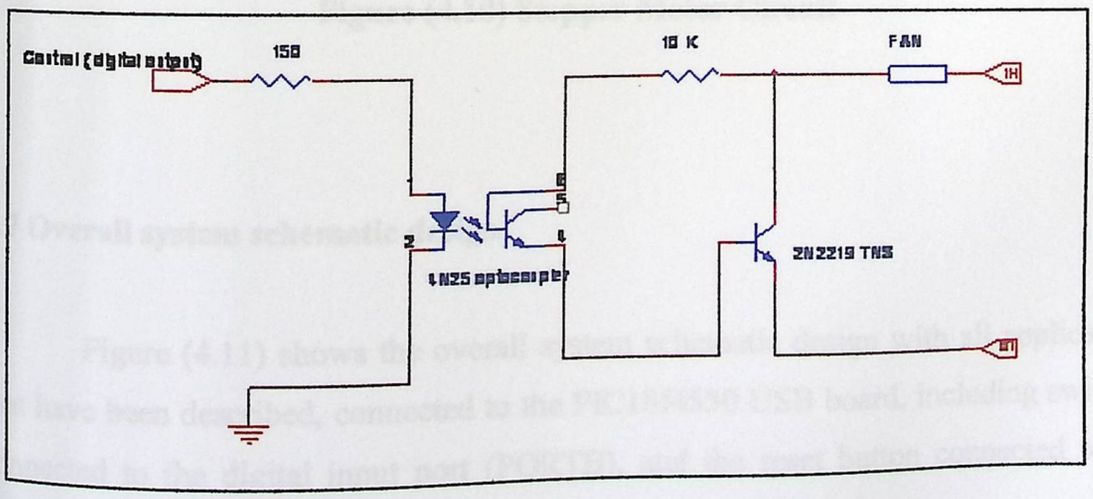


Figure (4.9) Fan Circuit

4.6.4 Stepper Motor Circuit

The circuit in figure (4.10) represents another example of digital output port implementation, with four pins of ULN2803 IC are connected to four digital output pins of the PIC Microcontroller, this DC stepper motor needs a 12V to work in proper manner, so we've used an external adapter to supply voltage to the circuit, the four pins (1, 2, 3, 4) of the ULN2803 are connected to PORTD of the PIC.

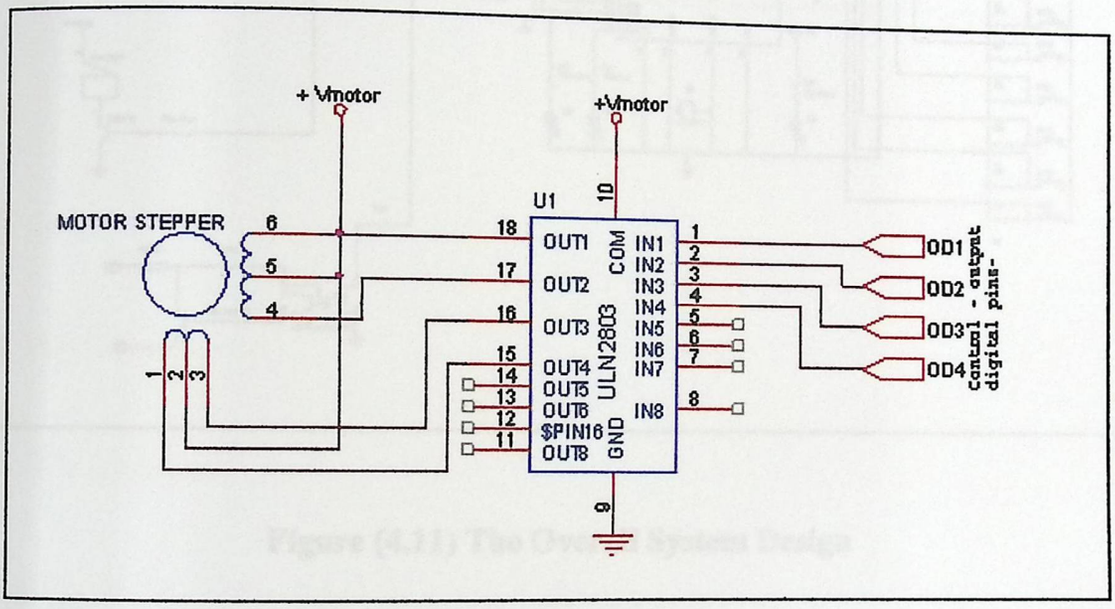


Figure (4.10) Stepper Motor Circuit

4.7 Overall system schematic design

Figure (4.11) shows the overall system schematic design with all applications that have been described, connected to the PIC18f4550 USB board, including switches connected to the digital input port (PORTB), and the reset button connected to the MCLR pin.

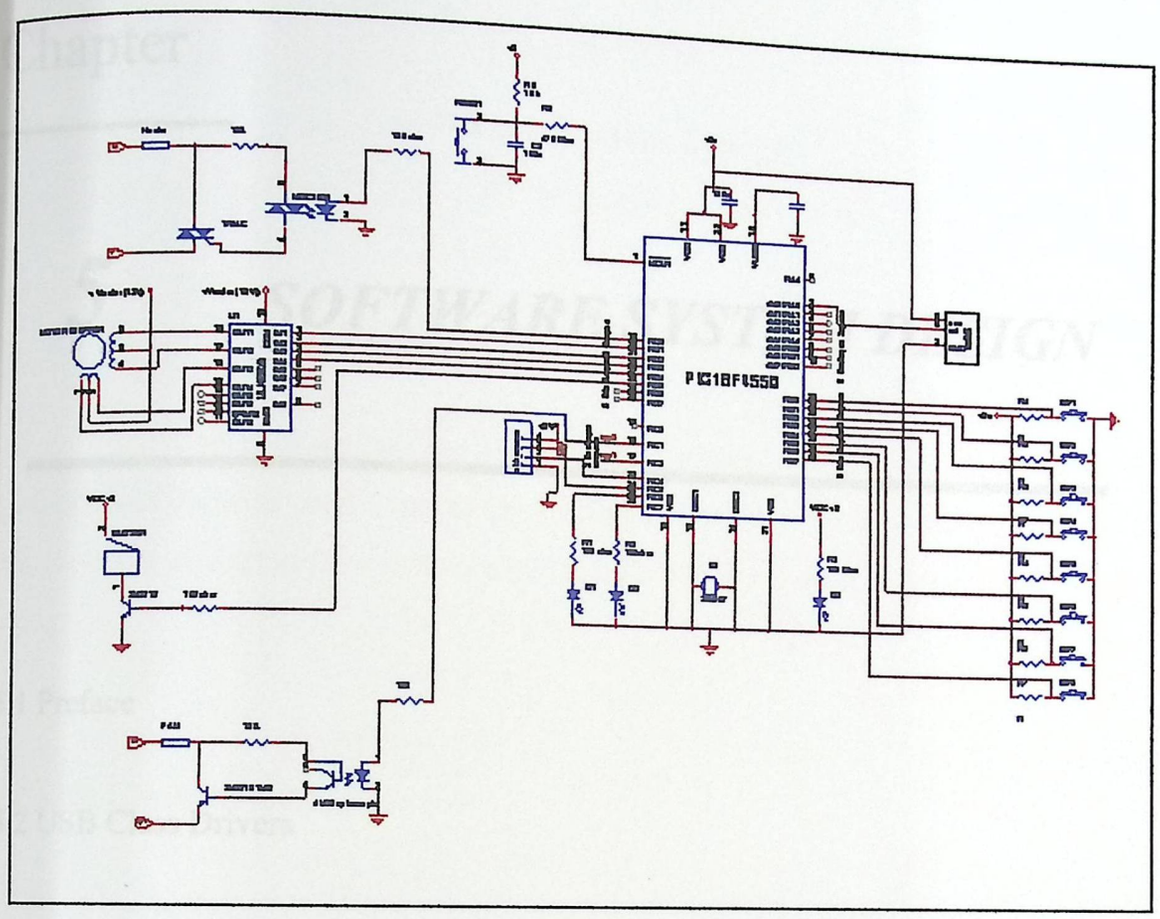


Figure (4.11) The Overall System Design

Chapter

5 SOFTWARE SYSTEM DESIGN

5.1 Preface

5.2 USB Class Drivers

5.3 USB Libraries.

5.4 MPLAB IDE Version 8.0.

5.5 Microsoft Visual C++ Express 2008

5.6 Enumeration about Devices.

5.7 System Flowcharts.

5.8 Graphical User Interface.

Chapter Five

Software System Design

5.1 Preface

This chapter describes the basic firmware used to program PIC18f4550, the methods and libraries used to make the bindings between firmware and the GUI; also it describes the overall system software and the development environment.

5.2 USB Class Drivers

Windows supports several USB classes that the USB Device Working Group (DWG) has defined such as Human Interface Device (HID), Communication Device Class (CDC), Mass storage class (MSC), printing class and others, if a user wanted to develop a driver for a new application that doesn't match any of the predefined USB classes, so he need to write what is called a custom class, for more information you can read the "USB Complete "book.

Here we illustrate the HID class because it is the most popular one, the CDC class, and Microchip General Purpose (Custom Class) USB Driver which we depend on and make the needed modifications upon it for implementing this project.

5.2.1 Human Interface Devices (HID)

Human Interface Devices (HID) is a class of USB devices that gives structure to the data that will be transferred between the device and the host computer. During the enumeration process, the device describes the information that it can receive and send. This allows a host computer to handle the data being received from the USB device without requiring a specially designed device driver [22].

The **HID** class consists primarily of devices that are used by humans to control the operation of computer systems. Typical examples of **HID** class devices include:

- Keyboards and pointing devices—for example, standard mouse devices, trackballs, and joysticks.
- Front-panel controls—for example: knobs, switches, and buttons.
- Devices that may not require human interaction but provide data in a similar format to **HID** class devices—for example, bar-code readers, thermometers, or voltmeters.

5.2.2 USB Communications Device Class (USB-CDC)

Using a relatively simple driver information file (*.inf), the communications device class (CDC) allows your device to talk to the host PC via a "virtual COM port". This means that your USB device appears to your PC application software as a serial ("RS232") device, just like a physical COM port, but faster. This is probably the simplest method to implement a fast USB connection. The maximum data transfer rate

using USB CDC with a "virtual COM port" is about 1Mbit/sec. (This is a Windows driver limitation, not a USB limitation.)

5.2.3 Microchip General Purpose (Custom Class) USB Driver

Microchip provides a general purpose Windows driver which can be used by Windows applications to interface with a custom class USB device. For USB applications that do not readily fit within the constraints of these other device class options, Microchip general purpose driver may be used, so for the data acquisition card we depend on the general driver and make the required modifications, to suite the needed applications for our project.

5.3 USB libraries

Perhaps the most time intensive portion of this project was spent in choosing the best USB library. There are several choices and each varied in stability, functionality, and extensibility.

In general the chosen library should achieve the following:

- It should be easy to use and few lines of client code should accomplish much
- It should be as platform independent as possible, and it should have C++ bindings.

Here we discuss two alternative libraries which are the" MPUSBAPI" Library and "libusb-win32" library including their obvious features

5.3.1 Libusb-win32

Libusb-win32 is an open-source USB library to provide user space access to USB devices. It supports Linux 2.6/2.4/2.2, FreeBSD/OpenBSD/NetBSD, and Darwin/MacOS X.

Libusb-win32 has also the filter driver, which is not working for Vista and may cause problems. Libusb-win32 has the counterpart libusb for Linux, Mac OS X, BSDs and Solaris, so it has some advantages for cross-platform projects; it also supports isochronous transfer which is not supported using the "MPUSBAPI.DLL" library.

5.3.2 MPUSBAPI Library and DLL Source

The MPUSBAPI.DLL file is a library which provides a number of functions including the basic ones needed for reading and writing to a USB device. A list of the functions available and the calling conventions for those functions is currently documented in the form of inline comments in the source code for the DLL file.

This library works in conjunction with the USB device driver MCHUSB.SYS. The DLL is compiled using Borland® C++ Builder™ 6 development environment, and the source code is provided in the:
"<installdirectory>\Microchip\USB\Utilities\MCHPUSB Custom Driver \Mpusbapi\ Dll\Borland_C\Source" directory.

Since this library achieves most of the needed functionality that were mentioned ,also because of the availability of the examples that demonstrates the use of Visual C++ with Microchip's custom driver and DLL ,we attended to use it in this project to read and write from the USB[22].

5.4 MPLAB IDE version 8.0

MPLAB Integrated Development Environment (IDE) is a free, integrated toolset for the development of embedded applications employing Microchip's PIC and microcontrollers. MPLAB IDE runs as a 32-bit application on MS Windows, it is easy to use and includes a host of free software components for fast application development. In this project we developed the firmware using MPLAB IDE version 8.0; figure (5.1) gives simple view of MPLAB interface[6].

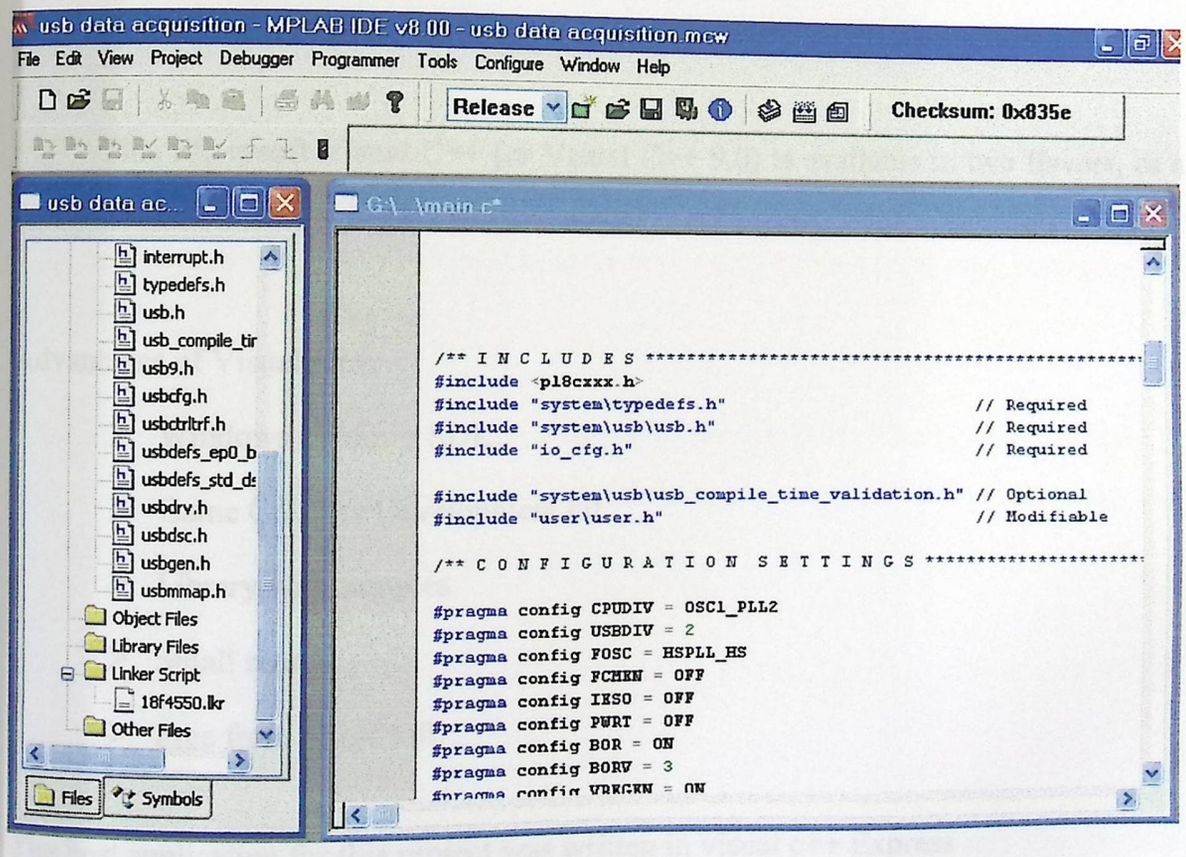


Figure (5.1) MPLAB IDE work space

5.5 Microsoft Visual C++ 2008 Express

The Microsoft Visual Studio 2008 Express Editions with SP1 are a free set of tools that are simple, fun and easy to learn. Visual C++ 2008 Express Edition with SP1 is the development environment for creating native Windows applications that deliver the highest quality rich user experiences. This new version includes an easy installation of the Windows Platform SDK. It has tools for developing and debugging C++ code, especially code written for the Microsoft Windows API, the DirectX API, and the Microsoft .NET Framework. This program allows creating Win32 and Win64 applications[21].

This Microsoft Visual C++ (or Visual C++ 9.0) is available in two flavors, as a part of Microsoft Visual Studio and as a standalone "Express Edition" product.

Advantages of Visual C++:

- Windows platform SDK.
- Game Creators Development kit.
- Library with samples.
- Small size.
- Free from Microsoft.

The host application for this project was written in visual c++ Express

5.6 Enumeration about Devices

Enumeration is the initial exchange of information that enables the host to learn about the device and assign a device driver .The process includes assigning an address

to the device, reading data structures from the device, assigning and loading a device driver, and selecting a configuration from the options presented in the retrieved data. The device is then configured and ready to transfer data using any of the endpoints in its configuration [1].

When a USB device is **attached** to a powered port, the following actions are taken:

1. The hub to which the USB device is now attached informs the host of the event via a reply on its status change pipe At this point, the USB device is in the **Powered state** and the port to which it is attached is disabled.

2. Now that the host knows the port to which the new device has been attached, the host then waits for at least 100 ms to allow completion of an insertion process and for power at the device to become stable. The host then issues a port enable and reset command to that port.

3. The hub performs the required reset processing for that port. When the reset signal is released, the port has been enabled. The USB device is now in the **Default state** and can draw no more than 100 mA from VBUS.

4 The host assigns a unique address to the USB device, moving the device to the **Address state.**

5 Before the USB device receives a unique address, its Default Control Pipe is still accessible via the default address. The host reads the device descriptor to determine what actual maximum data payload size this USB device's default pipe can use.

