P-14

e - CONTROL OF ELECTRIC DRIVES

A PROJECT REPORT

Submitted by

LAKSHMANAN.A	(71201105021)
MUTHU.C.T.	(71201105026)
PANNEERSELVAM.P	(71201105028)
SELVAN.T	(71201105051)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

ELECTRICAL & ELECTRONICS ENGINEERING

Under the guidance of

Mrs.K.PREMALATHA

KUMARAGURU COLLEGE OF TECHNOLOGY,
COIMBATORE - 641006

ANNA UNIVERSITY :: CHENNAI - 600 025

APRIL – 2005

ANNA UNIVERSITY : CHENNAI-600 025

BONAFIDE CERTIFICATE

Certified that this project report titled " **e–CONTROL OF ELECTRIC DRIVES** " is the bonafide work of

LAKSHMANAN.A	-	**Register No. 71201105021**
MUTHU.C.T	-	**Register No. 71201105026**
PANNEERSELVAM.P	-	**Register No. 71201105028**
SELVAN.T	-	**Register No. 71201105051**

who carried out the project work under my supervision.

[illegible]

Signature of Head of the Department

[illegible]

Signature of the guide

Prof.K.Regupathy Subramanian B.E.,M.Sc.
HOD EEE / DEAN (R&D),
Kumaraguru college
of technology.

Mrs.K.Premalatha M.E,
Senior Lecturer EEE,
Kumaraguru college
of technology.

CERTIFICATE OF EVALUATION

College : **KUMARAGURU COLLEGE OF TECHNOLOGY**

Branch : **Electrical & Electronics Engineering**

Semester : **Eighth Semester**

Sl.No.	Name of the Students	Title of the Project	Name of the Supervisor with Designation
01	Lakshmanan.A	**“ e - Control of Electric Drives ”**	Mrs.K.Premalatha M.E. Senior Lecturer
02	Muthu.C.T		
03	Panneerselvam.P		
04	Selvan.T		

The report of the project work submitted by the above students in partial fulfillment for the award of Bachelor of Engineering degree in Electrical & Electronics Engineering of Anna University were evaluated and confirmed to be report of the work done by the above students and then evaluated.

[illegible]

(INTERNAL EXAMINER)

[illegible] 15/4/05

(EXTERNAL EXAMINER)

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

We thank the management of **Kumaraguru College of Technology,** Coimbatore, for giving us this opportunity to undertake this project.

We extend our heart bound thanks to **Dr .K.K. Padmanabhan,** Principal Kumaraguru College of technology for permitting us to undertake this project.

We also thank **Dr.Regupathy Subramanian,** Head of the Department of electrical and electronics engineering and Dean (R&D) Kumaraguru College of Technology, Coimbatore and **Dr T.M. Kameswaran** Dean (EEE) Kumaraguru College of Technology, Coimbatore for permitting us to undertake this project.

We mention our gratitude and sincere thanks to our faculty guide, **Mrs. K.Premalatha** and **Mr. P.Thirumoorthi** for their timely help throughout the duration of project. We would like to thank them for duly evaluating our progress and encouraging us.

We would also like to thank the laboratory staff, electrical and electronics engineering for providing us all the resources for the completion of the project.

Finally we would like to thank our parents for their support and help.

ABSTRACT

In the recent years, technology has grown to a major extent so that people and the scientific community expects the process of monitoring and controlling their device parameters from remote places. The project "e-Control of Electric Drives" uses the wizard of the twentieth century, the Internet to monitor and control the various parameters of electrical appliances. The project employs TCP/IP Protocols and CGI (Common Gateway Interface) for establishing connection with the server. The project is developed in such a manner that it could be used in the LAN and WAN networks also in addition to the INTERNET. The scheme of controlling multiple appliances is also possible by providing the necessary hardware accessories using the Multi user and Multi tasking capabilities of LINUX.

Dedicated hardware circuits compromising of the Micro controllers, Thyristors and the various other omnipotent hardware circuits and devices which are required for the control of electrical appliances is designed and are connected with the Server through Serial Interface whose complete details are furnished to a greater level in the forth coming chapters.

TABLE OF CONTENTS

APPENDICES

FIGURE NAME	PAGE NO.
2.1 System Block Diagram	5
4.1 Client – Server Architecture	11
4.2 Client – Server Communication	13
4.3 Three Way Handshake	14
4.4 TCP Connection Termination	15
4.5 Socket Address Structure	18
4.6 Flow chart of Client – Server Connection	20
5.1 Asynchronous Data Transmission	34
5.2 Serial Communication	37
6.1 Generation of Firing Pulses	55
6.2 Flow chart of controller 1	56
6.3 Flow chart of controller 2	57

TABLE NAME	**PAGE NO.**
5.1 Termios Structure	42

INTRODUCTION

INTRODUCTION

"Conversation enriches understanding, but real solitude research is the school of genius"

1.1 GENERAL

A thing is right when it tends to preserve integrity, stability and beauty of biotic community. Creative activity is particularly the apt way to express oneself but the ability to create and the productions which results from such ability are generally regarded as possessing value by our society.

The project "e-Control of Electric Drives" is one such a creative project, which mainly aims for controlling and monitoring of various parameters of an electrical appliance through the Internet. The first visible signs which led to the building of the project was to control the appliances from remote places. From the turn of the century, Internet has dominated the entire world. The motive drove to understand the basics of information transfer through internet.

The connection between the Client and Server is established using TCP/IP Protocol, which concerns on Peer-to-Peer networking. Once the working of project is checked in this architecture it is updated to the CGI.The CGI (Common Gateway Interface) provides the path for configuring a dedicated web server. The importance for using the CGI is to enable it in the APACHE Server. The information consisting of the client request is transmitted to the server through the Internet. Queer little twists and quirks go into the making of the project. To suppress them all is important.

The request of the Client is obtained in the Server. Then it is transmitted to microcontroller through Asynchronous Serial Communication through POSIX Interface. The micro controller performs the job of processing the client request provided to them by the server. The first micro controller receives the request in addition to the process of monitoring the temperature and voltage. The second micro controller performs the real job of producing the exact time delay for the control of appliances. A separate power circuit compose of thyristors perform the job of issuing the exact firing pulses for the control of the electrical appliance. By the integrated work of the Client, Server ,the micro controllers , the power circuit and our effort has lead to the building of the project,

“e-Control of Electric Drives”

1.2 SYSTEM FEATURES

The classical method which is used for the process monitoring and parameters controlling of electrical appliances is to use the digital meters and analog controlling circuits. To the maximum the controlling circuit and the device will be present near to one another. The need for the control of electrical appliances from remote places was the need of the hour. The power line communication though can help in remote monitoring its bandwidth is low and restrictions are more. The microwave communication as the name implies the cost for establishing the control laboratory and other required accessory devices are very high. As a striking balance between these two the INTERNET finds the best with the remote client and the server is possible with slight ease.

1.3 ORGANISATION OF THE THESIS

The thesis is organized in a manner which provides a clear picture of the INTERNET communications and the required hardware details. The explanation of the thesis begins with the TCP/IP Protocols explained following the concept of sockets which are the real packets for the data transfer. The unit that follows the TCP/IP is the CGI (Common Gateway Interface) which is responsible for configuring a dedicated server. The details regarding the enabling of CGI in the APACHE server is explained in this unit. The Asynchronous Serial Port programming is explained in the succeeding unit. It includes the details regarding the POSIX Interface under LINUX and the concept of customizing the TERMIOS Structure with the user defined options. The unit that finds place next discusses about the role of Micro controllers which forms the heart of the project. It compromises of the Micro controller basics, programming overview and the thyristorised power circuit which provides the way for the process control of electrical appliances.

SYSTEM COMPONENTS

SYSTEM COMPONENTS

2.1 Introduction

The project "e-Control of Electric Drives" is broadly classified into two major areas of scientific technology namely,

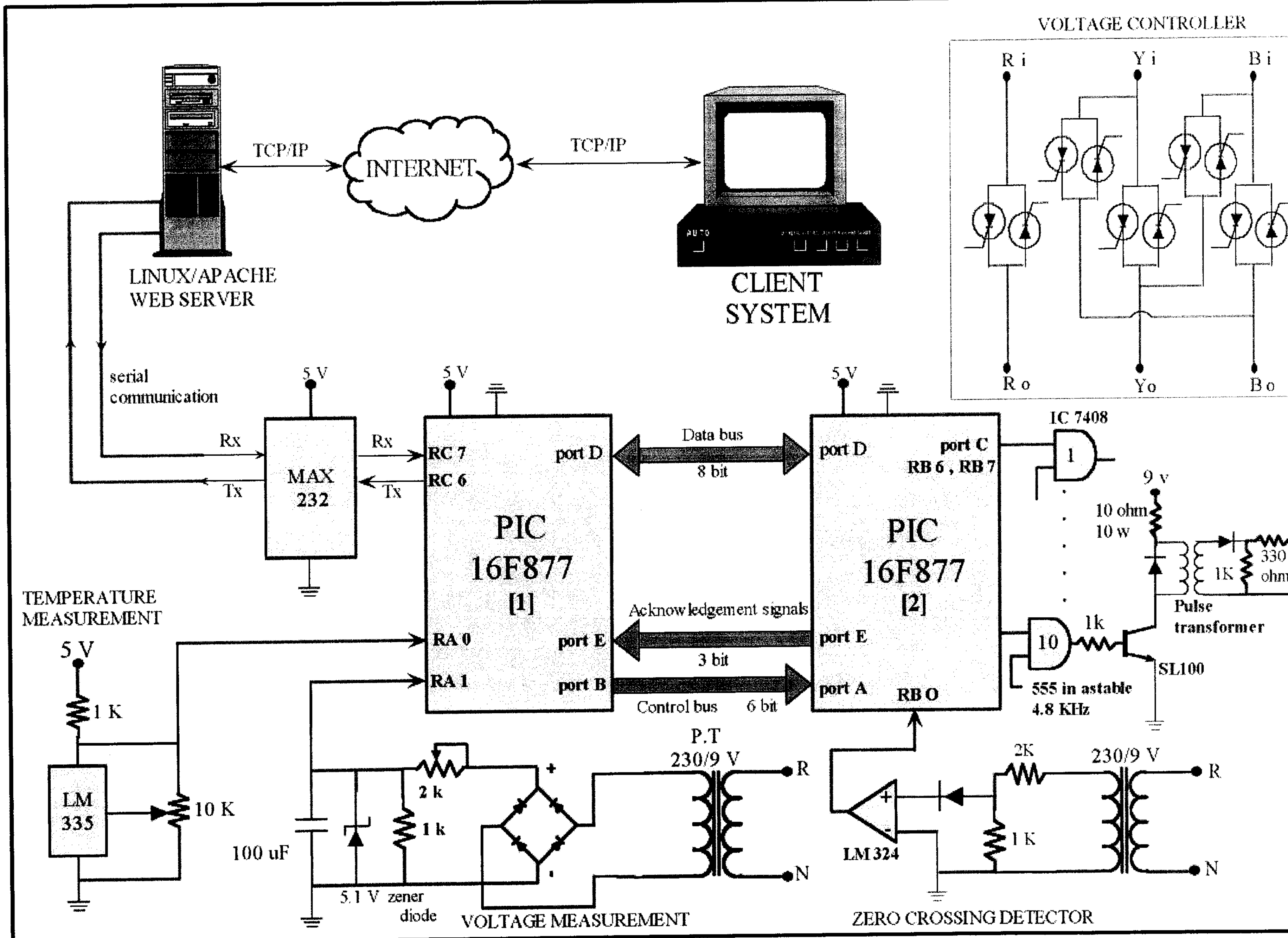
- Computer Networks
- Electrical Engineering

The portion of the project compromising of the Internet communication includes the Client and the Server. The hardware circuits are designed for controlling and monitoring the electrical appliances.

2.2 INTERNET Communication

With respect to the figure shown below the client is a remote system that can be present in any corner of the world subjected to the condition that it should be connected with the INTERNET. The dedicated server will be present near the electrical appliance whose parameters have to be controlled. The connection between the client and the server is established through the TCP/IP Protocol which also explains the concept of Peer to Peer networking. The client request are received by the server, analyzed and given as an input to the hardware circuit.

The sever communicates with the micro controllers through the Asynchronous Serial port communication. This programming is done using the POSIX Interface under LINUX.


VOLTAGE CONTROLLER
R i
Y i
B i
R o
Yo
B o
TCP/IP
INTERNET
TCP/IP
LINUX/APACHE WEB SERVER
CLIENT SYSTEM
serial communication
5 V
MAX 232
Rx
Tx
Rx
Tx
5 V
RC 7
RC 6
PIC 16F877 [1]
port D
RA 0
port E
RA 1
port B
Data bus
8 bit
Acknowledgement signals
3 bit
Control bus
6 bit
5 V
port D
port C RB 6 , RB 7
PIC 16F877 [2]
port E
port A
RB O
IC 7408
1
10
9 v
10 ohm 10 w
1k
1K
330 ohm
Pulse transformer
SL100
555 in astable 4.8 KHz
TEMPERATURE MEASUREMENT
5 V
1 K
LM 335
10 K
100 uF
5.1 V zener diode
2 k
1 k
+
-
P.T
230/9 V
R
N
VOLTAGE MEASUREMENT
LM 324
2K
1 K
230/9 V
R
N
ZERO CROSSING DETECTOR

separate start and stop bits and the parity bits. The parity check may be either odd or even but the client and the server must be synchronized to the type of parity used.

2.3 Hardware Details

The micro controllers which solely perform the job of controlling the device parameters communicate with the server through the RS232 Serial Interface. The information received in the server is then transmitted to the client through the INTERNET. Acknowledgements are used on either side for ensuring the security of transmission of data. The data between the two micro controllers is transmitted through an 8 bit data bus. The control signals and the acknowledgements are designed separately.5 bits of data are used for control signals and 3 bit for acknowledgements between the two micro controllers.

The first micro controller in addition to the processing of request also monitors the temperature and voltage of the system. The LM335 IC is used for measuring the temperature whose voltage reading is calibrated into temperature using appropriate formulas. The system voltage which has to be measured is passed to a suitable transformation ratio so that the current value of the system voltage can be monitored and updated .The second micro controller generates the firing pulses. It also counts the number of pulses between the occurrences of two output levels of the zero crossing detector by which it is able to measure the system frequency.

LINUX-The Tutankhamen Treasure

LINUX

3.1 LINUX - From the Sands...

Changes are the only things which never Changes in this ever changing World . The Earth changes its climate and the Moon changes its growth over a period of time. When the World around us crept over the Microsoft Windows operating system a group of programmers wondered about the free version operating systems and the free softwares. Among them **Richard Stallman** was the person who contributed for the development of GNU Operating system which is nothing but the open source version of UNIX. The kernel of the Linux operating system was written by **Linus Torvalds**, but Richard Stallman and his team programmers were the persons who enriched Linux with the application level tools.

In the year 1991 Torvalds announced his plans to create a free operating system. Torvalds expected help from his fellow programmers in making the system more efficient and professional. As a result over the next few years Linux developers swelled into thousands and worked to make Linux compatible with GNU programs. Vendors like **RED HAT, NOVELL, CALDERA** and **DEBIAN** created and still create popular distributions of Linux that bundle the operating system with useful programs and a graphical interface.

3.2 LINUX in our project

The reason which forces us to use LINUX in our project are because of the following characteristics

- Stability
- Customizability
- Network friendly
- Security

Stability

Besides from the aspect of an Electrical Engineer Stability plays a major role in the field of Computer Networks. Linux was developed by programmers who program for their self satisfaction and not for any corporate systems. When a work is created with the total involvement of the creator the expected mistakes are minimal.

Customizability

The main theme behind using the Linux is its source code. Because of the fact that its source code is open the programmer can customize the environment and the applications to his needs. Basically Linux is a **Multitasking** and **Multi-user** Operating system. In a Multi-user operating system the computer resources such as hard disk and memories are accessible to many users. Each user is provided with a terminal which compromises of a keyboard and a monitor which is much more economical than the single user capability. Another high light of Linux is Multitasking which allows us to carry out more than one job at the same processes is carried out.

Network Friendly

Linux has an excellent provision for communicating with fellow users. The communication may be with in the network of a single main computer or between two or more such many computer networks. The user can easily implement the **SMTP, SNMP** Protocols for the exchange of mail and network management. Linux is capable of acting as Client or Server to any popular operating system in use today and is also quite capable of being used to run **Internet Service Providers**. The web clients and the web servers communicate using the **TCP** Protocol which is an inbuilt module in Linux.

Security

Linux provides security from the mischief mongers at three different stages. The first stage of security is provided by assigning passwords and login names to individual users ensuring that nobody can access our work. At the file level there are read, write and execute permissions to each file which decide who can access a particular file, who can modify it and who can execute it.

Lastly there is file encryption. This utility encodes our files into an unreadable format so that the secrets are safe.

Above all as the motto of open source says,

"In a World with out fences, who needs gates".