6. The host reads the configuration information from the device by reading each configuration zero to n-1, where n is the number of configurations.

7. The host assigns a configuration value to the device. The device is now in the **Configured state**. The USB device may now draw the amount of VBUS power described in its descriptor for the selected configuration. From the device's point of view, it is now **ready** for use.

5.7 System Flowcharts

This section explains the function of the program to make the system work properly. DAQ card has four parts (digital input, digital output, analog input, analog output) each part has a flowchart and an algorithm to control the function of it.

At the beginning Windows should prompt the user for a driver when the device first enumerates, Figure (5.2) shows the flowchart of installing driver of DAQ, this process requires matching the numbers of VID (04D8) and PID (000C) with the numbers correlated with data in the drivers install file mchpusb.inf.

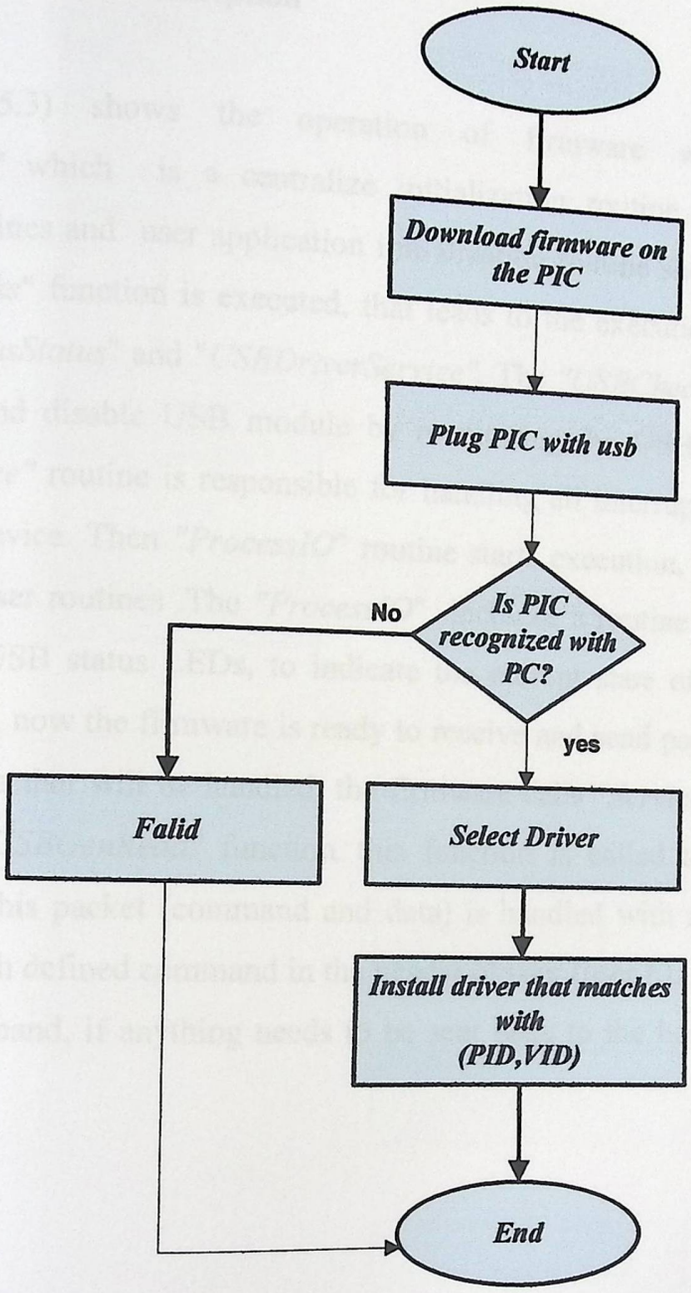


Figure (5.2) Flowchart for installing driver of DAQ

- **Firmware Description**

Figure (5.3) shows the operation of firmware which starts with *"InitializeSystem"* which is a centralize initialization routine, all required USB initialization routines and user application initialization routine should be called from it ,next "*USBTasks*" function is executed, that leads to the execution of two routines the *"USBCheckBusStatus*" and "*USBDriverService"*. The *"USBCheckBusStatus*" routine used to enable and disable USB module by monitoring the USB power signal, the *"USBDriverService"* routine is responsible for handling all interrupts and determining the state of the device. Then *"ProcessIO*" routine starts execution, this routine is the entry point into user routines .The *"ProcessIO*" includes a routine that is responsible for flashing the USB status LEDs, to indicate the current state of the device called *"BlinkUSBStatus"*, now the firmware is ready to receive and send packet which contain command and data that will be handled, the firmware calls *"ServiceRequests"* routine that to execute "*USBGenRead"* function, this function is called to perform a USB packet read then this packet (command and data) is handled with a switch statement with a case for each defined command in the header of user (user.h), After reading and handling the command, if anything needs to be sent back to the host, that happens in "*USBGenWrite* " .

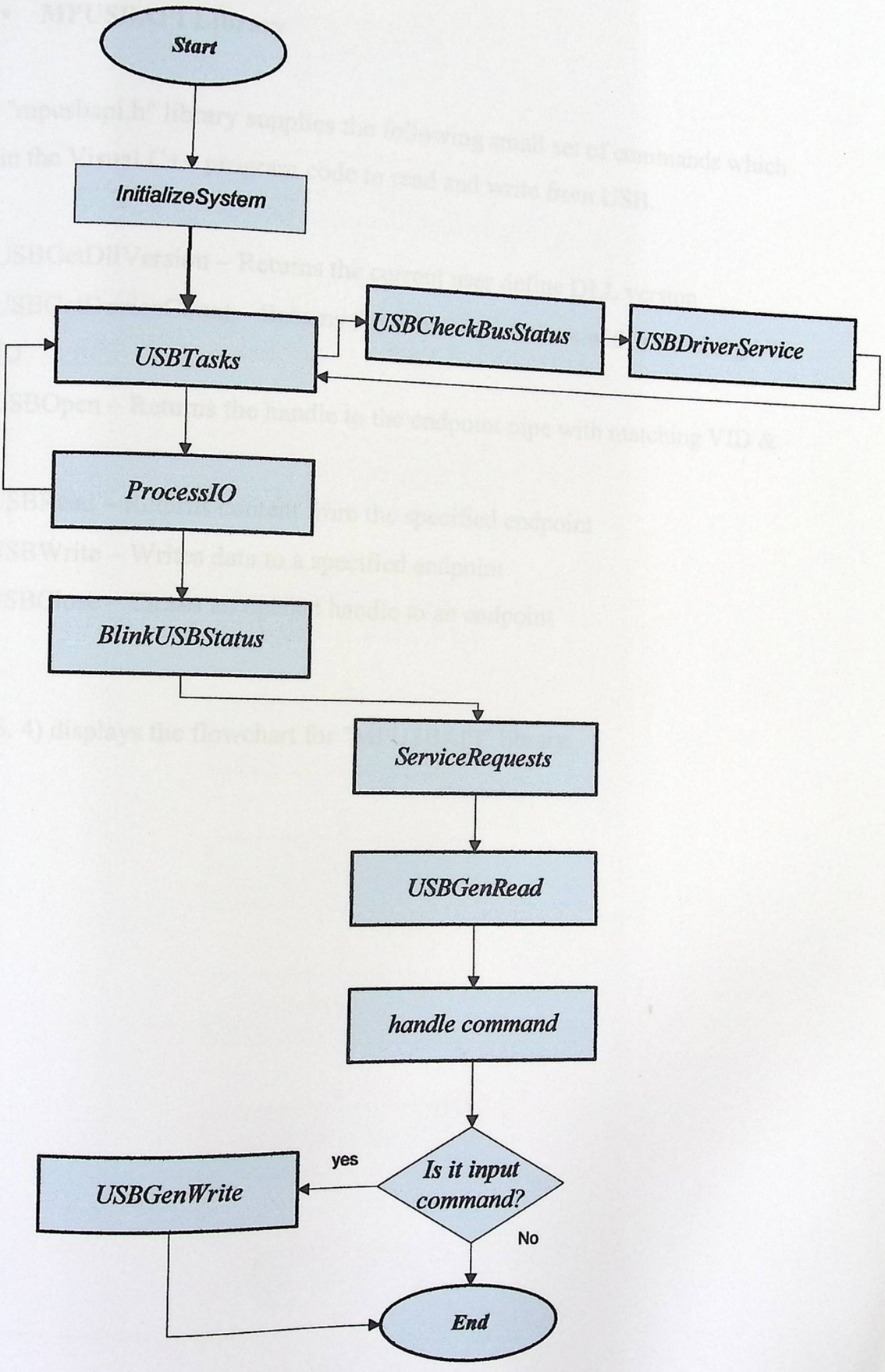


Figure (5.3) firmware Flowchart

- **MPUSBAPI Library**

The "mpusbapi.h" library supplies the following small set of commands which we include in the Visual C++ program code to read and write from USB.

- MPUSBGetDllVersion – Returns the current user define DLL version
- MPUSBGetDeviceCount – Returns the number of devices with a matching VID & PID
- MPUSBOpen – Returns the handle to the endpoint pipe with matching VID & PID
- MPUSBRead – Returns content from the specified endpoint
- MPUSBWrite – Writes data to a specified endpoint
- MPUSBClose – Closes an opened handle to an endpoint

Figure (5. 4) displays the flowchart for "MPUSBAPI" library.

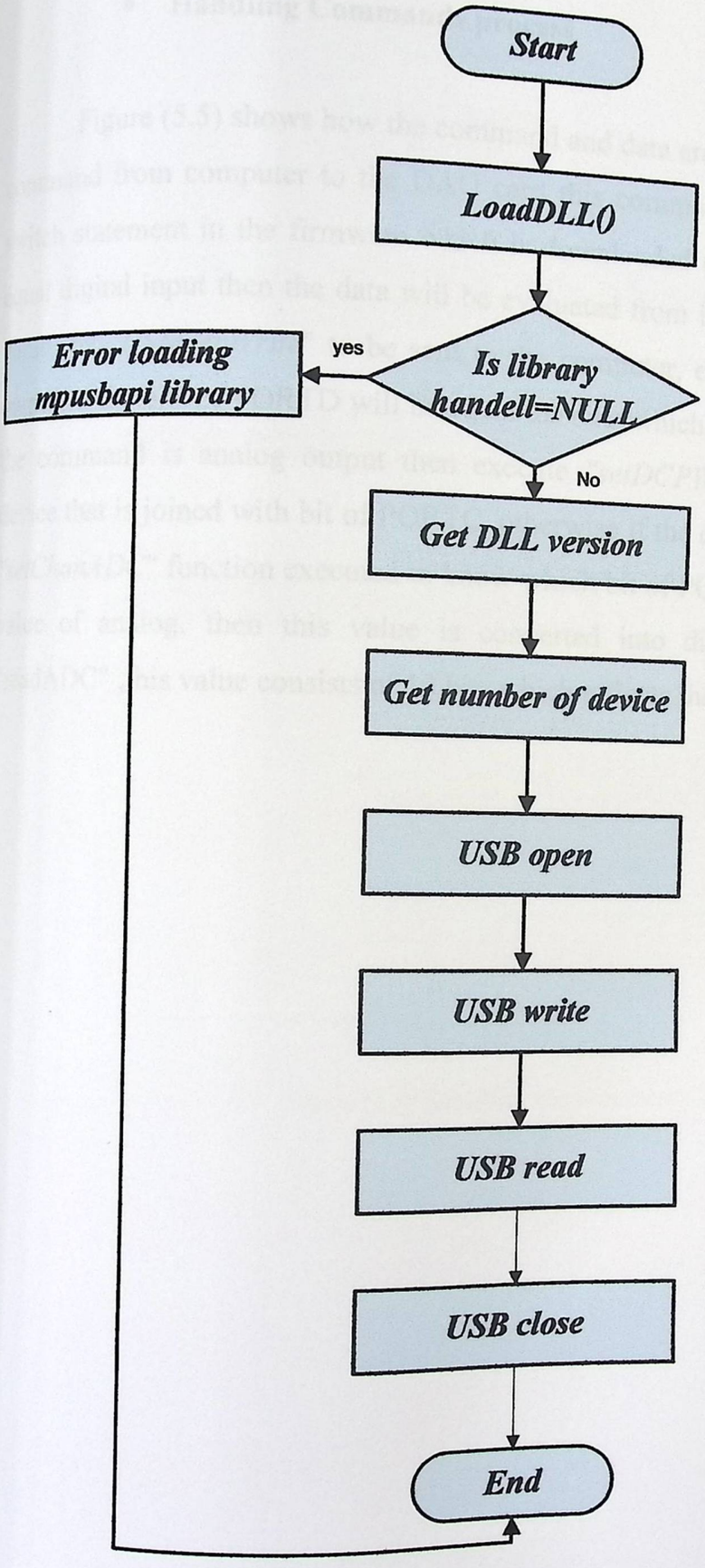


Figure (5.4) Loading MPUSBAPI library Flowchart

- **Handling Commands process**

Figure (5.5) shows how the command and data are handled, after the user send a command from computer to the DAQ card this command is compared with cases of switch statement in the firmware which is downloaded on PIC, so if the command is equal digital input then the data will be evaluated from bits of PORTB then write this value by "*USBGenWrite*" to be sent to the computer, else if the command is digital output so the bits of PORTD will be equal the data which is sent from computer, else if the command is analog output then execute *"setDCPWM1"* function to enable the device that is joined with bit of PORTC ,otherwise if the command is analog input then *"setChanADC"* function executed to know which bit of PORTA is enabled and read the value of analog, then this value is converted into digital value that is read by "readADC" ,this value consists of 10 bits which reflects the resolution of the DAQ .

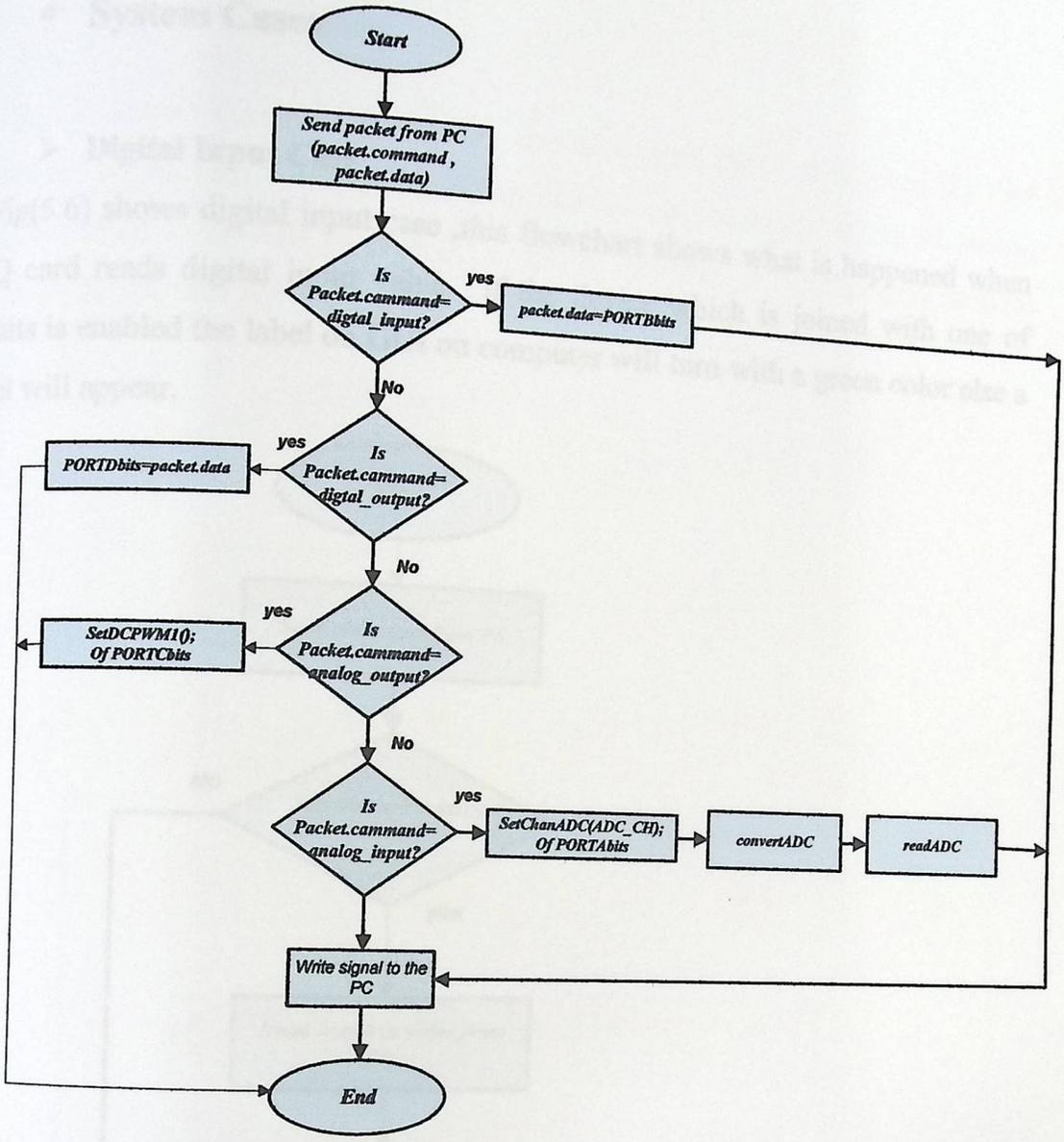


Figure (5.5) Handling the command Flowchart

▪ System Cases

➢ Digital Input Case

Fig(5.6) shows digital input case ,this flowchart shows what is happened when the DAQ card reads digital input value , if the device which is joined with one of PORTBbits is enabled the label on GUI on computer will turn with a green color else a gray label will appear.