INTERNET COMMUNICATONS

INTERNET COMMUNICATIONS

4.1 TCP/IP – Mysteries Unleashed

4.1.1 Communication

"Share the money or material you will lose a certain but my young friend share the knowledge you have you will receive a lot than you gave". Communication forms the basis for the sharing of knowledge. During the pre-historic times man used stones and chisels on the walls of the caves and communicated through the diagrams. When the days crept by he gave meaning to the various sounds produced by his vocal cords.

The entire field of communication changed at two instant of time. One during the invention of telephone and the other was the Computer.
Internet – The word with which our project is tied up aims for the speed control of drive. The drive whose speed is to be controlled and the user who actually controls the speed will be present in remote locations. The respective commands given by the user will be carried by the twentieth century genie (Internet) to the server to which the drive is connected. The magic behind the internet can be exploited by analyzing the **Transfer Control Protocol** and its cousin **Internet Protocol** which is done in the foregoing chapters.

4.1.2 TCP/IP – Introduction

Most network applications can be divided into two pieces: a client and a server. When a client application communicates with the server application regulations are called the **Networking Protocols**. Web clients and servers communicate using TCP Protocol. TCP in turn uses the IP Protocol and the IP communicate with the data link layer of certain form.TCP and IP are usually the part of the Protocol stack with in the kernel. The IP Protocol has two different versions IPv4 and IPv6.The IP Protocol can be directly accessed by, bypassing the transport layer called Raw Sockets but which is used less frequently.

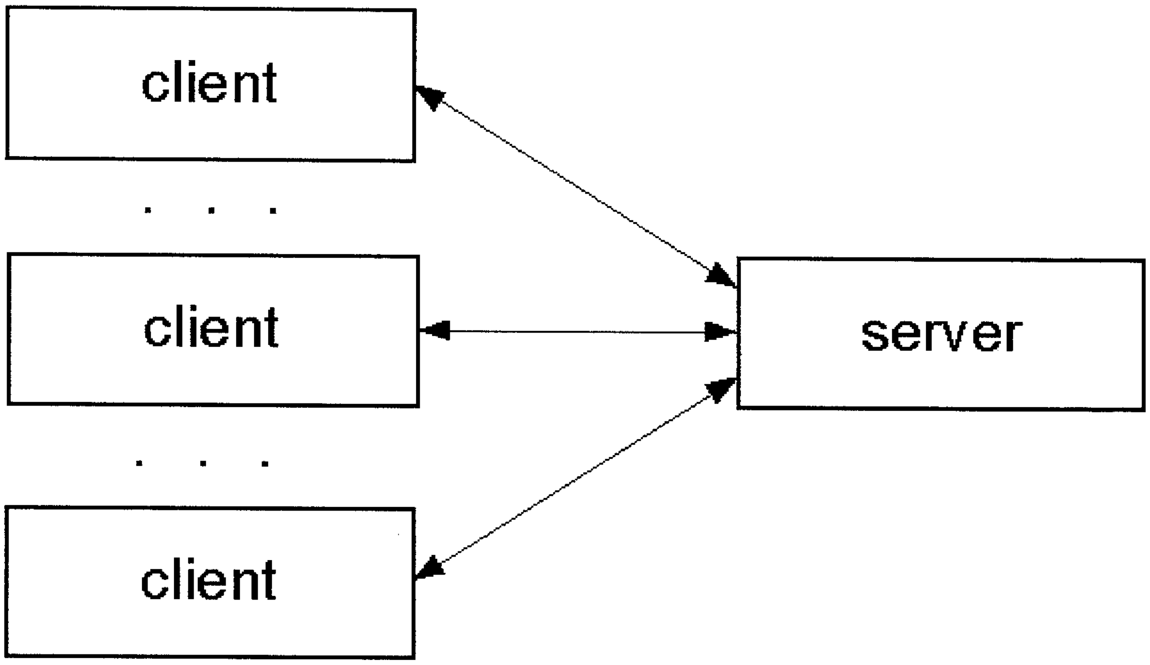


Figure 4.1. Client – Server Architecture

4.1.3 TCP – Distinguishing features

The features of TCP when understood make it easier for us to write robust clients and servers. The main feature of TCP is the three way hand shake.

server across the connection and then terminates the connection. When TCP sends data to other end it requires an acknowledgement in return. If the acknowledgement is not received TCP automatically retransmits the data and waits for a longer amount of time. After some number of retransmissions TCP will give up with the total amount of time spent trying to send data typically between 4 to 10 minutes.

TCP contains the algorithms to estimate Round Trip Time (RTT) between client and server dynamically so that it knows how long to wait for an acknowledgement.TCP also sequences the data by associating the sequence number with every byte that it sends. If we want to send an application consisting of 2048 bytes TCP sends it in two segments: 1 – 1024 in the first segment and 1025 – 2048 in the second. If the segments arrive out of order the receiving TCP will reorder them based on their sequence number numbers before passing to their receiving application.

TCP provides **Flow Control**. TCP always tells it peer exactly how many bytes of data it is willing to accept from the peer. This is called the **Advertised Window.** The window size changes dynamically over time (i.e.) as the data is received from the sender the window size decreases, but as the receiving applications reads the data from the buffer the window size increases.

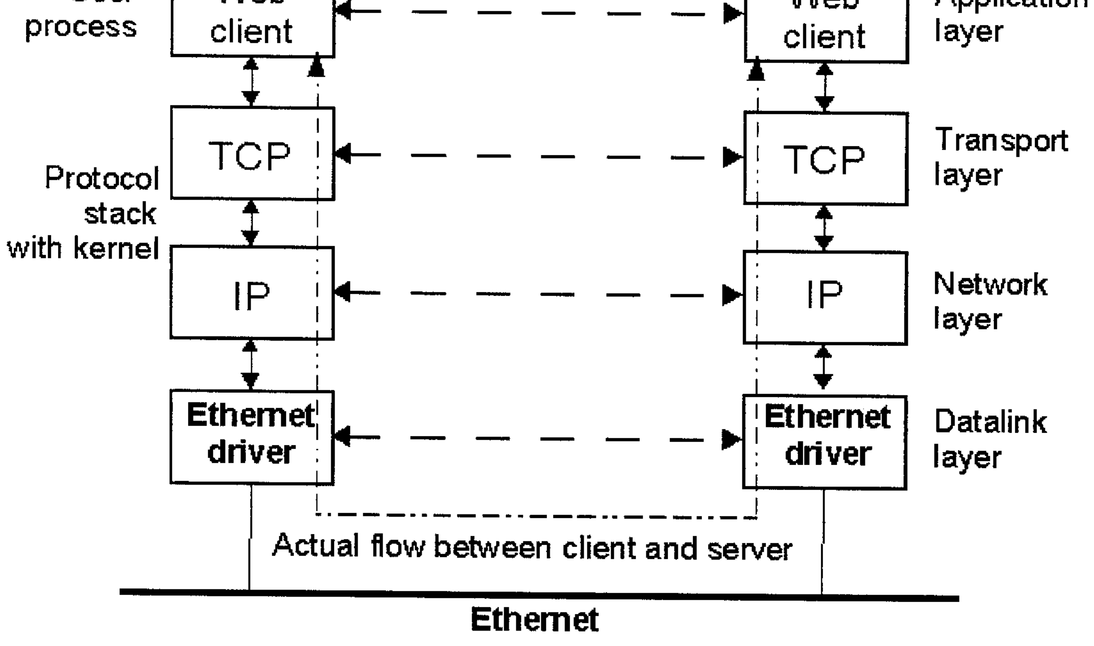


Figure 4.2. Client – Server Communication

4.1.4 Three Way Handshake

The following scenario occurs when the TCP connection is established.

1. The server must be prepared to accept an incoming connection. This is normally done by calling **Socket, Bind** and **Listen** and this is called a **Passive Open**.
2. The client issues an **Active Open** by calling **Connect.**

   This cause the client TCP to send an SYN segment to tell the server the clients initial sequence number for the data that the client will send on the connection.
3. The server must acknowledge the clients SYN and the server must send its own SYN containing the initial sequence number for the data that the server will send on connection. The server sends the SYN and the ACK in a single segment.