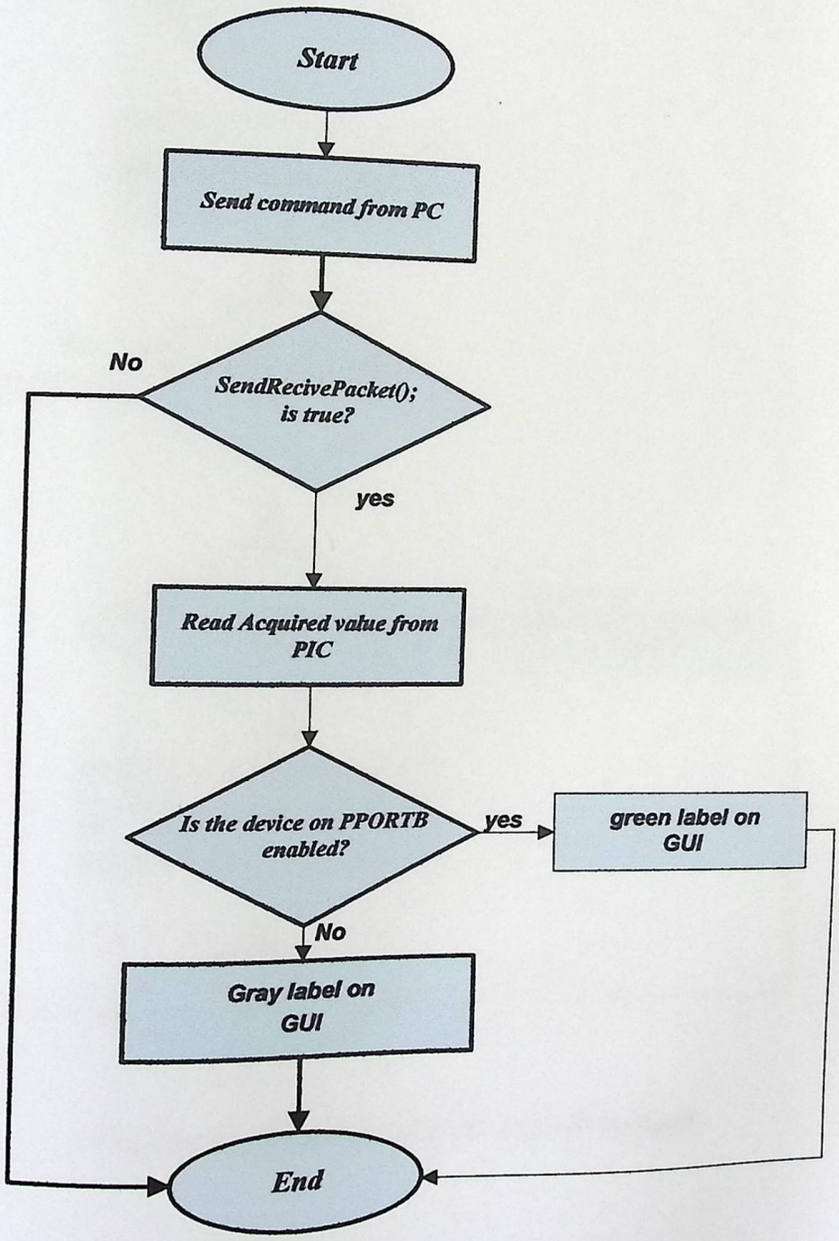


Figure (5.6) Digital Input case Flowchart

➢ **Digital Output Case**

Fig(5.7) shows digital output case , this flowchart shows that when the user send command by clicking on checkbox, then checkbox is enabled (true) ,so the device which is connected to the bits of PORTD is turned on , else if the checkbox is false (disabled) the device will be turned off.

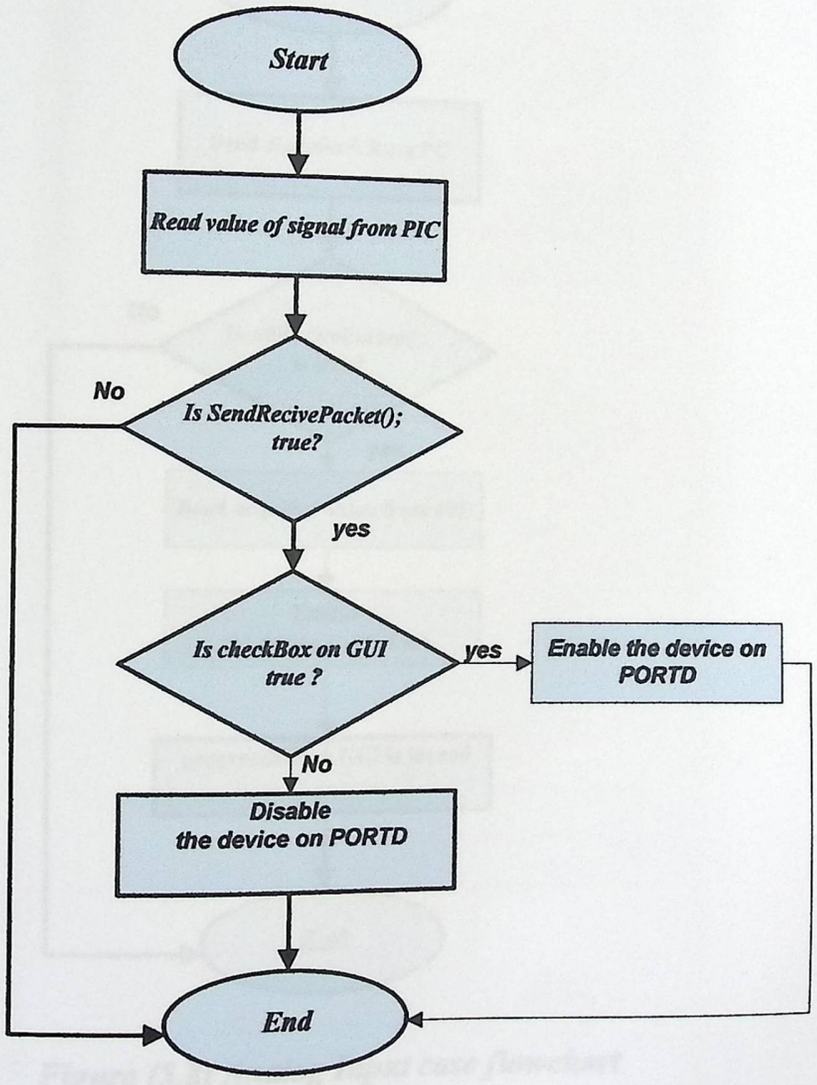


Figure (5.7)Digital Output case flowchart

➢ **Analog Input Case**

Figure (5.8) shows the analog input case, in this flowchart we can see that the value of progress bare on GUI on computer changes according to the variations of the value that happens to the device connected to PORTA bits.

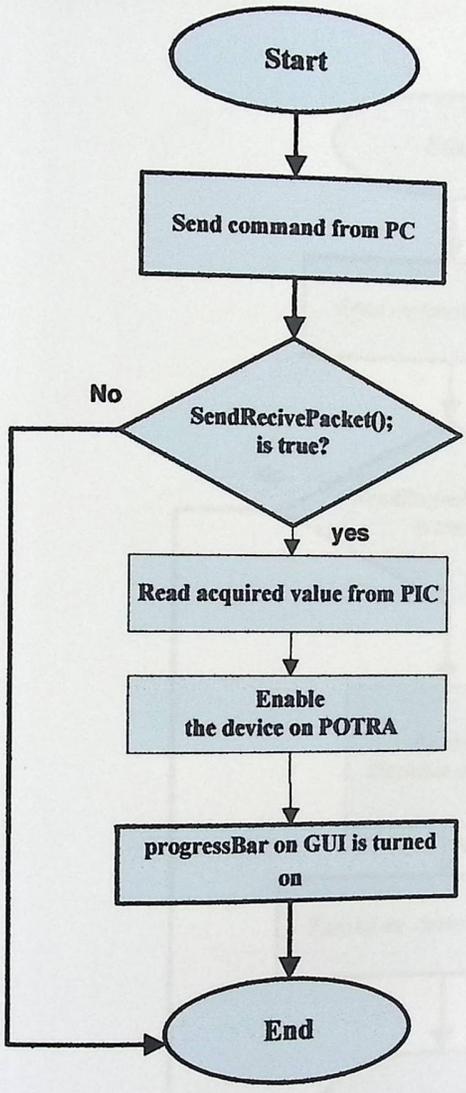


Figure (5.8) Analog Input case flowchart

➢ **Analog Output Case**

Figure(5.9) shows the flowchart of the analog output case , when the user changes the resolution of track Bar, then the resolution of the device that is connected to PORTC varies according to the changes of trace Bare .

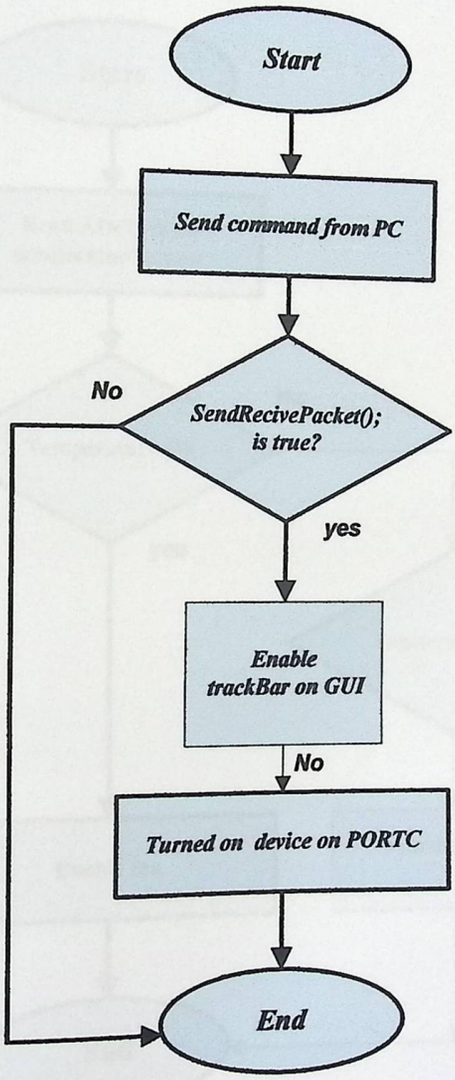


Figure (5.9) Analog Input case flowchart

- **Temperature Sensor Function**

The flowchart in figure (5.10) demonstrates the process of the LM35 temperature sensor function , after the sensor senses the temperature, so if it exceeds 28°C the fan circuit will be enabled, otherwise if it is below 20 °C the heater will be turned on ,between these two temperature values neither of them is enabled.

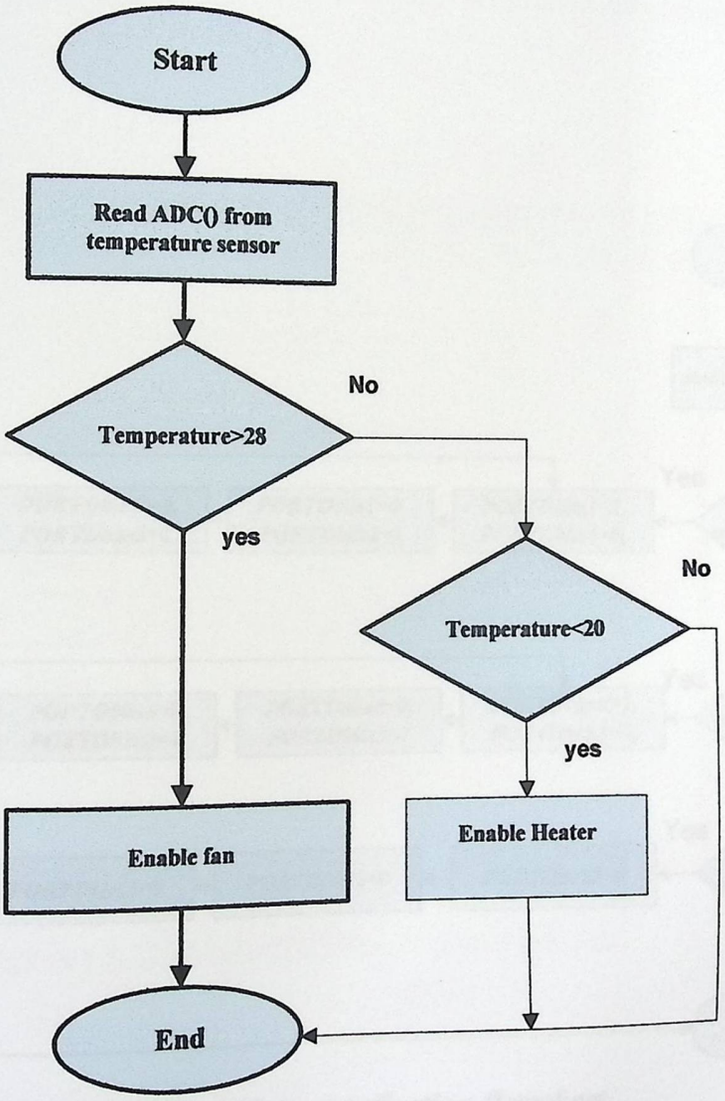
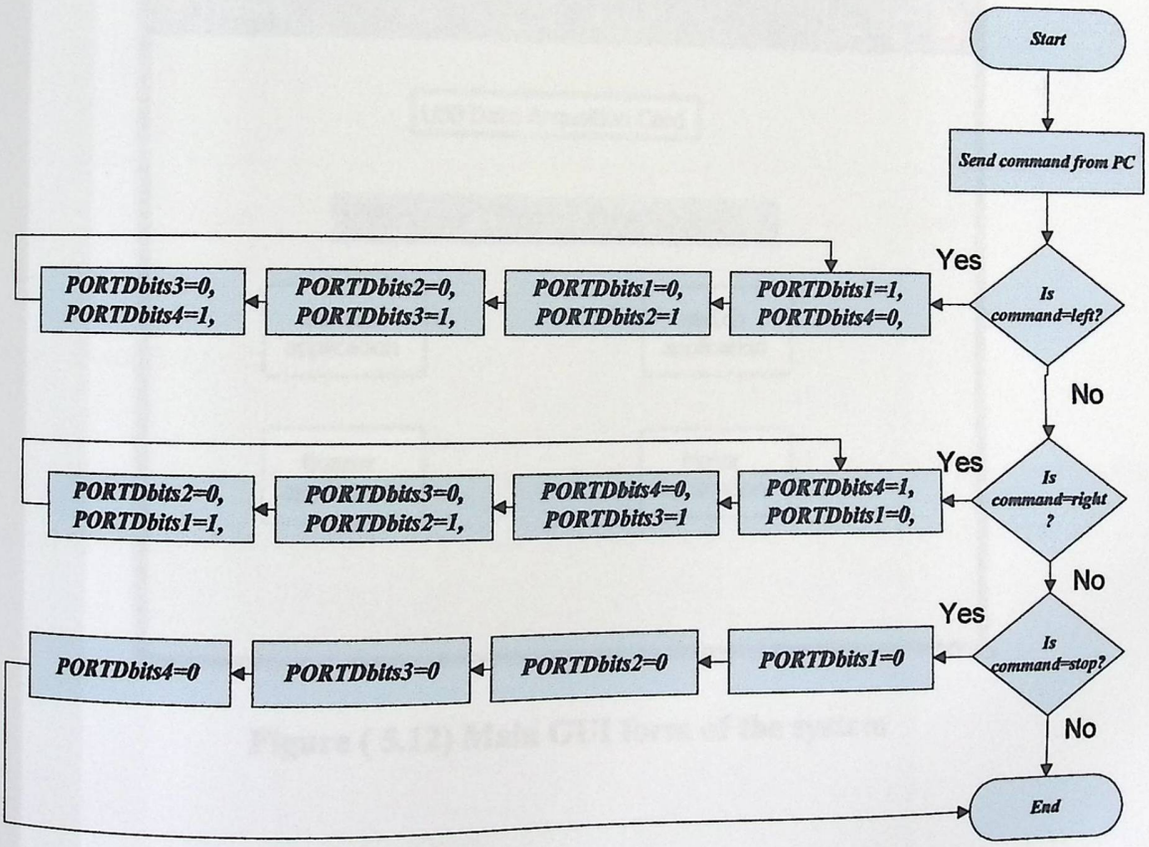


Figure (5.10) sensor application flowchart

- **Stepper motor function**

The flowchart in figure (5.11) displays the general process in which the stepper motor works .There are three cases for the movement of the motor ,it could be turned right , left, or stopped ,according to the commands sent as the flowchart describes.



Figure(5.11) Stepper Motor application flowchart

5.8 Graphical User Interface

The GUI for the system was done by visual C++ 2008, as this system is an educational one so the graphical interface should provide an easy and flexible platform to enable user to interact with the system, figure (5.12) shows the main form for the system, it is provided with the capabilities that enable user to choose the application he wants to activate, those applications were added to test the system functionality.