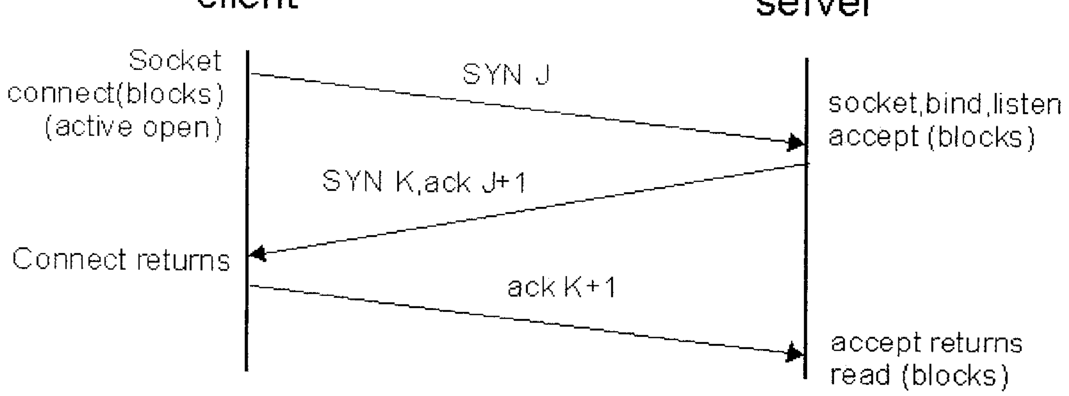


Figure 4.3. Three Way Handshake

4. The client must acknowledge the servers SYN.The minimum number of packets required for this exchange is three. Hence this is called **TCP's Three Way Handshake.**

4.1.5 TCP Connection Termination

While it takes three segments to establish a connection, it takes four to terminate a connection.

1. One application calls **Close** first and we say that this end perform the **Active Close**. This end's TCP sends FIN segment which means it is finished sending data.
2. The other end receives the FIN performs the **Passive Close**. The received FIN is acknowledged by TCP. The receipt of FIN is also passed to the application as an end of file.
3. Some time later the application that received the end of file will close the socket.
4. The TCP on the system that receives the final FIN acknowledges the FIN.

Since the FIN and the ACK are required in each direction four segments are normally used. Although we show that the client

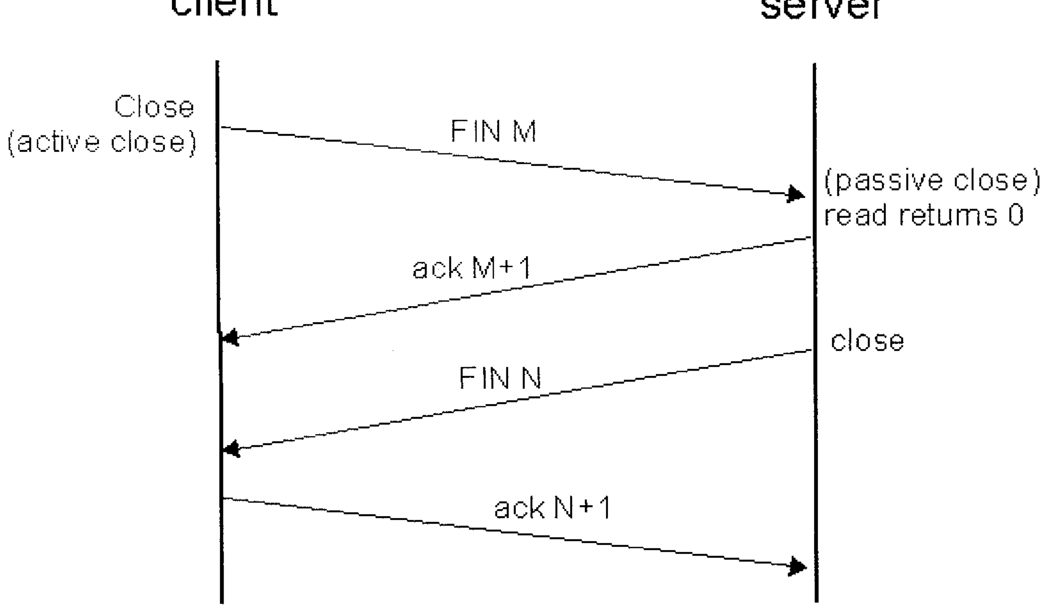


Figure 4.4. TCP Connection Termination

performs the active close, either end the client or the server can perform the active close. **In our project the client always request for the closing of connection (i.e.) the active close.**

4.1.6 TCP Port Numbers

At any time multiple processes can use TCP.TCP use 16 bit integer port numbers to differentiate between these processes. When the client wants to communicate with the server the client must identify the server with which it wants to communicate. For example TCP/IP implementation that supports FTP assigns the well known port 21 to the FTP server. Clients on the other hand use **Ephemeral Ports** that is short lived ports. These port numbers are automatically assigned by the TCP to the client. Clients normally do not care about the value of the Ephemeral Port. The client just needs to be certain that the Ephemeral Port is unique on the client host.

What are Sockets?

Sockets are mechanisms that provide access to TCP/IP on most system. It is the name given to a package of subroutines that provide this access. Sockets provide the infra structure for two computer processes to exchange the data between them. They provide network communication infrastructure. Analogous to a telephone conversation, where both parties require a telephone instrument, there exists a socket at end of a network connection.

To work with sockets there are three basic steps.

1. Create a connection between two processes.
2. Read/Write data.
3. Close the connection.

For any TCP connection there are 4 pieces of data that bind the two connections together.They are,

1. Local IP address.
2. Local TCP port.
3. Remote IP address.
4. Remote TCP port.

The address and a port that uniquely identify an end point are together called a socket.

Communication through sockets takes place in two ways. One is **Stream-Oriented** and other is **Datagram-Oriented.** Stream oriented communication is also referred as **Connection-Oriented** while datagram is referred as **Connectionless.**

There are two terms associated with sockets they are the **Family** and **Type**. The family represents the protocol family. There are currently five protocols in use. They are,

1. IPv4 Protocol.
2. IPv6 Protocol.
3. UNIX Domain Protocol.
4. Routing Sockets.
5. Key Sockets.

There are three types of sockets. They are,

1. Stream Socket.
2. Datagram Socket.
3. Raw Socket.

In our project we are using Stream Type IPv4 Version Protocol.

The concept of stream type sockets can be understood by knowing the socket address structure. Each Protocol should define its own structure for socket address. Every structure begins with **sockaddr_** and is suffixed with the specific protocol.

For our project the structure begins with sockaddr_in

Most functions in the API require a pointer a socket address structure as an argument.

all ready defined in <netinet/in.h>.

The IPv4 socket address is referred as **Internet Socket Address Structure.** The sockaddr_in is defined as follows

```
Struct sockaddr_in
{
        Uint8_t             sin_len;
        Sa_family_t         sin_family;
        In_port_t           sin_port;
        Struct in_addr      sin_addr;
        Char                sin_zero[8];
                                      };
```

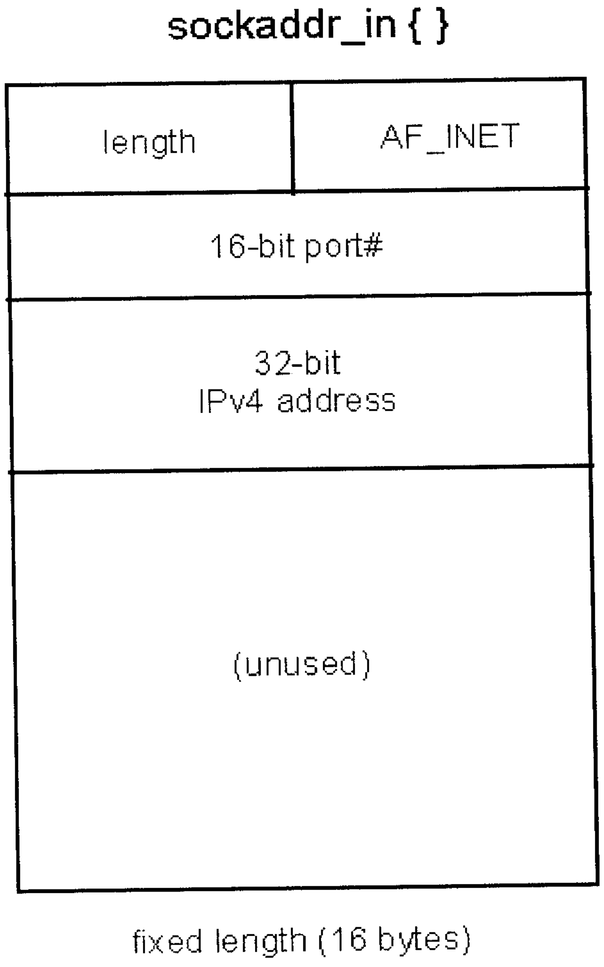


Figure 4.5. Socket Address Structure

The first member in sockaddr_in is the length of the structure. The second is the protocol family, which is **AF_INET** for Internet. The third member is a 16 bit TCP Port. The fourth one is the 32 bit IPv4 address.