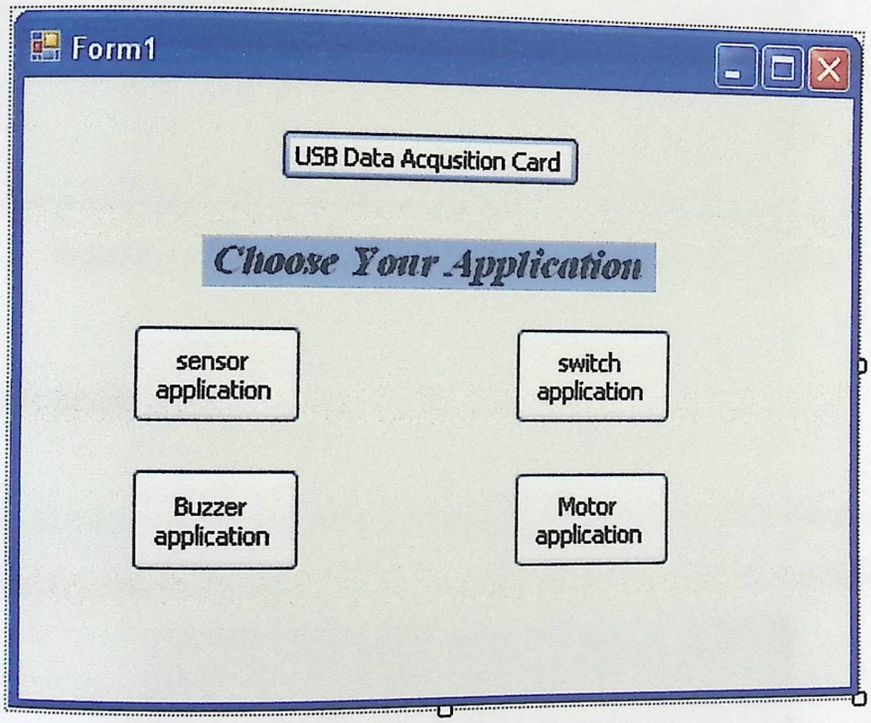


Figure (5.12) Main GUI form of the system

- **General USB data acquisition GUI**

Figure (5.13) displays the general data acquisition card ,this window is activated when the user presses the "USB data acquisition card " button in figure (5.12).it shows the analog input progress bars ,analog output trace bar,digital input lables, and digital output check

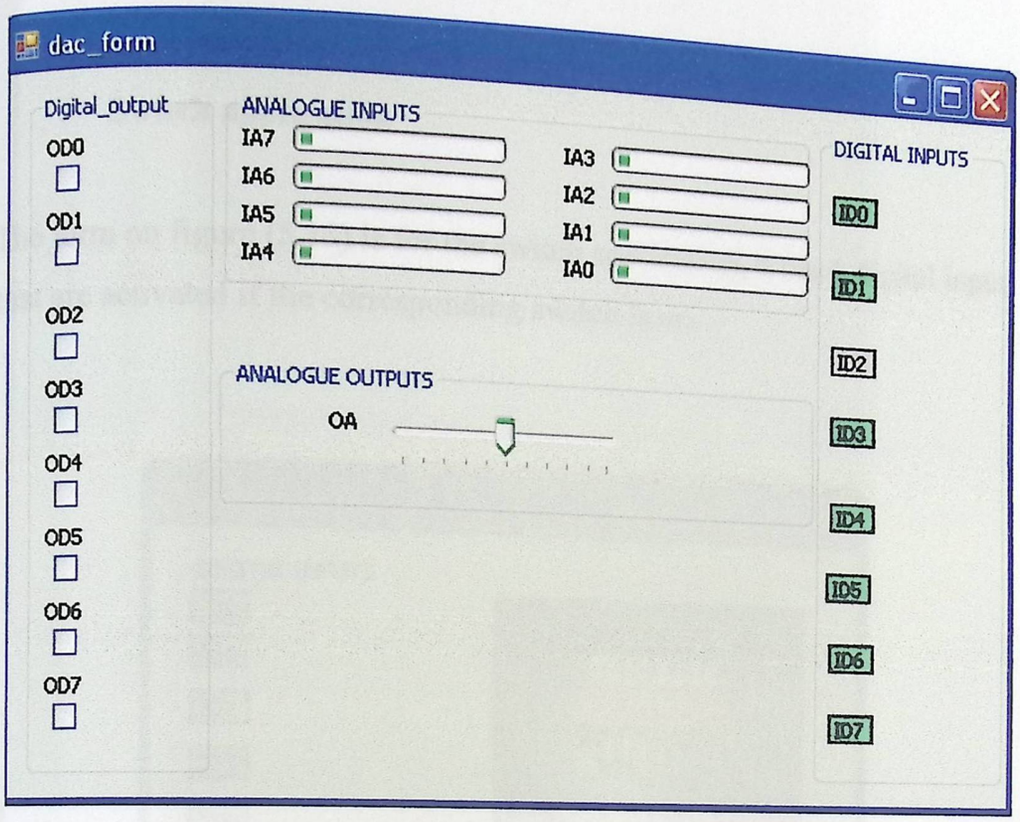


Figure (5.13) General USB data acquisition card form

- **Buzzer application**

Figure (5.14) displays the buzzer form that is activated when the user presses the buzzer application button in figure (5. 12), it has turning on and off capabilities.

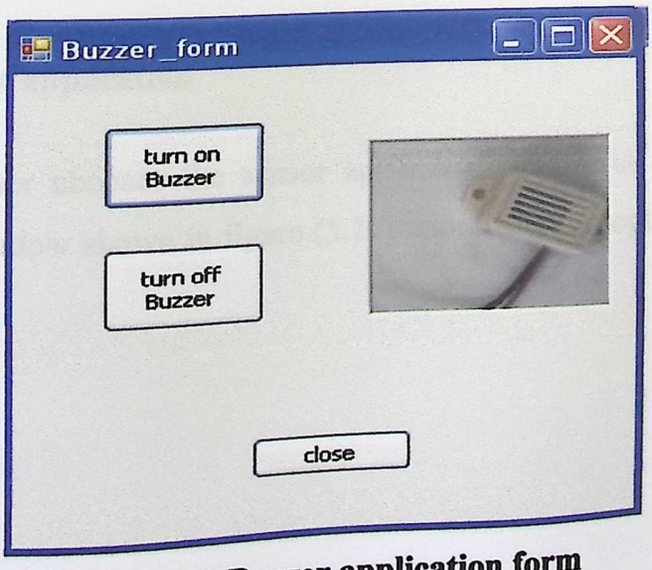


Figure (5.14) Buzzer application form

➢ **Switch application**

The form on figure (5.15) is for the switch application, it has 8 digital input labels that are activated if the corresponding switch is on.

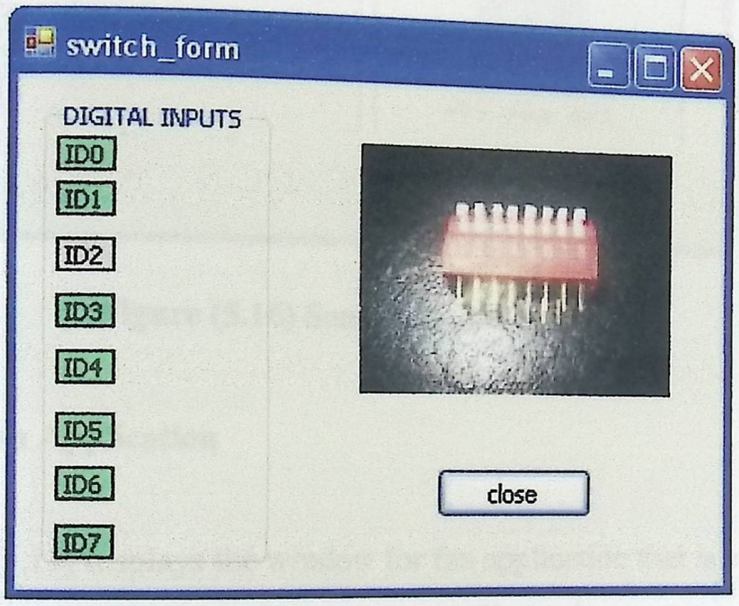


Figure (5.15) Switch application form

➢ **Sensor application**

When the user chooses the sensor application button of figure (5.12), the sensor application window shown in figure (5.16) appears, it has options for activating fan or heater circuit.

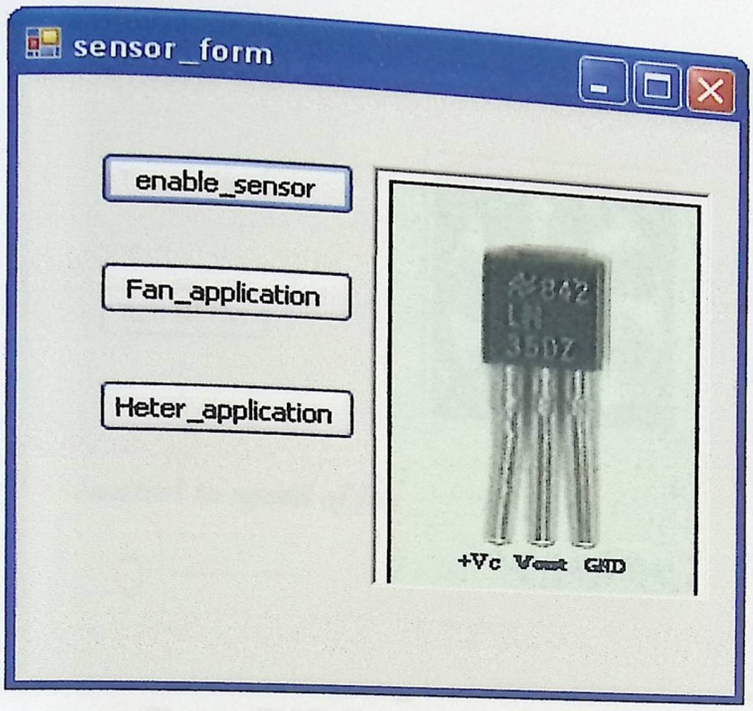


Figure (5.16) Sensor application form

- **Fan Application**

Figure (5.17) displays the window for fan application that is contained in sensor application form, the user can turn it on or off, also he can control the speed of fan by moving the trace bare.

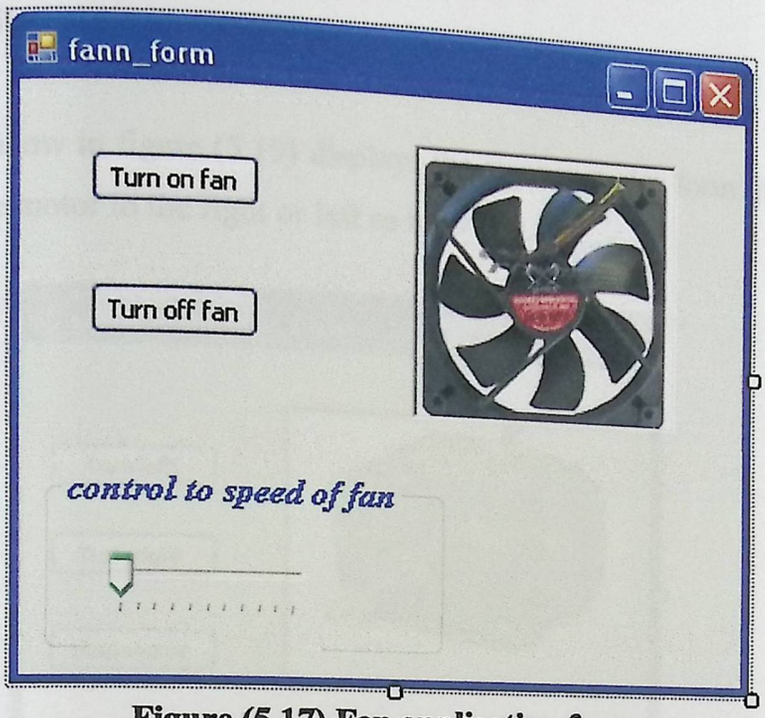


Figure (5.17) Fan application form

- **Heater application**

Figure (5.18) displays the heater application form, the user can turn it on or off, also this window is activated by the "heater application" button on sensor application.

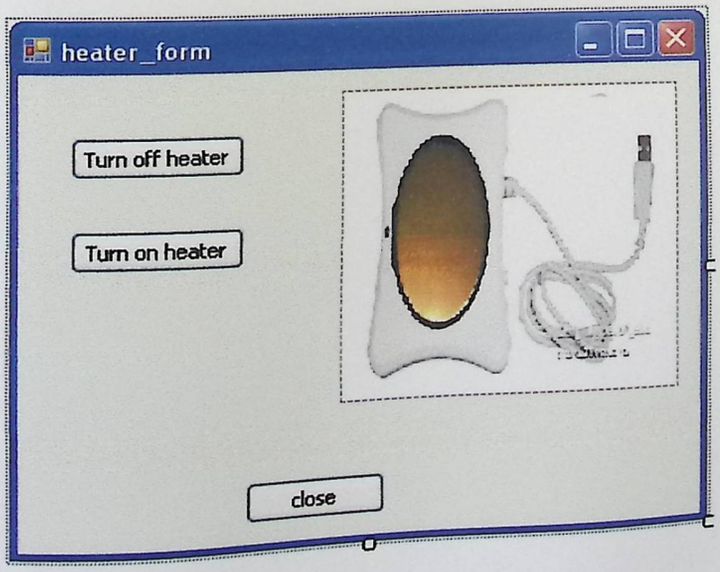


Figure (5.18) Heater application form

➢ **Stepper motor application**

The window in figure (5.19) displays the stepper motor form; it shows that the user can turn the motor to the right or left as he wants.

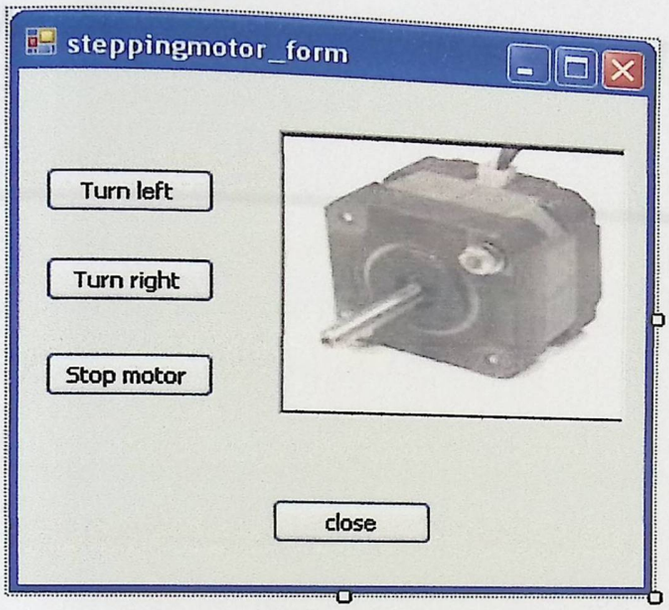


Figure (5.19) Stepper motor form

Chapter

6 IMPLEMENTATION AND TESTING

6.1 Preface

6.2 Implementation.

6.3 Project Components Testing.

Chapter Six

Implementation and Testing

6.1 Preface

To avoid project delays, it's critical to have clearly defined test strategies, and use these strategies throughout each phase of development. This chapter describes the implementation and testing processes for the system, the test process will be applied to hardware as well as software components of the system.

The implementation process will be done using different components and tools that will be discussed later in this chapter.

6.2 Implementation

We have implemented the overall project, including the USB board with the PIC18f4550 microcontroller, by first building the main USB circuit and ensuring that it works properly through programming the PIC via PIKIT2 programmer, with a simple test code, then downloading the major program code ,after that the circuits of the applications were built in separate phases, and ensured that the individual circuit worked well, the new circuit were added to the system , etc …, until all circuits were connected to the board, and worked well via the GUI on Visual C++.

This implementation process was done by using the following tools and components:

- Connectors and wires with different colors
- Four 10*2 cm bread boards.

- All ICs and components that are mentioned in the design chapter (chapter 4).
- Wire grabber and wire cutter.

6.3 Project components testing

This section describes the way by which each circuit and major components were tested including subsystem testing and the integrated system testing

6.3.1 Subsystem testing

Individual tests were applied to the following circuits:

1- PIC18f4550

2- Buzzer circuit

3- Heater circuit

4- Fan circuit

5- Stepper motor circuit

- **PIC18F4550 Testing**

The PIC microcontroller was tested by validating its ability for downloading programs, for this purpose we have used the PIKIT 2 programmer for more information about this programmer you can visit the microchip website, figure (6.1),(6.2) shows the PIC before and after testing respectively.

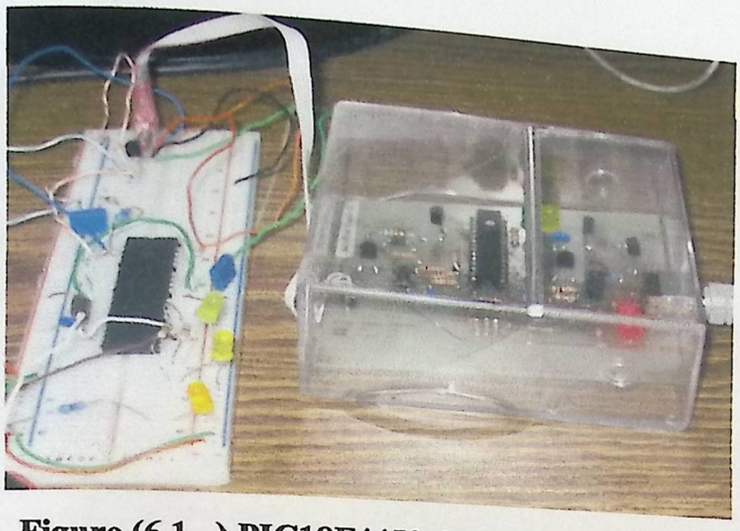


Figure (6.1) PIC18F4450 Circuit with PIKIT2 Programmer before Testing

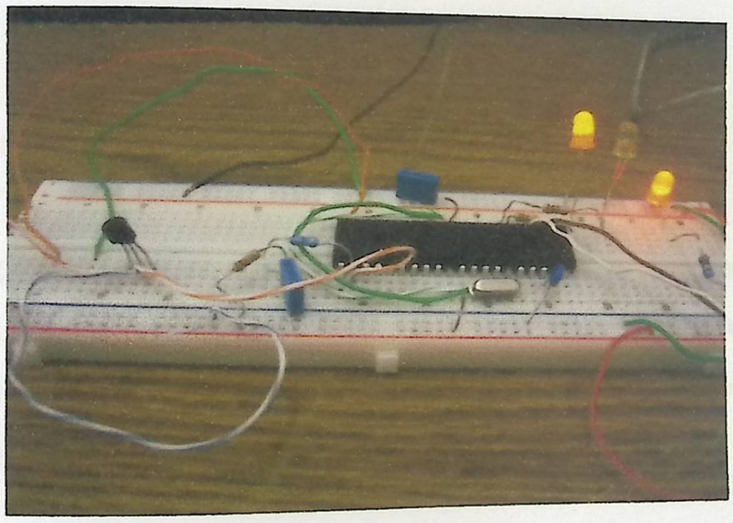


Figure (6.2) PIC18F4450 Circuit after Successful Test

- **Buzzer Testing**

An alarm circuit has been built using a simple buzzer and a transistor, one of the edges of the simple buzzer was connected to the high voltage (3V) from the function generator, while the other is connected to the ground, as a result the buzzer was activated and the buzzer rang, the circuit is shown in figure (6.3).