4.2.2 Connection Oriented Sockets

The TCP sockets are connection oriented. A TCP client establishes a connection with the server, communicates with the server and finally closes the connection.Untill the connection is terminated no other connection can be established through the same port.

A socket has a typical flow of events. The socket on the server waits for a request from the client. The server establishes an address that clients can use to find the server. When the address is established the server waits for client to request a particular service. The data exchange between the client and server takes place when the connection is established through a socket.

The connection between the client and the server depends upon the filling up of the socket address structure with appropriate byte order. There are two ways by which an integer is stored .The two methods are,

- Little Endian
- Big Endian

The POSIX.1g specifies that some of the field in the socket address structure need to be the network byte order. The function which is used to convert from one byte order to other is as follows

```
Unit32_t htons(uint32_t host 32 bit value);
```

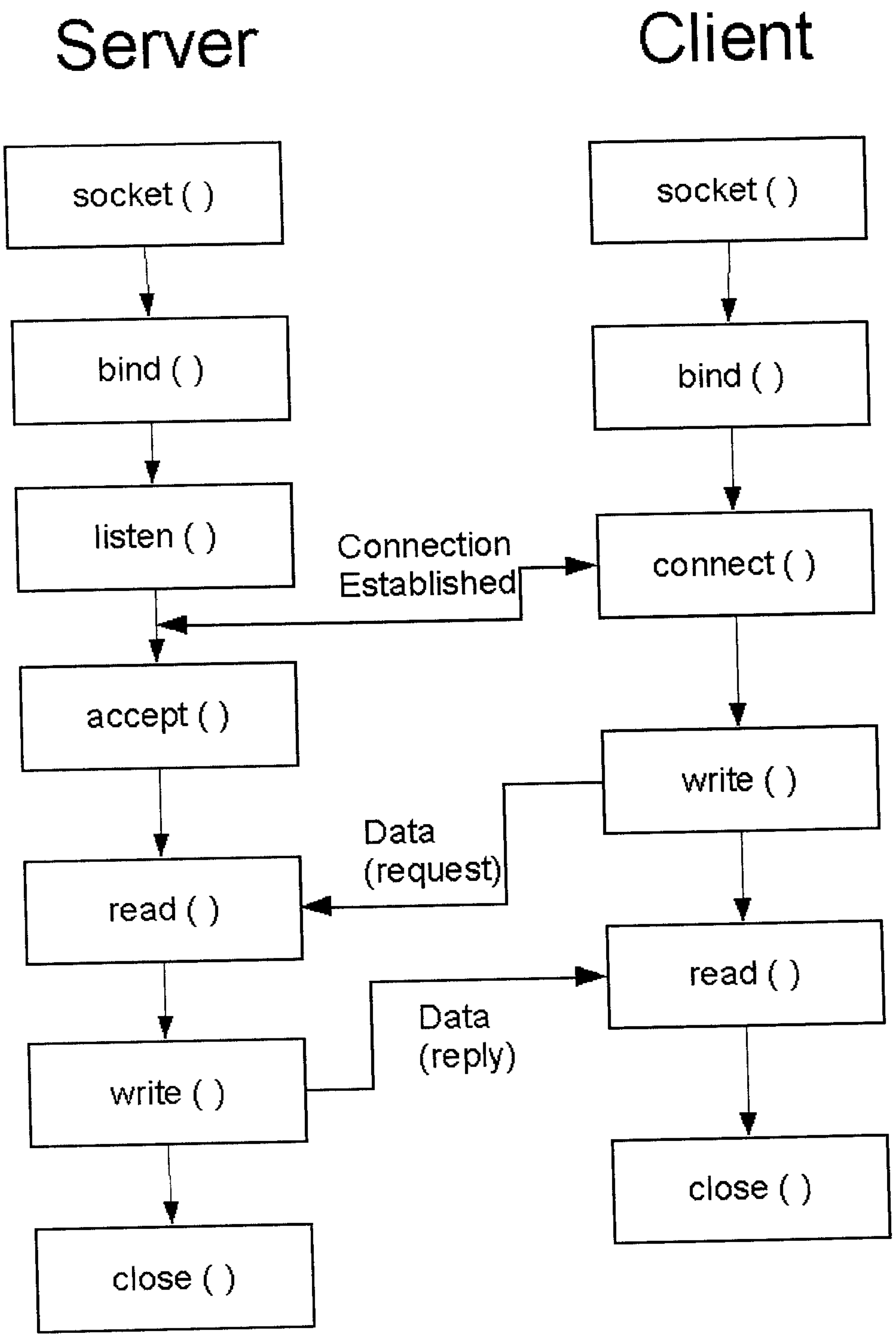

Server
Client
socket ( )
socket ( )
bind ( )
bind ( )
listen ( )
Connection Established
connect ( )
accept ( )
write ( )
Data (request)
read ( )
read ( )
Data (reply)
write ( )
close ( )
close ( )

figure 4.6

1. The Server creates a socket for end point communication.
2. The Server binds the socket to a unique name.
3. Now the Server waits for the client to connect to it.
4. When the client connects the server accepts the client request for connection.
5. The Server then sends and receives commands and acknowledgements from the client.
6. Finally when the server wants to terminate operations it closes the socket.

At the Client End,

1. The client creates a socket for end point communication.
2. Optionally the client can bind a unique name to the socket.
3. The Client then connects to the server.
4. Data is sent and received by the Client
5. The Client closes the socket when it desires to stop the operations.

The API functions used on the Server End are ,

1. socket ( )
2. bind ( )
3. listen ( )
4. accept ( )
5. read ( )
6. write ( )
7. close ( )

The API functions used on the Client End are,

1. socket ( )
2. connect ( )
3. write ( )
4. read ( )
5. close ( )

socket ( )

Both the client and the server process need to invoke the socket for signaling their respective end points for communication. The function takes three arguments.

1. Protocol Family
2. Socket Type
3. Protocol

The Protocol Family is also called as the **Domain. In our project the socket type is SOCK_STREAM and the Family is AF_INET.**

The Prototype for the function is

int socket (int family, int type, int protocol);

The function returns a non negative integer on successful creation of sockets and -1 on error. The third argument is set to zero for all sockets except Raw sockets. The value returned by the function is referred as sockfd .

bind ( )

The bind function is used to assign a local protocol address to the socket. It binds the socket with the unique name. The prototype of the function is,

int bind (int sockfd, struct sockaddr_in * addr, socklent_t addrlen);

Where,

sockfd is socket descriptor.

addr is the reference to the socket address structure

addrlen is size of socket address structure.

The TCP Server calls this function. The function indicates the willingness to accept client connection request. When a listen function is called on a socket the socket becomes a passive socket. It can not initiate any connection request. The prototype of the function is,

int listen (int sockfd, int backlog);

Where,

sockfd is socket descriptor.

backlog gives the maximum number of connections the Kernel can queue up for this socket.The function returns zero on success and -1 on error.

accept ( )

The TCP Server calls this function. The server application uses this function to accept the client. The completed connection in completed queue is returned when function is called. The prototype of the function is,

int accept (int sockfd,struct sockaddr * cliaddr, socklen_t * addrlen);

Where,

sockfd is socket descriptor.

cliaddr is client's socket address structure.

addrlen is the size of client's socket address structure.

The function returns zero on success and -1 on error.

connect ( )

The connect function is invoked by TCP Client. The function initiates the three way hand shake .The function returns a value only when the connection is established or an error is occurred. The prototype of the fuction is,

int connect (int sockfd, struct sockaddr * servaddr, socklen_t addrlen);

sockfd is socket descriptor.

servaddr is server's socket address structure

addrlen is the size of server's socket address structure.

The function returns zero on success and -1 on error.

read ( )

The function is used for reading bytes from the given descriptor. The prototype of the function is,

read(int sockfd, void * buff, sizeof buff);

Where,

sockfd is socket descriptor.

buff is the structure into which the data is to be read.

The function returns zero on success and -1 on error.

write ( )

The function is used for writing bytes to the given descriptor. The prototype of the function is,

write(int sockfd, void * buff, sizeof buff);

Where,

sockfd is socket descriptor.

buff is the structure into which the data is to be written.

The function returns zero on success and -1 on error.

close ( )

The function is used to close an open socket. The prototype of the function is ,

int close (int sockfd);

Where,

sockfd is socket descriptor.

The function returns zero on success and -1 on error.

4.3.1 Introduction

"Face Behind Interface"

The beauty of any project lies in the manner how it is presented. The presentation has to be in a user friendly manner which unanimously leads to the graphical interface.The APACHE Server is the tool which establishes the required connection with the user requested URL. The APACHE Server runs in the most popular operating system available such as LINUX , MACOS and even in WINDOWS.