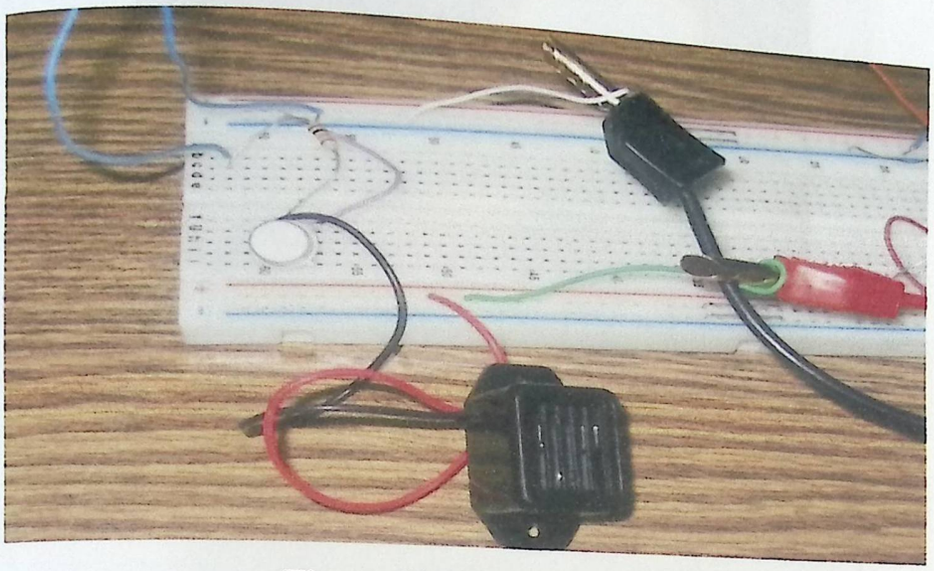


Figure (6.3) Buzzer testing

- **Heater testing**

The testing of the heater circuit shown in figure (6.4) was done using a lamp which contains a simple heater, for the reason of unavailability of real heater during testing; the lamp needs 220V, so an external power supply was used. Figure (6.5) displays the circuit after successful test.

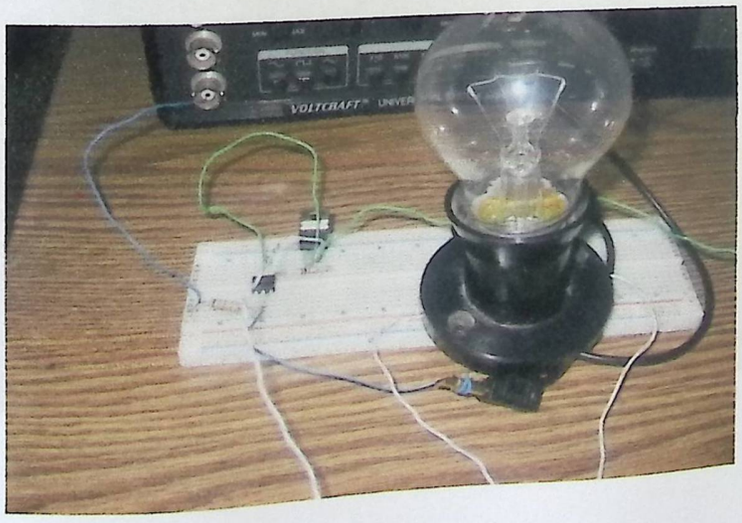


Figure (6.4) Heater circuit before test

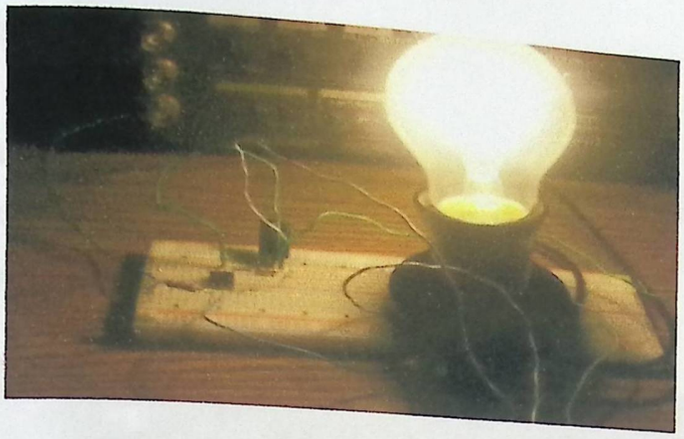


Figure (6.5) Heater circuit after test

- **Fan circuit**

The circuit in figure (6.6) displays the fan circuit, we used a CPU fan for testing this application, the function generator was fixed at 12 volt to run the fan, and it worked successfully.

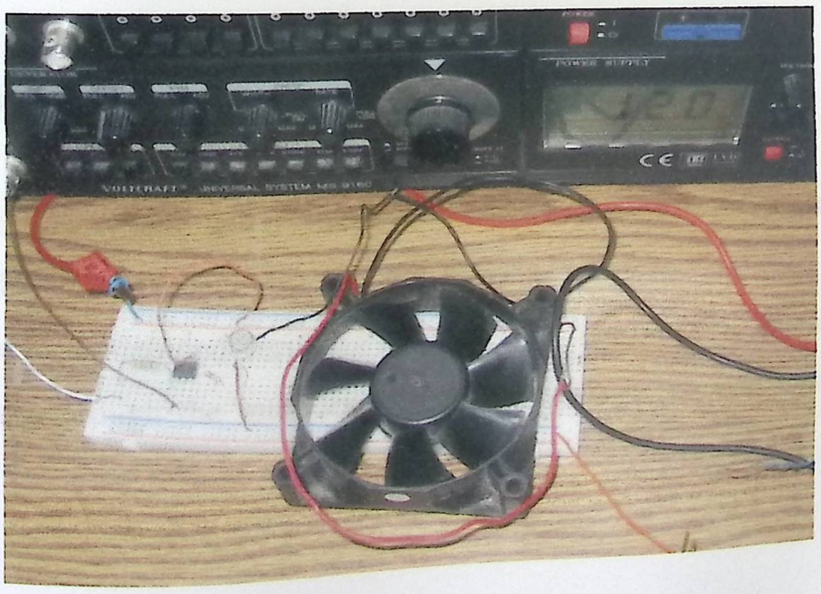


Figure (6.6) Fan Circuit Testing

- **Stepper motor testing**

The circuit in figure (6.7) displays a stepper motor, we used an adaptor to supply the circuit with 12 volt, the stepper motor needed four continuous pulses to run, we get the pulses from the function generator, and it worked fine

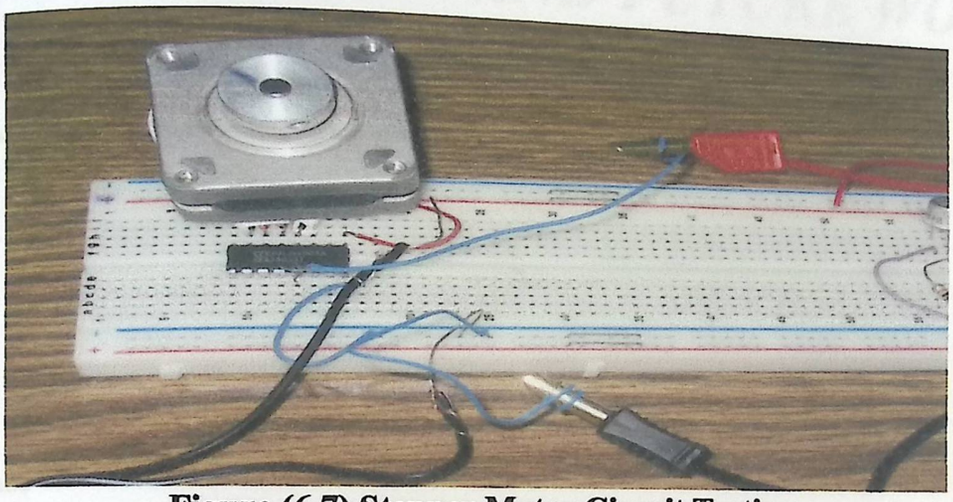


Figure (6.7) Stepper Motor Circuit Testing

6.3.2 Integrated System Testing

After the successful test of the hardware parts of the system, these circuits were connected to the USB board circuit, and tested again to ensure the proper work with the software, these tests included verifying the GUI interface on C++, so commands were sent by the user through the interface to activate the circuits, also this part was run successfully.

Chapter

7 CONCLUSION AND FUTURE WORK

7.1 Preface

7.2 Conclusion.

7.3 Problems.

7.4 Future Work.

Chapter Seven

Conclusion and Future Work

7.1 Preface

This chapter gives a holistic view on the project ,by representing the conclusions and results achieved , in addition to the problems that we faced during the development process, also it includes some recommendations for future, to whom who may need this project for later works.

7.2 Conclusion

Through evaluation process to the overall project, the following points have been noticed:

- The USB data acquisition card has been run successfully, achieving the general purpose of data acquisition.
- USB has many important benefits in the acquisition process, the easy installation, portability, less noise sensitivity and other advantages.
- All of the circuits of the applications have run successfully including the digital and analogue applications.
- Building a custom driver is not as easy process as expected; it needs a good understanding of the USB basics and functions.

- One of the limitations of USB is that the maximum current that USB could provide is about 500mA, which means that for certain applications an external power supply is needed.
- The USB Interface has been implemented successfully through developing the GUI on Visual C++ which is considered a rich programming language for supporting windows applications.
- Because of its low cost, high data-transfer rate, ease of use, and flexibility, USB has gained wide acceptance in the computer industry.
- Good knowledge has been accepted about the PIC 18F4550 microcontroller and its functions.

7.3 Problems

Some of hardware and software problems appeared during the development of the project here is an illustration of the main problems.

7.3.1 Hardware problems

- Latency of arrival of some components, which causes latency in completing some circuits.
- In few cases we found that some wires were connected wrongly, for instance swapping the high and low voltage to the circuit, which resulted in damaging few components.
- Some times we faced problems in dealing the PIKIT 2 programmer, according to hardware failer in some of its pins or connections, so we need to fix it or try another one, each time we need to download the program on PIC 18f4550.

- One of the limitations of the USB is that there is a maximum current of (500mA) which the USB port can support, which means that if you need more current for a certain application; you need to use external power supply.

- Some of warning messages appeared recording to errors in connections such as "a USB device has malfunctioned and exceeded the power limits of its hub ", but it was resolved successfully after reconnection of the wrong connections.

7.3.2 Software Problems

- The most important problem was, the lack of experience and knowledge in dealing with USB, since it was a new study field for us, so this required an additional effort and time for developing a new application which needs building of a custom driver that USB can recognize and use.

- Since there are several choices for choosing a USB library, that needed a lot of time from us to study the libraries and choose the best from the available ones.

- During the first work , a warning message indicated that the "USB device is not recognized " has appeared , after revising some of the configurations in main functions ,some of the modifications have been made ,so we succeeded in eliminating the problem and the USB device became recognized.

7.4 Future work

We recommend the following techniques and suggestions for future work and developing process:

- The inclusion of additional capabilities such as digital I/O lines, counter/timers, and encoders, can greatly increase the utility of a DAQ module.

- As a training system more applications can be added to the system so it will be useful for students in lab, also it will be useful for other projects as a support to make tests to make use of USB features.

- Enhancements could be made in order to use the system in special fields such as industry, security and automation.

- Improve the system by using wireless USB as a high quality technology.

References

Books:

[1] Axelson , J. (2005*). **Usb Complete : Everything You Need to Develop Custom Usb Peripherals***".3rd **edition**. Independent PubGroup.

[2] Dogan , I . (2008) . ***Advanced PIC Microcontroller Projects In C: From USB to RTOS with the PIC18F Series with CDROM***. Butter Worth Heinmann.United States

[3] Hyde , J . (2001) . ***USB Design by Example: A Practical Guide to Building I/O Devices (Paperback).***2nd edition. Intel University Press.

Websites:

http://www.alanmacek.com/usb/usb.c[4] Alan Macek , website:

[5] Admin ,website : http://www.embedds.com/pic18f4550-usb-prototyping-board

[6] Microchip, website : http://www.microchip.com

[7] Association Management , website :http://www.usb.org/home

[8] Audiophile Style, website: http://www.computeraudiophile.com/node/706

[9] Sreda, website: http://marjandeveloperelectro.blogspot.com/2008/09/ad-to-com-converter-reading-linear.html.

[10] Cornell University, website: http://instruct1.cit. cornell.edu/ courses/ee476/ Final Projects/ s2008/ djf35_ swl28/djf35_swl28/index.html

[11] Craig Peacok, website: http://www.beyondlogic.org/about.htm

[12] Microsoft Corporation, website: http://msdn.microsoft.com/en-us/library/ms894725.aspx.

[13] Quatech, website: http://www.quatech.com/support/comm-over-usb.php

[14] Atmel Corporation, website: http://www.atmel.com/dyn/Products/ Product_card.asp? part_id = 2014.

[15] Website: http://www.linux-usb.org/USB-guide/x75.html.

[16]National Semiconductor Corporation, website: http:// www.national.com.

[17] Digital projects. (2007).retrieved may 5, 2009, from USB acquisition website: http://www.it.lth.se/digp/sammanfattning/2007/lp3/grupp3/usbacquisition.htmldigitaldigital

[18] Picprojects. (2007).website:http://picprojects.org.uk/projects/picprojects.html

[19] Objective Development. All projects on V-bus, 2009 website: http://www.obdev.at/products/avrusb/prjall.html

[20] Resources for embedded micro-controller firmware development. June 2009 Web site:http://www.mjbauer.biz/mjb_resources.html.

[21] Microsoft software informer, 2008.website: http://microsoft-visual-c-2008-express-edition1.software.informer.com

[22] EDA board,2007.website:http://www.edaboard.com/ftopic294140.html

[23] Electronic parts, 2009. website:http://www.nowcomponents.com/item1/page.cfm/3387

[24] Free scale semiconductor, 2009.website: http://www.freescale.com/webapp/sps/site/overview.jsp?code=DRMTRSTPRMTR

[25] Yale University ,2006 website :http://pheatt.emporia.edu/projects/usb/Brown .grober .savvides.pdf

Appendix A

Datasheets

PIC18F2455/2550/4455/4550
Data Sheet

28/40/44-Pin High-Performance,
Enhanced Flash USB Microcontrollers
with nanoWatt Technology

MICROCHIP PIC18F2455/2550/4455/4550

28/40/44-Pin High-Performance, Enhanced Flash USB Microcontrollers with nanoWatt Technology

Universal Serial Bus Features:

- USB V2.0 Compliant
- Low Speed (1.5 Mb/s) and Full Speed (12 Mb/s)
- Supports Control, Interrupt, Isochronous and Bulk Transfers
- Supports up to 32 endpoints (16 bidirectional)
- 1-Kbyte dual access RAM for USB
- On-chip USB transceiver with on-chip voltage regulator
- Interface for off-chip USB transceiver
- Streaming Parallel Port (SPP) for USB streaming transfers (40/44-pin devices only)

Power-Managed Modes:

- Run: CPU on, peripherals on
- Idle: CPU off, peripherals on
- Sleep: CPU off, peripherals off
- Idle mode currents down to 5.8 µA typical
- Sleep mode currents down to 0.1 µA typical
- Timer1 oscillator: 1.1 µA typical, 32 kHz, 2V
- Watchdog Timer: 2.1 µA typical
- Two-Speed Oscillator Start-up

Flexible Oscillator Structure:

- Four Crystal modes including High Precision PLL for USB
- Two External Clock modes, up to 48 MHz
- Internal oscillator block:
  - 8 user-selectable frequencies, from 31 kHz to 8 MHz
  - User-tunable to compensate for frequency drift
- Secondary oscillator using Timer1 @ 32 kHz
- Dual oscillator options allow microcontroller and USB module to run at different clock speeds
- Fail-Safe Clock Monitor
  - Allows for safe shutdown if any clock stops

Peripheral Highlights:

- High-current sink/source 25 mA/25 mA
- Three external interrupts
- Four Timer modules (Timer0 to Timer3)
- Up to 2 Capture/Compare/PWM (CCP) modules:
  - Capture is 16-bit, max. resolution 6.25 ns (TCY/16)
  - Compare is 16-bit, max. resolution 100 ns (TCY)
  - PWM output: PWM resolution is 1 to 10-bit
- Enhanced Capture/Compare/PWM (ECCP) module:
  - Multiple output modes
  - Selectable polarity
  - Programmable dead time
  - Auto-Shutdown and Auto-Restart
- Enhanced USART module:
  - LIN bus support
- Master Synchronous Serial Port (MSSP) module supporting 3-wire SPI™ (all 4 modes) and I^2C™ Master and Slave modes
- 10-bit, up to 13-channels Analog-to-Digital Converter module (A/D) with programmable acquisition time
- Dual analog comparators with input multiplexing

Special Microcontroller Features:

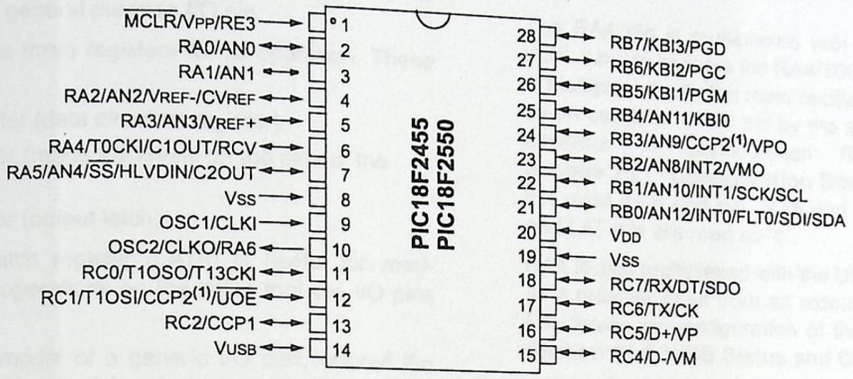
- C compiler optimized architecture with optional extended instruction set
- 100,000 erase/write cycle Enhanced Flash program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory typical
- Flash/Data EEPROM Retention: > 40 years
- Self-programmable under software control
- Priority levels for interrupts
- 8 x 8 Single-Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
  - Programmable period from 41 ms to 131s
- Programmable Code Protection
- Single-Supply 5V In-Circuit Serial Programming™ (ICSP™) via two pins
- In-Circuit Debug (ICD) via two pins
- Optional dedicated ICD/ICSP port (44-pin devices only)
- Wide operating voltage range (2.0V to 5.5V)

Device	Program Memory		Data Memory		I/O	10-bit A/D (ch)	CCP/ECCP (PWM)	SPP	MSSP		EAUSART	Comparators	Timers 8/16-bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					SPI™	Master I^2C™			
PIC18F2455	24K	12288	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F2550	32K	16384	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F4455	24K	12288	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3
PIC18F4550	32K	16384	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3

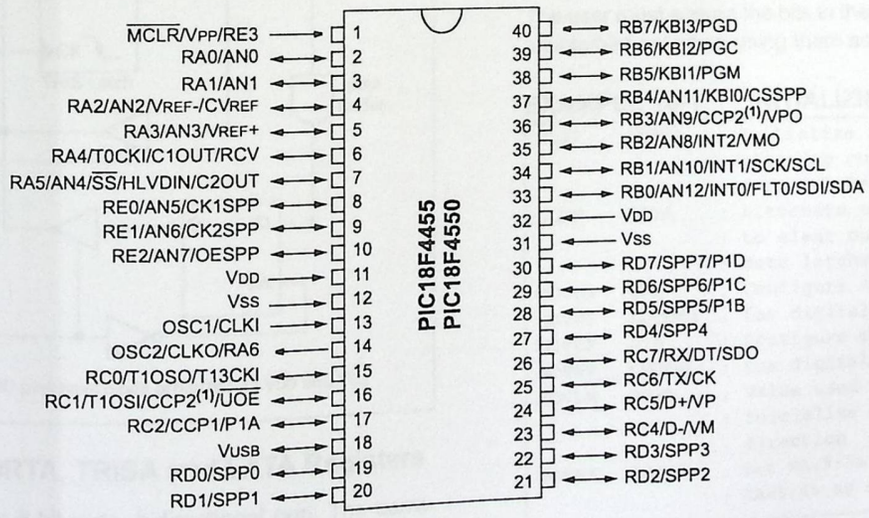
Preliminary

Pin Diagrams

28-Pin PDIP, SOIC



40-Pin PDIP



Note 1: RB3 is the alternate pin for CCP2 multiplexing.

10.0 I/O PORTS

Depending on the device selected and features enabled, there are up to five ports available. Some pins of the I/O ports are multiplexed with an alternate function from the peripheral features on the device. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.

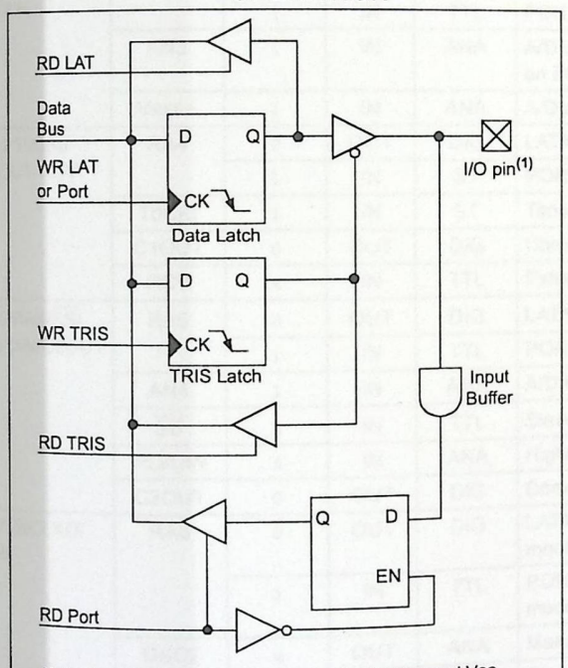
Each port has three registers for its operation. These registers are:

- TRIS register (data direction register)
- Port register (reads the levels on the pins of the device)
- LAT register (output latch)

The Data Latch register (LATA) is useful for read-modify-write operations on the value that the I/O pins are driving.

A simplified model of a generic I/O port, without the interfaces to other peripherals, is shown in Figure 10-1.

FIGURE 10-1: GENERIC I/O PORT OPERATION



Note 1: I/O pins have diode protection to VDD and VSS.

10.1 PORTA, TRISA and LATA Registers

PORTA is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISA. Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input (i.e., put the corresponding output driver in a high-impedance mode). Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin).

Reading the PORTA register reads the status of the pins; writing to it will write to the port latch.

The Data Latch register (LATA) is also memory mapped. Read-modify-write operations on the LATA register read and write the latched output value for PORTA.

The RA4 pin is multiplexed with the Timer0 module clock input to become the RA4/T0CKI pin. The RA6 pin is multiplexed with the main oscillator pin; it is enabled as an oscillator or I/O pin by the selection of the main oscillator in Configuration Register 1H (see **Section 25.1 "Configuration Bits"** for details). When not used as a port pin, RA6 and its associated TRIS and LAT bits are read as '0'.

RA4 is also multiplexed with the USB module; it serves as a receiver input from an external USB transceiver. For details on configuration of the USB module, see **Section 17.2 "USB Status and Control"**.

Several PORTA pins are multiplexed with analog inputs, the analog VREF+ and VREF- inputs and the comparator voltage reference output. The operation of pins RA3:RA0 and RA5 as A/D converter inputs is selected by clearing/setting the control bits in the ADCON1 register (A/D Control Register 1).

Note:	On a Power-on Reset, RA5 and RA3:RA0 are configured as analog inputs and read as '0'. RA4 is configured as a digital input.

All other PORTA pins have TTL input levels and full CMOS output drivers.

The TRISA register controls the direction of the RA pins, even when they are being used as analog inputs. The user must ensure the bits in the TRISA register are maintained set when using them as analog inputs.

EXAMPLE 10-1: INITIALIZING PORTA

```
CLRF    PORTA   ; Initialize PORTA by
                ; clearing output
                ; data latches
CLRF    LATA    ; Alternate method
                ; to clear output
                ; data latches
MOVLW   0Fh     ; Configure A/D
MOVWF   ADCON1  ; for digital inputs
MOVLW   07h     ; Configure comparators
MOVWF   CMCON   ; for digital input
MOVLW   0CFh    ; Value used to
                ; initialize data
                ; direction
MOVWF   TRISA   ; Set RA<3:0> as inputs
                ; RA<5:4> as outputs
```

TABLE 10-1: PORTA I/O SUMMARY

Pin	Function	TRIS Setting	I/O	I/O Type	Description
RA0/AN0	RA0	0	OUT	DIG	LATA<0> data output; not affected by analog input.
		1	IN	TTL	PORTA<0> data input; disabled when analog input enabled.
	AN0	1	IN	ANA	A/D input channel 0 and Comparator C1- input. Default configuration on POR; does not affect digital output.
RA1/AN1	RA1	0	OUT	DIG	LATA<1> data output; not affected by analog input.
		1	IN	TTL	PORTA<1> data input; reads '0' on POR.
	AN1	1	IN	ANA	A/D input channel 1 and Comparator C2- input. Default configuration on POR; does not affect digital output.
RA2/AN2/ VREF-/CVREF	RA2	0	OUT	DIG	LATA<2> data output; not affected by analog input. Disabled when CVREF output enabled.
		1	IN	TTL	PORTA<2> data input. Disabled when analog functions enabled; disabled when CVREF output enabled.
	AN2	1	IN	ANA	A/D input channel 2 and Comparator C2+ input. Default configuration on POR; not affected by analog output.
	VREF-	1	IN	ANA	A/D and comparator voltage reference low input.
	CVREF	x	OUT	ANA	Comparator voltage reference output. Enabling this feature disables digital I/O.
RA3/AN3/ VREF+	RA3	0	OUT	DIG	LATA<3> data output; not affected by analog input.
		1	IN	TTL	PORTA<3> data input; disabled when analog input enabled.
	AN3	1	IN	ANA	A/D input channel 3 and Comparator C1+ input. Default configuration on POR.
	VREF+	1	IN	ANA	A/D and comparator voltage reference high input.
RA4/T0CKI/ C1OUT/RCV	RA4	0	OUT	DIG	LATA<4> data output; not affected by analog input.
		1	IN	ST	PORTA<4> data input; disabled when analog input enabled.
	T0CKI	1	IN	ST	Timer0 clock input.
	C1OUT	0	OUT	DIG	Comparator 1 output; takes priority over port data.
	RCV	x	IN	TTL	External USB transceiver RCV input.
RA5/AN4/$\overline{SS}$/ HLVDIN/C2OUT	RA5	0	OUT	DIG	LATA<5> data output; not affected by analog input.
		1	IN	TTL	PORTA<5> data input; disabled when analog input enabled.
	AN4	1	IN	ANA	A/D input channel 4. Default configuration on POR.
	$\overline{SS}$	1	IN	TTL	Slave select input for SSP (MSSP module).
	HLVDIN	1	IN	ANA	High/Low-Voltage Detect external trip point input.
	C2OUT	0	OUT	DIG	Comparator 2 output; takes priority over port data.
OSC2/CLKO/ RA6	RA6	0	OUT	DIG	LATA<6> data output. Available only in ECIO, ECPIO and INTIO modes; otherwise, reads as '0'.
		1	IN	TTL	PORTA<6> data input. Available only in ECIO, ECPIO and INTIO modes; otherwise, reads as '0'.
	OSC2	x	OUT	ANA	Main oscillator feedback output connection (all XT and HS modes).
	CLKO	x	OUT	DIG	System cycle clock output (FOSC/4); available in EC, ECPLL and INTCKO modes.

Legend: OUT = Output, IN = Input, ANA = Analog Signal, DIG = Digital Output, ST = Schmitt Buffer Input, TTL = TTL Buffer Input, x = Don't care (TRIS bit does not affect port direction or is overridden for this option).

TABLE 10-2: SUMMARY OF REGISTERS ASSOCIATED WITH PORTA

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
PORTA	—	RA6(1)	RA5	RA4	RA3	RA2	RA1	RA0	54
LATA	—	LATA6(1)	LATA5	LATA4	LATA3	LATA2	LATA1	LATA0	54
TRISA	—	TRISA6(1)	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	54
ADCON1	—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0	52
CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	53
CVRCON	CVREN	CVROE	CVRR	CVRSS	CVR3	CVR2	CVR1	CVR0	53
UCON	—	PPBRST	SE0	PKTDIS	USBEN	RESUME	SUSPND	—	55

Legend: — = unimplemented, read as '0'. Shaded cells are not used by PORTA.

Note 1: RA6 and its associated latch and data direction bits are enabled as I/O pins based on oscillator configuration; otherwise, they are read as '0'.

10.2 PORTB, TRISB and LATB Registers

PORTB is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISB. Setting a TRISB bit (= 1) will make the corresponding PORTB pin an input (i.e., put the corresponding output driver in a high-impedance mode). Clearing a TRISB bit (= 0) will make the corresponding PORTB pin an output (i.e., put the contents of the output latch on the selected pin).

The Data Latch register (LATB) is also memory mapped. Read-modify-write operations on the LATB register read and write the latched output value for PORTB.

Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is performed by clearing bit $\overline{\text{RBPU}}$ (INTCON2<7>). The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.

Note: On a Power-on Reset, RB4:RB0 are configured as analog inputs by default and read as '0'; RB7:RB5 are configured as digital inputs.

By programming the configuration bit, PBADEN (CONFIG3H<1>), RB4:RB0 will alternatively be configured as digital inputs on POR.

Four of the PORTB pins (RB7:RB4) have an interrupt-on-change feature. Only pins configured as inputs can cause this interrupt to occur; any RB7:RB4 pin configured as an output is excluded from the interrupt-on-change comparison. The pins are compared with the old value latched on the last read of PORTB. The "mismatch" outputs of RB7:RB4 are ORed together to generate the RB Port Change Interrupt with Flag bit, RBIF (INTCON<0>).

The interrupt-on-change can be used to wake the device from Sleep. The user, in the Interrupt Service Routine, can clear the interrupt in the following manner:

a) Any read or write of PORTB (except with the `MOVFF (ANY), PORTB` instruction). This will end the mismatch condition.

b) Clear flag bit, RBIF.

A mismatch condition will continue to set flag bit, RBIF. Reading PORTB will end the mismatch condition and allow flag bit, RBIF, to be cleared.

The interrupt-on-change feature is recommended for wake-up on key depression operation and operations where PORTB is only used for the interrupt-on-change feature. Polling of PORTB is not recommended while using the interrupt-on-change feature.

Pins, RB2 and RB3, are multiplexed with the USB peripheral and serve as the differential signal outputs for an external USB transceiver (TRIS configuration). Refer to **Section 17.2.2.2 "External Transceiver"** for additional information on configuring the USB module for operation with an external transceiver.

RB4 is multiplexed with CSSPP, the chip select function for the Streaming Parallel Port (SPP) – TRIS setting. Details of its operation are discussed in **Section 18.0 "Streaming Parallel Port"**.

EXAMPLE 10-2: INITIALIZING PORTB

```
CLRF    PORTB   ; Initialize PORTB by
                ; clearing output
                ; data latches
CLRF    LATB    ; Alternate method
                ; to clear output
                ; data latches
MOVLW   0Eh     ; Set RB<4:0> as
MOVWF   ADCON1  ; digital I/O pins
                ; (required if config bit
                ; PBADEN is set)
MOVLW   0CFh    ; Value used to
                ; initialize data
                ; direction
MOVWF   TRISB   ; Set RB<3:0> as inputs
                ; RB<5:4> as outputs
                ; RB<7:6> as inputs
```

13.0 TIMER2 MODULE

The Timer2 module timer incorporates the following features:

- 8-bit timer and period registers (TMR2 and PR2, respectively)
- Readable and writable (both registers)
- Software programmable prescaler (1:1, 1:4 and 1:16)
- Software programmable postscaler (1:1 through 1:16)
- Interrupt-on-TMR2 to PR2 match
- Optional use as the shift clock for the MSSP module

The module is controlled through the T2CON register (Register 13-1) which enables or disables the timer and configures the prescaler and postscaler. Timer2 can be shut off by clearing control bit, TMR2ON (T2CON<2>), to minimize power consumption.

A simplified block diagram of the module is shown in Figure 13-1.

13.1 Timer2 Operation

In normal operation, TMR2 is incremented from 00h on each clock (FOSC/4). A 2-bit counter/prescaler on the clock input gives direct input, divide-by-4 and divide-by-16 prescale options. These are selected by the prescaler control bits, T2CKPS1:T2CKPS0 (T2CON<1:0>). The value of TMR2 is compared to that of the period register, PR2, on each clock cycle. When the two values match, the comparator generates a match signal as the timer output. This signal also resets the value of TMR2 to 00h on the next cycle and drives the output counter/postscaler (see **Section 13.2 "Timer2 Interrupt"**).

The TMR2 and PR2 registers are both directly readable and writable. The TMR2 register is cleared on any device Reset, while the PR2 register initializes at FFh. Both the prescaler and postscaler counters are cleared on the following events:

- a write to the TMR2 register
- a write to the T2CON register
- any device Reset (Power-on Reset, $\overline{\text{MCLR}}$ Reset, Watchdog Timer Reset or Brown-out Reset)

TMR2 is not cleared when T2CON is written.

REGISTER 13-1: T2CON: TIMER2 CONTROL REGISTER

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **T2OUTPS3:T2OUTPS0**: Timer2 Output Postscale Select bits
0000 = 1:1 Postscale
0001 = 1:2 Postscale
•
•
•
1111 = 1:16 Postscale

bit 2 **TMR2ON**: Timer2 On bit
1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

13.2 Timer2 Interrupt

Timer2 also can generate an optional device interrupt. The Timer2 output signal (TMR2 to PR2 match) provides the input for the 4-bit output counter/postscaler. This counter generates the TMR2 match interrupt flag which is latched in TMR2IF (PIR1<1>). The interrupt is enabled by setting the TMR2 Match Interrupt Enable bit, TMR2IE (PIE1<1>).

A range of 16 postscale options (from 1:1 through 1:16 inclusive) can be selected with the postscaler control bits, T2OUTPS3:T2OUTPS0 (T2CON<6:3>).

13.3 TMR2 Output

The unscaled output of TMR2 is available primarily to the CCP modules, where it is used as a time base for operations in PWM mode.

Timer2 can be optionally used as the shift clock source for the MSSP module operating in SPI mode. Additional information is provided in **Section 19.0 "Master Synchronous Serial Port (MSSP) Module"**.

FIGURE 13-1: TIMER2 BLOCK DIAGRAM

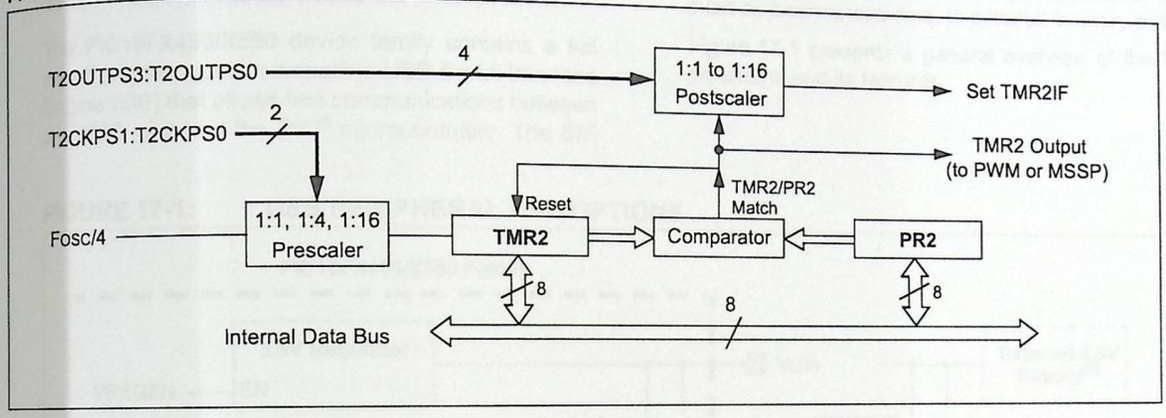


TABLE 13-1: REGISTERS ASSOCIATED WITH TIMER2 AS A TIMER/COUNTER

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	51
PIR1	SPPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	54
PIE1	SPPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	54
IPR1	SPPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	54
TMR2	Timer2 Register								52
T2CON	—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0	52
PR2	Timer2 Period Register								52

Legend: — = unimplemented, read as '0'. Shaded cells are not used by the Timer2 module.

Note 1: These bits are unimplemented on 28-pin devices; always maintain these bits clear.

17.0 UNIVERSAL SERIAL BUS (USB)

This section describes the details of the USB peripheral. Because of the very specific nature of the module, knowledge of USB is expected. Some high-level USB information is provided in **Section 17.10 "Overview of USB"** only for application design reference. Designers are encouraged to refer to the official specification published by the USB Implementers Forum (USB-IF) for the latest information. USB Specification Revision 2.0 is the most current specification at the time of publication of this document.

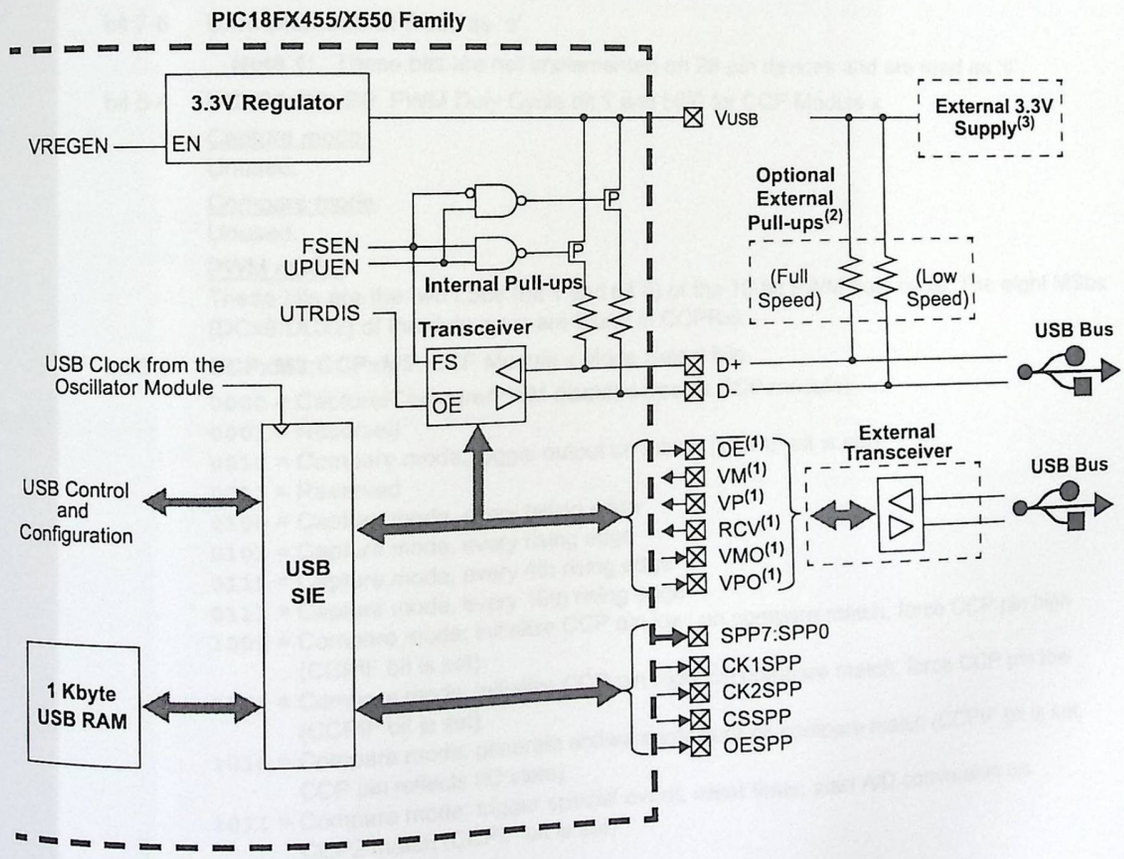
17.1 Overview of the USB Peripheral

The PIC18FX455/X550 device family contains a full speed and low-speed compatible USB Serial Interface Engine (SIE) that allows fast communications between any USB host and the PIC® microcontroller. The SIE can be interfaced directly to the USB, utilizing the internal transceiver, or it can be connected through an external transceiver. An internal 3.3V regulator is also available to power the internal transceiver in 5V applications.

Some special hardware features have been included to improve performance. Dual port memory in the device's data memory space (USB RAM) has been supplied to share direct memory access between the microcontroller core and the SIE. Buffer descriptors are also provided, allowing users to freely program endpoint memory usage within the USB RAM space. A Streaming Parallel Port has been provided to support the uninterrupted transfer of large volumes of data, such as isochronous data, to external memory buffers.

Figure 17-1 presents a general overview of the USB peripheral and its features.

FIGURE 17-1: USB PERIPHERAL AND OPTIONS



Note 1: This signal is only available if the internal transceiver is disabled (UTRDIS = 1).

2: The internal pull-up resistors should be disabled (UPUEN = 0) if external pull-up resistors are used.

3: Do not enable the internal regulator when using an external 3.3V supply.

15.0 CAPTURE/COMPARE/PWM (CCP) MODULES

PIC18F2455/2550/4455/4550 devices all have two CCP (Capture/Compare/PWM) modules. Each module contains a 16-bit register, which can operate as a 16-bit Capture register, a 16-bit Compare register or a PWM Master/Slave Duty Cycle register.

In 28-pin devices, the two standard CCP modules (CCP1 and CCP2) operate as described in this chapter. In 40/44-pin devices, CCP1 is implemented as an Enhanced CCP module, with standard Capture and Compare modes and Enhanced PWM modes. The ECCP implementation is discussed in **Section 16.0 "Enhanced Capture/Compare/PWM (ECCP) Module"**.

The Capture and Compare operations described in this chapter apply to all standard and Enhanced CCP modules.

> **Note:** Throughout this section and **Section 16.0 "Enhanced Capture/Compare/PWM (ECCP) Module"**, references to the register and bit names for CCP modules are referred to generically by the use of 'x' or 'y' in place of the specific module number. Thus, "CCPxCON" might refer to the control register for CCP1, CCP2 or ECCP1. "CCPxCON" is used throughout these sections to refer to the module control register regardless of whether the CCP module is a standard or Enhanced implementation.

REGISTER 15-1: CCPxCON: STANDARD CCP CONTROL REGISTER

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—(1)	—(1)	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

Note 1: These bits are not implemented on 28-pin devices and are read as '0'.

bit 5-4 **DCxB1:DCxB0**: PWM Duty Cycle bit 1 and bit 0 for CCP Module x

Capture mode:
Unused.

Compare mode:
Unused.

PWM mode:
These bits are the two LSbs (bit 1 and bit 0) of the 10-bit PWM duty cycle. The eight MSbs (DCx9:DCx2) of the duty cycle are found in CCPRxL.

bit 3-0 **CCPxM3:CCPxM0**: CCP Module x Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCP module)
0001 = Reserved
0010 = Compare mode, toggle output on match (CCPIF bit is set)
0011 = Reserved
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode: initialize CCP pin low; on compare match, force CCP pin high (CCPIF bit is set)
1001 = Compare mode: initialize CCP pin high; on compare match, force CCP pin low (CCPIF bit is set)
1010 = Compare mode: generate software interrupt on compare match (CCPIF bit is set, CCP pin reflects I/O state)
1011 = Compare mode: trigger special event, reset timer, start A/D conversion on CCP2 match (CCPIF bit is set)
11xx = PWM mode

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

National Semiconductor

November 2000

LM35
Precision Centigrade Temperature Sensors

General Description

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in ° Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full −55 to +150°C temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. As it draws only 60 µA from its supply, it has very low self-heating, less than 0.1°C in still air. The LM35 is rated to operate over a −55° to +150°C temperature range, while the LM35C is rated for a −40° to +110°C range (−10° with improved accuracy). The LM35 series is available packaged in hermetic TO-46 transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package. The LM35D is also available in an 8-lead surface mount small outline package and a plastic TO-220 package.

Features

- Calibrated directly in ° Celsius (Centigrade)
- Linear + 10.0 mV/°C scale factor
- 0.5°C accuracy guaranteeable (at +25°C)
- Rated for full −55° to +150°C range
- Suitable for remote applications
- Low cost due to wafer-level trimming
- Operates from 4 to 30 volts
- Less than 60 µA current drain
- Low self-heating, 0.08°C in still air
- Nonlinearity only ±¼°C typical
- Low impedance output, 0.1 Ω for 1 mA load

Typical Applications

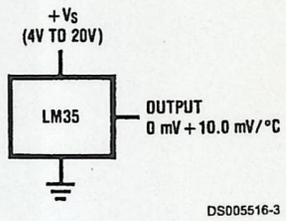
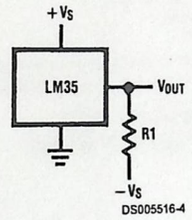


FIGURE 1. Basic Centigrade Temperature Sensor (+2°C to +150°C)

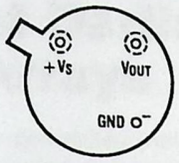


Choose $R_1 = -V_S/50\ \mu A$
V_{OUT} = +1,500 mV at +150°C
= +250 mV at +25°C
= −550 mV at −55°C

FIGURE 2. Full-Range Centigrade Temperature Sensor

Connection Diagrams

TO-46
Metal Can Package*

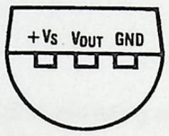


BOTTOM VIEW
DS005516-1

*Case is connected to negative pin (GND)

Order Number LM35H, LM35AH, LM35CH, LM35CAH or LM35DH
See NS Package Number H03H

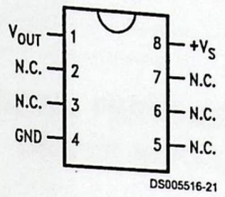
TO-92
Plastic Package



BOTTOM VIEW
DS005516-2

Order Number LM35CZ, LM35CAZ or LM35DZ
See NS Package Number Z03A

SO-8
Small Outline Molded Package

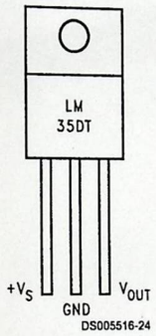


DS005516-21

N.C. = No Connection

Top View
Order Number LM35DM
See NS Package Number M08A

TO-220
Plastic Package*



DS005516-24

*Tab is connected to the negative pin (GND).
Note: The LM35DT pinout is different than the discontinued LM35DP.

Order Number LM35DT
See NS Package Number TA03F

MOTOROLA

Order this document by ULN2803/D

Octal High Voltage, High Current Darlington Transistor Arrays

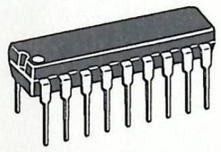
ULN2803
ULN2804

OCTAL PERIPHERAL DRIVER ARRAYS

SEMICONDUCTOR TECHNICAL DATA

The eight NPN Darlington connected transistors in this family of arrays are ideally suited for interfacing between low logic level digital circuitry (such as TTL, CMOS or PMOS/NMOS) and the higher current/voltage requirements of lamps, relays, printer hammers or other similar loads for a broad range of computer, industrial, and consumer applications. All devices feature open–collector outputs and free wheeling clamp diodes for transient suppression.

The ULN2803 is designed to be compatible with standard TTL families while the ULN2804 is optimized for 6 to 15 volt high level CMOS or PMOS.



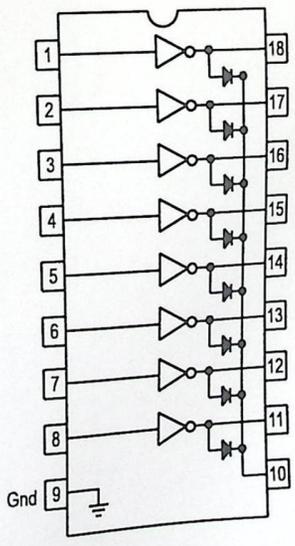
A SUFFIX
PLASTIC PACKAGE
CASE 707

MAXIMUM RATINGS (T_A = 25°C and rating apply to any one device in the package, unless otherwise noted.)

Rating	Symbol	Value	Unit
Output Voltage	V_O	50	V
Input Voltage (Except ULN2801)	V_I	30	V
Collector Current – Continuous	I_C	500	mA
Base Current – Continuous	I_B	25	mA
Operating Ambient Temperature Range	T_A	0 to +70	°C
Storage Temperature Range	T_{stg}	–55 to +150	°C
Junction Temperature	T_J	125	°C

$R_{\theta JA}$ = 55°C/W
Do not exceed maximum current limit per driver.

PIN CONNECTIONS



ORDERING INFORMATION

Device	Characteristics		
	Input Compatibility	$V_{CE(Max)}/I_{C(Max)}$	Operating Temperature Range
ULN2803A ULN2804A	TTL, 5.0 V CMOS 6 to 15 V CMOS, PMOS	50 V/500 mA	T_A = 0 to + 70°C

Rev 1

ELECTRICAL CHARACTERISTICS (T_A = 25°C, unless otherwise noted)

Characteristic		Symbol	Min	Typ	Max	Unit
Output Leakage Current (Figure 1)		I_{CEX}				µA
(V_O = 50 V, T_A = +70°C)	All Types		–	–	100	
(V_O = 50 V, T_A = +25°C)	All Types		–	–	50	
(V_O = 50 V, T_A = +70°C, V_I = 6.0 V)	ULN2802		–	–	500	
(V_O = 50 V, T_A = +70°C, V_I = 1.0 V)	ULN2804		–	–	500	
Collector–Emitter Saturation Voltage (Figure 2)		$V_{CE(sat)}$				V
(I_C = 350 mA, I_B = 500 µA)	All Types		–	1.1	1.6	
(I_C = 200 mA, I_B = 350 µA)	All Types		–	0.95	1.3	
(I_C = 100 mA, I_B = 250 µA)	All Types		–	0.85	1.1	
Input Current – On Condition (Figure 4)		$I_{I(on)}$				mA
(V_I = 17 V)	ULN2802		–	0.82	1.25	
(V_I = 3.85 V)	ULN2803		–	0.93	1.35	
(V_I = 5.0 V)	ULN2804		–	0.35	0.5	
(V_I = 12 V)	ULN2804		–	1.0	1.45	
Input Voltage – On Condition (Figure 5)		$V_{I(on)}$				V
(V_{CE} = 2.0 V, I_C = 300 mA)	ULN2802		–	–	13	
(V_{CE} = 2.0 V, I_C = 200 mA)	ULN2803		–	–	2.4	
(V_{CE} = 2.0 V, I_C = 250 mA)	ULN2803		–	–	2.7	
(V_{CE} = 2.0 V, I_C = 300 mA)	ULN2803		–	–	3.0	
(V_{CE} = 2.0 V, I_C = 125 mA)	ULN2804		–	–	5.0	
(V_{CE} = 2.0 V, I_C = 200 mA)	ULN2804		–	–	6.0	
(V_{CE} = 2.0 V, I_C = 275 mA)	ULN2804		–	–	7.0	
(V_{CE} = 2.0 V, I_C = 350 mA)	ULN2804		–	–	8.0	
Input Current – Off Condition (Figure 3) (I_C = 500 µA, T_A = +70°C)	All Types	$I_{I(off)}$	50	100	–	µA
DC Current Gain (Figure 2) (V_{CE} = 2.0 V, I_C = 350 mA)	ULN2801	h_{FE}	1000	–	–	–
Input Capacitance		C_I	–	15	25	pF
Turn–On Delay Time (50% E_I to 50% E_O)		t_{on}	–	0.25	1.0	µs
Turn–Off Delay Time (50% E_I to 50% E_O)		t_{off}	–	0.25	1.0	µs
Clamp Diode Leakage Current (Figure 6) (V_R = 50 V)	T_A = +25°C	I_R	–	–	50	µA
	T_A = +70°C				100	
Clamp Diode Forward Voltage (Figure 7) (I_F = 350 mA)		V_F	–	1.5	2.0	V

TEST FIGURES

(See Figure Numbers in Electrical Characteristics Table)

Figure 1.

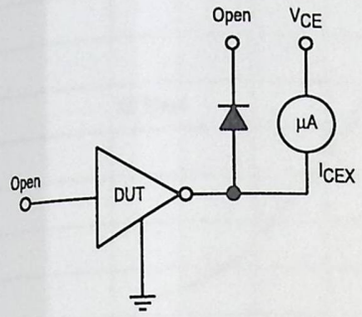


Figure 2.

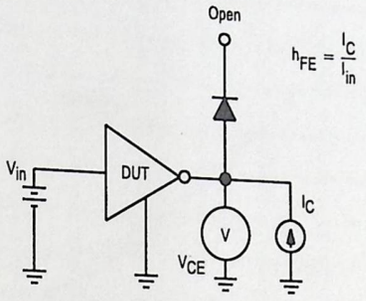


Figure 3.