The problem of creating the graphical interface was really a problem until the CGI and GTK tools were established. CGI as the name suggest it stands for Common Gateway Interface. The applauding factor behind the CGI is that the programmer can use any of the high level languages especially C ,C++, PERL and PYTHON .The headers and the footers which create an unique interface in the web layout is produced by using the normal HTML Procedures. The APACHE Server needs the configuring of the CGI that is the CGI Tool has to be enabled. The process of configuring and producing graphical based request driven options is explained in the forth coming chapters.

In order for the proper working of CGI programs we need to have Apache configured to permit CGI execution. There are several ways to do this.

Script Alias

The Script Alias directive tells Apache that a particular directory is set aside for CGI programs. Apache will assume that every file in this directory is a CGI program, and will attempt to execute it, when that particular resource is requested by a client.

The Script Alias directive looks like:

```
ScriptAlias /cgi-bin/ /usr/local/apache/cgi-bin/
```

The example shown is from your default httpd.conf configuration file, if you installed Apache in the default location. The ScriptAlias directive is much like the Alias directive, which defines a URL prefix that is to mapped to a particular directory. Alias and ScriptAlias are usually used for directories that are outside of the Document Root directory. The difference between Alias and ScriptAlias is that ScriptAlias has the added meaning that everything under that URL prefix will be considered a CGI program. So, the example above tells Apache that any request for a resource beginning with /cgi-bin/ should be served from the directory /usr/local/apache/cgi-bin/, and should be treated as a CGI program.

For example, if the URL http://www.example.com/cgi-bin/test.pl is requested, Apache will attempt to execute the file /usr/local/apache/cgi-bin/test.pl and return the output. Of course, the file will have to exist, and be executable, and return output in a particular way, or Apache will return an error message.

CGI programs are often restricted to ScriptAlias'ed directories for security reasons. In this way, administrators can tightly control who is allowed to use CGI programs. However, if the proper security precautions are taken, there is no reason why CGI programs cannot be run from arbitrary directories. For example, you may wish to let users have web content in their home directories with the UserDir directive. If they want to have their own CGI programs, but don't have access to the main cgi-bin directory, they will need to be able to run CGI programs elsewhere.

Explicitly using Options to permit CGI execution

You could explicitly use the Options directive, inside your main server configuration file, to specify that CGI execution was permitted in a particular directory:

```
<Directory /usr/local/apache/htdocs/somedir>
    Options +ExecCGI
</Directory>
```

The above directive tells Apache to permit the execution of CGI files. You will also need to tell the server what files are CGI files. The following Add Handler directive tells the server to treat all files with the cgi or pl extension as CGI programs:

```
AddHandler cgi-script cgi pl
```

.htaccess files

A .htaccess file is a way to set configuration directives on a per-directory basis. When Apache serves a resource, it looks in the directory from apply directives found therein. .htaccess files can be permitted with the AllowOverride directive, which specifies what types of directives can appear in these files, or if they are not allowed at all. To permit the directive we will need for this purpose, the following configuration will be needed in your main server configuration:

```
AllowOverride Options
```

In the .htaccess file, you'll need the following directive:

```
Options +ExecCGI
```

which tells Apache that execution of CGI programs is permitted in this directory.

4.3.3 Form Processing

The interface consisting of the radio buttons, text blocks and various other graphical themes is coded using the concept of Forms. The Forms are really important in transmitting the data beween the client and the server. The details regarding the characteristics of the applications employed say the firing data, the direction of rotation and various other details are provided to the client as the block. These details when received in the server ,they are parsed and given one by one on que basis to the micro controllers.

Forms are used to collect information. The information must be handled by a program running on the server. This is often called CGI or Common Gateway Interface. The web server is actually a program running somewhere on a host computer. This normally doesn't do anything but serve web pages. It will pass on the form data to another program running third program on the host. Depending on the outcome there may be feedback to the originating web browser.

Scripts

Form output is often handled by some kind of script. It is possible to write dedicated software for this, but scripts are generally easier to write and maintain. A script will often run unaltered on windows as well as a mac or unix based machine. The most popular scripting language is PERL, used originally for processing large amounts of text on unix computers. In our project we used C as our scripting language which can be ported to other operating systems.

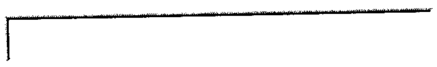
Text Box Processing

The input tag defaults to a one line text box. The command used for creating the text box for entering the firing data is as follows,

```
<INPUT TYPE="text">
```

We can also specify the size of the box. We also use a different attribute: SIZE. This size bears no relation to the amount of characters we can put inside. The box looks as follows,

```
<INPUT TYPE="text" SIZE="40">
```

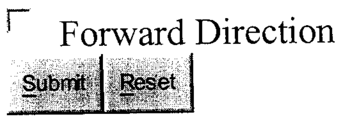


We may limit the amount of text someone may put inside a box. We can limit the size with the MAXLENGTH attribute.

Handling the Check Boxes

Often we will only need to know if something is true or not. To this purpose we are using the checkbox. Next to the checkbox we will display a text, containing a statement. Like "Check this box if you want forward or reverse direction."

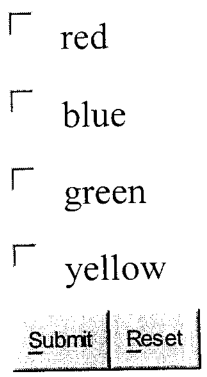
```
<INPUT TYPE="checkbox" NAME="Forward" VALUE="true">
```


Forward Direction

As we might expect right now, this is just another incarnation of the input tag. This time the type is set to "checkbox". The value is not shown in your browser. The name and value are only submitted if the box is checked, otherwise it is ignored. we can also use this tag to make lists with multiple checkboxes.

```
<INPUT TYPE="checkbox" NAME="color_red" VALUE="true">
<INPUT TYPE="checkbox" NAME="color_blue" VALUE="true">
<INPUT TYPE="checkbox" NAME="color_green" VALUE="true">
<INPUT TYPE="checkbox" NAME="color_yellow" VALUE="true">
```

I like the following colors:


red

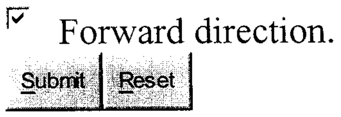
blue

green

yellow

tag will fix the problem.

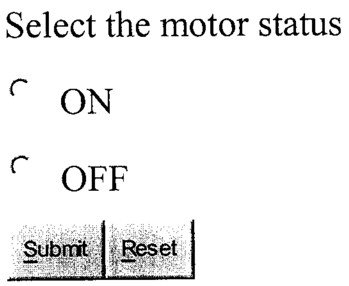
```
<INPUT TYPE="checkbox" NAME="Forward" VALUE="true"
CHECKED>
```


☑ Forward direction.

Processing of Radio Buttons

Sometimes we want to choose only one option out of a list. Like the power status of the drive say ON or OFF condition.

```
<INPUT TYPE="radio" NAME="Power Status" VALUE="ON">
<INPUT TYPE="radio" NAME="Power Status" VALUE="OFF">
```

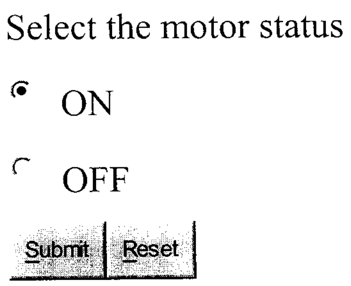

Select the motor status

○ ON

○ OFF

To this purpose we will use a set of radio buttons. This name refers to radio with different presets, each setting a different channel. we can only listen to one channel at a time. Pushing one button will switch any other one off. All buttons in a set must have the same name. The values will probably differ, although that is not obligatory.

```
CHECKED>
```



Select the motor status

[✓ ON]

OFF

So we already set that on with the CHECKED attribute. If we use this attribute more than once, only the last one will be active. If we change the value, and then press the reset button, the default (checked) value will be selected again.

Mysteries and Myths of
SERIAL PORT PROGRAMMING

SERIAL PORT COMMUNICATION

5.1 Serial Vs Parallel Communication

'Adversity is the touch stone of achievements'. The difficulty of Serial interfacing and serial port programming is overwhelmed once the basics of serial data transmission and the use of handshaking signals defined for RS 232 Interface is understood properly. The reason we are not opting for parallel port interface is that a wire for each bit is needed and more over when the source and destination are few feet apart a parallel cable can be bulky and expensive. As our project mainly concerns an Internet communication where the client and the server might be even located at a very large distance.