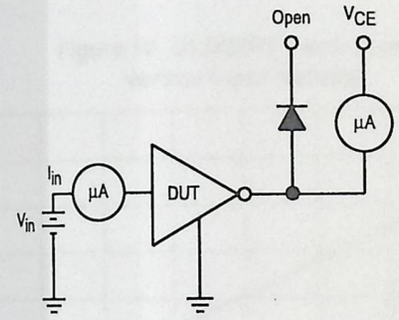


Figure 4.

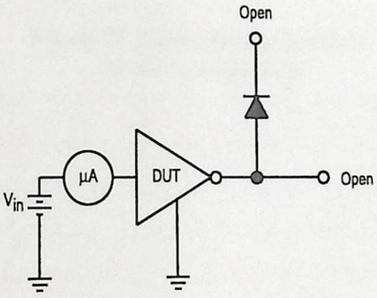


Figure 5.

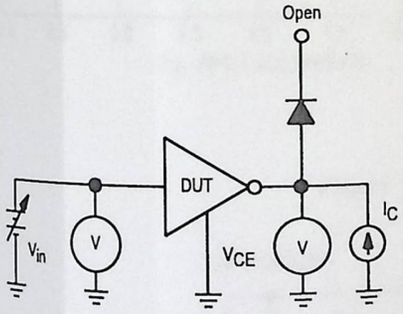


Figure 6.

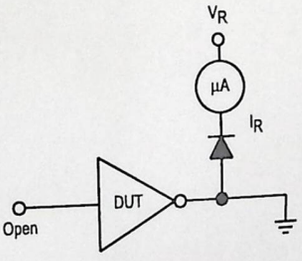
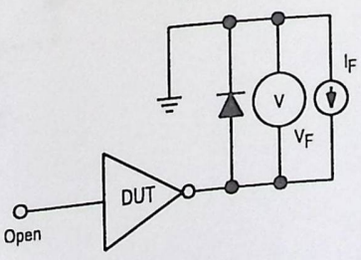


Figure 7.



TYPICAL CHARACTERISTIC CURVES – T_A = 25°C, unless otherwise noted

Output Characteristics

Figure 8. Output Current versus Saturation Voltage

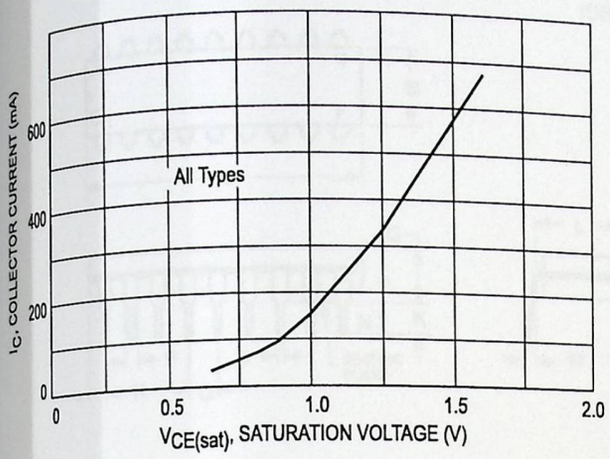
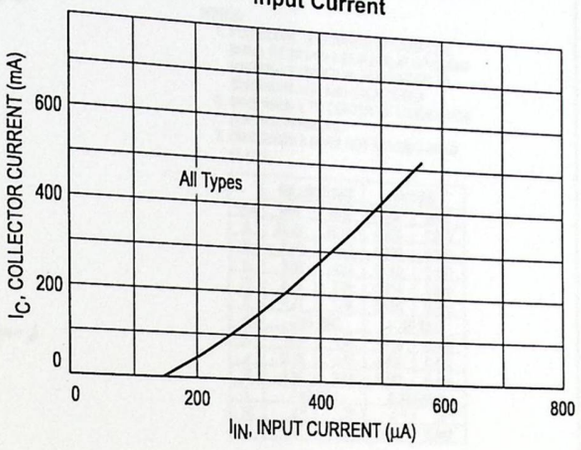


Figure 9. Output Current versus Input Current



Input Characteristics

Figure 10. ULN2803 Input Current versus Input Voltage

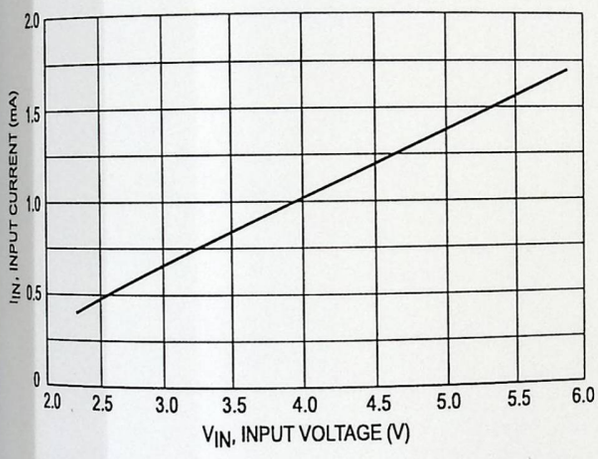


Figure 11. ULN2804 Input Current versus Input Voltage

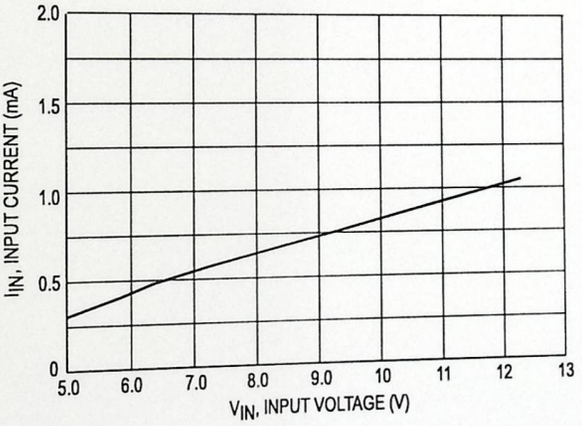
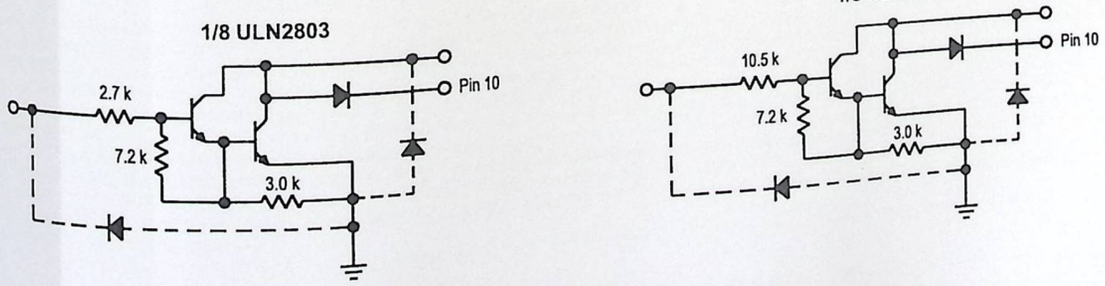
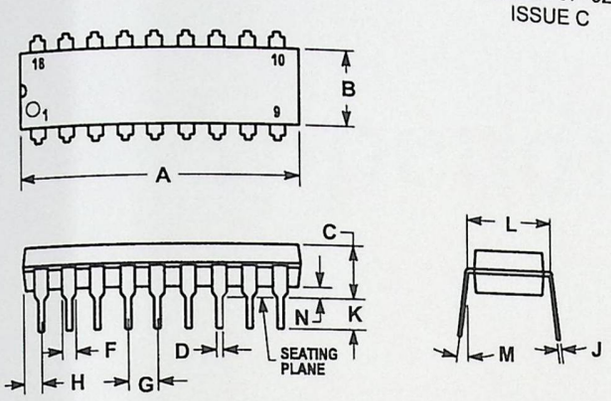


Figure 12. Representative Schematic Diagrams



OUTLINE DIMENSIONS

A SUFFIX
PLASTIC PACKAGE
CASE 707–02
ISSUE C



NOTES:
1. POSITIONAL TOLERANCE OF LEADS (D), SHALL BE WITHIN 0.25 (0.010) AT MAXIMUM MATERIAL CONDITION, IN RELATION TO SEATING PLANE AND EACH OTHER.
2. DIMENSION L TO CENTER OF LEADS WHEN FORMED PARALLEL.
3. DIMENSION B DOES NOT INCLUDE MOLD FLASH.

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	22.22	23.24	0.875	0.915
B	6.10	6.60	0.240	0.260
C	3.56	4.57	0.140	0.180
D	0.36	0.56	0.014	0.022
F	1.27	1.78	0.050	0.070
G	2.54 BSC		0.100 BSC	
H	1.02	1.52	0.040	0.060
J	0.20	0.30	0.008	0.012
K	2.92	3.43	0.115	0.135
L	7.62 BSC		0.300 BSC	
M	0°	15°	0°	15°
N	0.51	1.02	0.020	0.040

Appendix B

Source code

How to reach us:
USA/EUROPE/Locations Not Listed: Motorola Literature Distribution;
P.O. Box 20912; Phoenix, Arizona 85036. 1–800–441–2447 or 602–303–5454

MFAX: RMFAX0@email.sps.mot.com – TOUCHTONE 602–244–6609
INTERNET: http://Design–NET.com

JAPAN: Nippon Motorola Ltd.; Tatsumi–SPD–JLDC, 6F Seibu–Butsuryu–Center, 3–14–2 Tatsumi Koto–Ku, Tokyo 135, Japan. 03–81–3521–8315

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852–26629298

Appendix B

Source code

```
#ifndef PICDEM_FS_DEMO_H
#define PICDEM_FS_DEMO_H

/** I N C L U D E S **************************************************************/
#include "system\typedefs.h"

/** D E F I N I T I O N S ********************************************************/
/* PICDEM FS USB Demo Version */
#define MINOR_VERSION   0x00    //Demo Version 1.00
#define MAJOR_VERSION   0x01

/** S T R U C T U R E S **********************************************************/
typedef union DATA_PACKET
{
   byte _byte[USBGEN_EP_SIZE];  //For byte access
   word _word[USBGEN_EP_SIZE/2];//For word access(USBGEN_EP_SIZE msut be even)
   struct
   {
      enum    //PROTOCOLO
      {
                  READ_VERSION          = 0x00,
                  OUTPUT_DIGITAL_BIT    = 0x11,
                  INPUT_DIGITAL_BIT     = 0x13,
                  OUTPUT_ANALOGE        = 0x14,
                  INPUT_ANALOGE         = 0x15,
                  RESET                 = 0xFF,
      }CMD;
      byte len;
   };

   struct      //output_digital_bit
   {
      unsigned :8;
      byte output_digital_bit_number_output;
      byte state_output_digital;
   };

   struct      //input_digital_bit
   {
      unsigned :8;
      byte input_digital_bit_number_input;
      byte state_input_digital;
};
   struct      //output_analog
   {
      unsigned :8;
      byte output_analoge_byte_number_output;
            byte value_output_analoge_H;
            byte value_output_analoge_L;
   };
   struct      //input_analog
   {
      unsigned :8;
      byte input_analoge_number_input;
      byte value_input_analoge_H;
            byte value_input_analoge_L;
  };
} DATA_PACKET;
/** P U B L I C  P R O T O T Y P E S *********************************************/
void UserInit(void);
void ProcessIO(void);

#endif
```

```
/** I N C L U D E S **************************************************************/
#include <p18cxxx.h>
#include <portb.h>
#include <pwm.h>
#include <timers.h>
#include <adc.h>
#include "system\typedefs.h"

#include "system\usb\usb.h"

#include "io_cfg.h"
#include "user\user.h"

/** V A R I A B L E S ************************************************************/
#pragma udata
byte counter;

DATA_PACKET dataPacket;

/** P R I V A T E  P R O T O T Y P E S ***************************************/

void BlinkUSBStatus(void);
void ServiceRequests(void);

/** D E C L A R A T I O N S *****************************************************/
#pragma code
void UserInit(void)
{

init_prots(); //initalization for prots as input and outpt
dac(); // initalization fot timer ,PWM,openADC

}//end UserInit


void ProcessIO(void)
{
      #if defined(USE_BLINK_USB_STATUS)
            BlinkUSBStatus();
      #endif

   // User Application USB tasks
   if((usb_device_state < CONFIGURED_STATE)||(UCONbits.SUSPND==1)) return;

   ServiceRequests();

}//end ProcessIO

void ServiceRequests(void)
{

   Word R;
   if(USBGenRead((byte*)&dataPacket,sizeof(dataPacket)))
   {
       counter = 0;
       switch(dataPacket.CMD)
       {
           case READ_VERSION:

               dataPacket._byte[2] = MINOR_VERSION;
               dataPacket._byte[3] = MAJOR_VERSION;
               counter=0x04;
           break;

          case OUTPUT_DIGITAL_BIT:
                              if(dataPacket.state_output_digital!=0)
```

```
                                        user
                                        dataPacket.state_output_digital=1;
switch(dataPacket.output_digital_bit_number_output)
                                        {
                                        case 0:
LATDbits.LATD0=dataPacket.state_output_digital;
                                        break;
                                        case 1:
LATDbits.LATD1=dataPacket.state_output_digital;
                                        break;
                                        case 2:
LATDbits.LATD2=dataPacket.state_output_digital;
                                        break;
                                        case 3:
LATDbits.LATD3=dataPacket.state_output_digital;
                                        break;
                                        case 4:
LATDbits.LATD4=dataPacket.state_output_digital;
                                        break;
                                        case 5:
LATDbits.LATD5=dataPacket.state_output_digital;
                                        break;
                                        case 6:
LATDbits.LATD6=dataPacket.state_output_digital;
                                        break;
                                        case 7:
LATDbits.LATD7=dataPacket.state_output_digital;
                                        break;
                                    }
                counter = 0x01;
                break;
                        break;


        case INPUT_DIGITAL_BIT:
                                    switch(dataPacket.input_digital_bit_number_input)
                                    {
                                        case 0:
dataPacket.state_input_digital=PORTBbits.RB0;
                                        break;
                                        case 1:
dataPacket.state_input_digital=PORTBbits.RB1;
                                        break;
                                        case 2:
dataPacket.state_input_digital=PORTBbits.RB2;
                                        break;
                                        case 3:
dataPacket.state_input_digital=PORTBbits.RB3;
                                        break;
                                        case 4:
dataPacket.state_input_digital=PORTBbits.RB4;
                                        break;
                                        case 5:
```

```
                                        user
dataPacket.state_input_digital=PORTBbits.RB5;
                                        break;
                                        case 6:
dataPacket.state_input_digital=PORTBbits.RB6;
                                        break;
                                        case 7:
dataPacket.state_input_digital=PORTBbits.RB7;
                                        break;
                                }
            counter=0x03;
            break;

        case OUTPUT_ANALOGE:
            w=(dataPacket.value_output_analoge_H);
                                R=R<<8;
                                R=R|dataPacket.value_output_analoge_L;
if(dataPacket.output_analoge_byte_number_output==0)
                                {
                                        SetDCPWM1(R);
                                }
            else

                                {
                                break;
                                }
                                counter=0x01;
            break;

        case INPUT_ANALOGE:
                                switch(dataPacket.input_analoge_number_input)
                                {
                                        case 0:
                                        SetChanADC(ADC_CH0);
                                        break;
                                        case 1:
                                        SetChanADC(ADC_CH1);
                                        break;
                                        case 2:
                                        SetChanADC(ADC_CH2);
                                        break;
                                        case 3:
                                        SetChanADC(ADC_CH3);
                                        break;
                                        case 4:
                                        SetChanADC(ADC_CH4);
                                        break;
                                        case 5:
                                        SetChanADC(ADC_CH5);
                                        break;
                                        case 6:
                                        SetChanADC(ADC_CH6);
                                        break;
                                        case 7:
                                        SetChanADC(ADC_CH7);
                                        break;

                                }
                                ConvertADC(); // Start conversion
                                while(BusyADC()); // Wait for completion
                                w=ReadADC(); // Read result
                                dataPacket.value_input_analog_H=(R>>8)&0x0003;
                                dataPacket.value_input_analog_L=R&0x00FF;
                                counter=0x04;

            break;

        case RESET:
```

```
            Reset();
            break;

        default:
            break;
    }//end switch()
    if(counter != 0)
    {
        if(!mUSBGenTxIsBusy())
            USBGenWrite((byte*)&dataPacket,counter);
    }//end if
}//end if

}//end ServiceRequests

/********************************************************************************

void BlinkUSBStatus(void)
{
  static word led_count=0;

  if(led_count == 0)led_count = 10000U;
  led_count--;

  #define mLED_Both_Off()          {mLED_1_Off();mLED_2_Off();}
  #define mLED_Both_On()           {mLED_1_On();mLED_2_On();}
  #define mLED_Only_1_On()         {mLED_1_On();mLED_2_Off();}
  #define mLED_Only_2_On()         {mLED_1_Off();mLED_2_On();}

  if(UCONbits.SUSPND == 1)
  {
      if(led_count==0)
      {
          mLED_1_Toggle();
          mLED_2 = mLED_1;         // Both blink at the same time
      }//end if
  }
  else
  {
      if(usb_device_state == DETACHED_STATE)
      {
          mLED_Both_Off();
      }
      else if(usb_device_state == ATTACHED_STATE)
      {
          mLED_Both_On();
      }
      else if(usb_device_state == POWERED_STATE)
      {
          mLED_Only_1_On();
      }
      else if(usb_device_state == DEFAULT_STATE)
      {
          mLED_Only_2_On();
      }
      else if(usb_device_state == ADDRESS_STATE)
      {
          if(led_count == 0)
          {
              mLED_1_Toggle();
              mLED_2_Off();
          }//end if
      }
      else if(usb_device_state == CONFIGURED_STATE)
      {
          if(led_count==0)
          {
```

```
            mLED_1_Toggle();            user
            mLED_2 = !mLED_1;
        }//end if                       // Alternate blink
    }//end if(...)
}//end if(UCONbits.SUSPND...)

}//end BlinkUSBStatus
/** EOF user.c ***************************************************************/
```