In addition long runs of parallel can act as a transmission line that is susceptible to reflections and induce noise. The Serial I/O Techniques will definitely offer an optimum solution to these problems.

Data to be transferred are sent one bit at a time using fewer wires. By defining appropriate standards for logic levels we can reduce the effects of long transmission lines and combat noise problems.

5.2 Asynchronous Serial Communication

In case of Asynchronous data transmission the information that has to be sent is broken up into numerous bits and each bit is transferred at a single instance of time. So separate markers are required at the starting point and ending point of the information. This function is performed by the start and the stop bit. There is no fixed criteria for arrival of information

information until another start bit is encountered.

For the computer to understand the serial data coming into it, it needs some way to determine where one character ends and the next begins. In asynchronous mode the serial data line stays in the mark state until a character is transmitted. A **start** bit precedes each character and is followed immediately by each bit in the character, an optional parity bit, and one or more **stop** bits. The start bit is always a space (0) and tells the computer that new serial data is available. Data can be sent or received at any time, thus the name asynchronous.

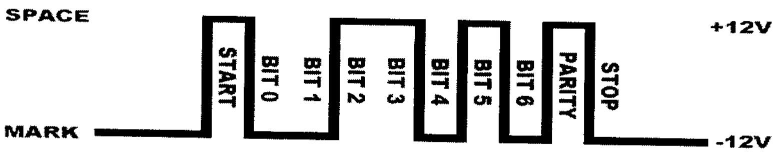


Figure 5.1 - Asynchronous Data Transmission

The optional parity bit is a simple sum of the data bits indicating whether or not the data contains an even or odd number of 1 bit. With **even parity,** the parity bit is 0 if there is an even number of 1's in the character. With **odd parity,** the parity bit is 0 if there is an odd number of 1's in the data. You may also hear the terms **space parity, mark parity**, and **no parity.** Space parity means that the parity bit is always 0, while mark parity means the bit is always 1. No parity means that no parity bit is present or transmitted. The remaining bits are called stop bits. There can be 1, 1.5, or 2 stop bits between characters and they always have a value of 1. Stop bits traditionally were used to give the computer time to process the previous character, but now only serve to synchronize the formats are usually expressed as "8N1", "7E1", and so forth. These stand for "8 data bits, no parity, 1 stop bit" and "7 data bits, even parity, 1 stop bit" respectively.

Full Duplex and Half Duplex

Full duplex means that the computer can send and receive data simultaneously - there are two separate data channels (one coming in, one going out).**Half duplex** means that the computer cannot send or receive data at the same time. Usually this means there is only a single data channel to talk over. This does not mean that any of the RS-232 signals are not used. Rather, it usually means that the communications link uses some standard other than RS-232 that does not support full duplex operation.

There are also facilities available for the checking of proper reception. It is provided by means of parity bits. These are nothing but the bits which indicate just the presence of error and not the error itself. As known the two types of parity are ,

1. Even parity
2. Odd parity

The former requires an even number of bits in the receptor. If the number of bits are odd in the transmitting end the parity generator present at the transmitting end will append an logical high bit in order to create an even parity. Analogously the latter requires an odd number at the receptor and if not present it indicates an error otherwise.

5.3 Terminologies in UNIX Serial Communication

Serial port devices are called TTY devices on all UNIX systems. TTY is communication devices. The name today is a misnomer because ISDN terminal adapters and medical magnetic resonance imaging equipment can be and often are connected to TTY devices. The data rate at which asynchronous serial data is transmitted is typically expressed in bits per second (bps).The term baud is often used when referring to the data rate. Baud is actually the number of signal transitions that occur per second, and it may or may not match the actual data rate. With asynchronous communications, the baud rate matches the data rate because only two signals exist on the wire, but there are other types of communications in which this may not be true.

Flow control is the mechanism by which the receiver informs the sender to stop and restart data flow. Flow control is used so the sender does not send data faster than the receiver can receive and process it. Flow control is either software (in-band (within the data stream) using special control characters such as XON and XOFF) or hardware (out-of-band (outside of the data stream) using additional wires and signals such as Request to Send (RTS) or Data Terminal Ready (DTR)). If flow control is not enabled correctly at both the sender and receiver, data may be lost. Asynchronous serial data is transmitted with a start bit of 0; 5 to 8 bits of character data; an optional parity bit; and 1 or 2 stop bits. If there is a mismatch in the data rate, parity setting, stop bits, or character size, the receiving serial port hardware will detect a framing error, parity error, or both.

A break is a special condition in which the transmit line is asserted for at least 250 milliseconds. (**Asserted** means that the voltage on the line is changed to mark a new condition, in this case a break condition.) Break conditions are used for exceptional events such as the need to change to a have a central processing unit) are Data Terminal Equipment (DTE). Most serial printers also are DTE. Data Communications Equipment (DCE) are devices such as modems and other data communications gear. In our project the Computer acts as the Data Terminal Equipment and the Microcontroller acts as the Data Communication Equipment.

The terminology of serial port wires such as Transmit Data (TD), Receive Data (RD), Data Carrier Detect (DCD), and Request to Send (RTS) is written from the perspective of the DTE end of the link. An output on DTE, such as transmit, becomes an input on DCE. When a DTE device is connected to a DCE device, a straight through cable is used (TD connects to TD, RD connects to RD, and so on).The following diagram illustrates the simplest form of serial communications using three wires: TD, RD, and a common signal ground (SG). Flow control must be software flow control (that is, in-band flow control using XON/XOFF). Three-wire connections are popular because of cable cost, and they are adequate for transferring text up to 38,400 bps (higher data rates are possible).

```
------------------------------

        DTE          DCE

------------------------------

         TD   ---->   TD
         RD   <----   RD
         SG   -----   SG

------------------------------

    Three-Wire Connection

------------------------------
```

Figure 5.2. Serial Communication

Data Carrier Detect (DCD) signals. When a serial port is opened, DTR is asserted. At the remote end, DTR is connected to DCD. If DCD becomes asserted, an application knows that the remote side is ready to receive and send data. When DCD is unasserted (that is, the voltage on the DCD line is changed to mark a new condition), the application knows that the remote application has closed the TTY line and is not available to send or receive additional data. A modem on a phone line, for example, will unassert DCD when the phone connection is closed. DTR and DCD are not typically used for flow control but rather for signaling the on or off state of a serial device.

5.4 Programming Overview

At the most basic level, you want to write data to a serial port and read from it. You should use the basic **open, read, write,** and **close** functions to write to and read from a serial port. These basic read and write functions, and others that are used to retrieve and manipulate serial port parameters, have been standardized in the Portable Operating System Interface Part 1 (POSIX.1) standard. POSIX.1 defines the C language application interface for input and output, among other things. The POSIX.1 functions **tcgetattr()** and **tcsetattr(),** and other similar functions described in the **termios.**

When using the basic **open(), read(),** and **write()** functions, you are using the default parameters of the serial port, and we are in canonical mode. ***Canonical mode*** means that a **read()** operation returns control to the application when a line of text has been entered, with a line of text being delimited by an end of line (EOL) character such as a carriage return. The default parameters include parameters such as the data rate (9600 bps), parity). Some processing of input and output data typically also occurs. This processing includes interpreting and taking action on control characters.

The parameters of a TTY device are controlled in the termios structure, which is defined in <sys/termios.h>. The termios structure is defined as follows,

```
struct termios
{
   tcflag_t   c_iflag;    /* input modes */
   tcflag_t   c_oflag;    /* output modes */
   tcflag_t   c_cflag;    /* control modes */
   tcflag_t   c_lflag;    /* line discipline modes */
   speed_t    c_ospeed;   /* output speed */
   speed_t    c_ispeed;   /* input speed - not supported */
   cc_t       c_cc[NCCS]; /* control chars */
};
```

The **c_iflag** parameter controls the input of data, with flags such as **IGNPAR**(ignore parity errors), **ISTRIP**(strip the high-order bit), and **IUCLC** (convert uppercase to lowercase). As with all flags for **termios** structure elements, these are defined in **<sys/termios.h >.** We can combine these flags for the desired combination of processing. The **c_oflag** parameter contains flags that control output processing. For example, the **OCRNL** flag maps carriage return (CR) to new line (NL) characters. You must set the **OPOST** flag if any output processing is to behavior such as what to do when DCD is asserted or unasserted **(CLOCAL),** the character size **(CSIZE),** and if hardware flow control should be enabled **(CNEW_RTSCTS).**

The **c_lflag** parameter affects the processing between the TTY device and the user. The **ICANON** flag enables or disables canonical mode, and other flags control processing such as character echo-back and signal generation. The characters in the **c_cc** array define control characters such as the erase, kill, end-of-file (EOF), START, and STOP characters. The key you press to kill a program (such as **Delete** or **Ctrl-C**) is determined by the **VKILL** entry in **c_cc** array. START is typically XON, and STOP is typically XOFF.

5.5 Programming with POSIX

Simple line-mode input and output is relatively straightforward, as shown

```
int fd,rc;
int bufsize = BUFSIZE;
char buffer[BUFSIZE];


fd = open("/dev/ttys0",O_NONBLOCK | O_RDWR);
rc = read(fd,buffer,bufsize);   /* blocks until data w/ CR available */
write(fd,buffer,rc);            /* just write it all back */
close(fd);                      /* close completes when all data written */
```

Setting the Baud Rate

The baud rate is stored in different places depending on the operating system. Older interfaces store the baud rate in the **c_cflag** member using provide the **c_ispeed** and **c_ospeed** members that contain the actual baud rate value.

The **cfsetospeed** and **cfsetispeed** functions are provided to set the baud rate in the **termios** structure regardless of the underlying operating system interface.

Setting the baud rate.

```
struct termios options;
/*
 * Get the current options for the port...
 */
tcgetattr(fd, &options);
/*
 * Set the baud rates to 19200...
 */
cfsetispeed(&options, B19200);
cfsetospeed(&options, B19200);
/*
 * Enable the receiver and set local mode...
 */
options.c_cflag |= (CLOCAL | CREAD);
/*
 * Set the new options for the port...
 */
tcsetattr(fd, TCSANOW, &options);
```

current serial port configuration. After we set the baud rates and enable local mode and serial data receipt, we select the new configuration using **tcsetattr.** The **TCSANOW** constant specifies that all changes should occur immediately without waiting for output data to finish sending or input data to finish receiving. There are other constants to wait for input and output to finish or to flush the input and output buffers.

Constants for tcsetattr

Constant	**Description**
TCSANOW	Make changes now without waiting for data to complete
TCSADRAIN	Wait until everything has been transmitted
TCSAFLUSH	Flush input and output buffers and make the change

Table 5.1. Termios Structure

Most systems do not support different input and output speeds, so be sure to set both to the same value for maximum portability.

Setting the Character Size

Unlike the baud rate, there is no convenience function to set the character size. Instead you must do a little bit masking to set things up. The character size is specified in bits:

```
    options.c_cflag |= CS8;    /* Select 8 data bits *
```

Setting Software Flow Control

Software flow control is enabled using the **IXON, IXOFF,** and **IXANY** constants:

```
    options.c_iflag |= (IXON | IXOFF | IXANY);
```

To disable software flow control simply mask those bits:

```
    options.c_iflag &= ~(IXON | IXOFF | IXANY);
```

The XON (start data) and XOFF (stop data) characters are defined in the **c_cc** array described below.

THREE PHASE VOLTAGE CONTROL
The Odyssey Rebuilt

THREE PHASE VOLTAGE CONTROL

6.1 Three Phase Voltage Controller

The realization of the voltage control required by the client is fulfilled through the three phase voltage controlling power circuit which is realized using thyristors separately for forward and reverse direction. The thyristors perform the function of controlling the line voltage with respect to the cut in voltage required. In our project the reverse direction of rotation is achieved by changing the phase sequence, that is the phase sequence RYB which represents the forward direction is changed as RBY.Three pair of thyristors connected in anti parallel support the forward direction. The thyristors in anti parallel direction is required for conducting the three phase waveform during the positive and negative half of the cycle in continuous conduction mode.

The firing angle that is given by the client is transmitted to the micro controller 1 through the server. The micro controller 1 employs an interrupt based service routine to receive the request in addition to the process of sensing the temperature and voltage. The micro controller 1 also monitors the current system frequency on the basis of the count of pulses from the zero crossing detector circuit. The micro controller 2 plays an leading role in the generation of firing pulses. It receives the required firing angle from the micro controller 1 through a 8 bit transfer of data through the parallel port. The program required for the generation of a firing signal is produced for a specified period of time interval. In other words , an exact time delay is generated for the equivalent firing angle.

controllers in detail.

6.2 Role of Micro Controllers

The Microcontrollers play the major role of generating and monitoring the various controlling signals and acknowledgements flowing in and around the entire Hardware module. The evolution of microcontroller technology is marked by a major fork as it forms the evolutionary tree represented by the power pc chips which form the heart of the various versatile applications in which they are employed.

6.2.1 Microcontrollers an Introduction

The microcontroller is a stand alone warrior which is embodied with all kinds of weapons required for the field in which it is used. It has an inbuilt CPU,Memory and Peripherals and more over it is a general purpose device. The upper hand of the micro controllers stands in its bit handling instructions, where as the microprocessors requires a numerous numbers of operational codes for moving the data to and fro.
The latter don't have any inbuilt Peripherals. The microcontrollers work much faster than the microprocessors because of the rapid movement of the bit within the chip. Microcontroller differs from a microprocessor in many ways. First and the most important is its functionality. In order for a microprocessor to be used, other components such as memory, or components for receiving and sending data must be added to it. In short that means that microprocessor is the very heart of the computer. On the other hand, microcontroller is designed to be all of that in one. No other

peripherals are already built into it. Thus, we save the time and space needed to construct devices.

6.2.2 PIC Microcontroller: A Frame Work

The PIC stands for the Peripheral Interface Controller which has been coined by Microchip Technologies.PIC Microcontrollers posses an array of features that make them attractive for a wide range of applications. Factors that account for this wide popularity are,

1. Speed
2. Instruction set simplicity
3. Integration of operational features
4. Programmable Timer operations
5. Interrupt control
6. Powerful output pin control
7. Serial programming via two pins

When talking about the speed a PIC executes most of its instructions in 0.2 micro seconds that is in other words 5 instructions per micro second (when oscillator frequency is 20 MHz) . More over the instruction set consists of just 35 instructions. Power on Reset and Brown out Protection ensures the chip operates when the supply voltage is at specified limits. Watch Dog Timer resets the PIC if the chip ever malfunctions from the normal operation. With respect to the interrupt control 12 independent interrupt sources can control when the CPU will deal with each source. A single instruction can select and drive a single output pin high or low in its execution time. The pin can drive a load of up to 25 milli amps.

6.2.3 PIC-Architectural Outline

Among the various types of PIC microcontrollers available in the market the one which we have chosen to employ in our project is PIC16F877, where PIC stands for Peripheral Interfaced Controllers, 16 stands for the mid-range series and the letter 'F' goes for Flash memory. This family of microcontroller uses what is called Harvard architecture to achieve an exponentially fast execution speed for given clock rate. This family belongs to 8 bit microcontrollers. The PIC16F877 device comes in 40 pin packages consisting of 5 ports namely port A, port B,...up to port E. The analog to digital converter module has 8 channels.
Regarding the memory organization there are 3 memory blocks. These are ,

1. Program Memory.
2. Data Memory.
3. EEPROM Data Memory.

The program memory and the data memory has separate buses so that concurrent access can occur. The device has a 13 bit program counter capable of addressing an 8K * 14 Flash program memory space. The reset vector is at 0000H and the interrupt is at 0004H.
Moving on to the data memory, it is portioned into multiple banks which contain the General Purpose Registers and the Special Function registers. The data memory consists of 4 banks where the lower locations of each bank are reserved for the Special Function Registers. Above the Special Function Registers are the General Purpose Registers, implemented as the

appropriate bits of the STATUS Register.

PIC has a RISC (which means Reduced Instruction Set)Architecture. The Instruction set can be categorized into 3 types namely,

1. Byte Wise Instruction.
2. Bit Wise Instruction
3. Control instruction.

6.3 Process Schedule

The work to be performed for processing the client request is subdivided between the two micro controllers equally.

The actions performed by the micro controller 1 are,

1. Receiving and acknowledging the client request
2. Temperature sensing
3. Voltage sensing

6.3.1 Request Processing

The information of the client that is passed to the server is transferred through serial port communication to the micro controller 1.

The acknowledgements for all requests other than the firing data are one in number. When request is the firing data first the micro controller 1 checks whether the request can be processed or not and only if it can be processed it asks for the firing data, other wise the request is not processed.

The LM335 sensor is used for measuring the system temperature.

The LM135 series are precision, easily-calibrated, integrated circuit temperature sensors. Operating as a 2-terminal zener, the LM335 has a breakdown voltage directly proportional to absolute temperature at +10 mV/°K. With less than 1W dynamic impedance the device operates over a current range of 400 µA to 5 mA with virtually no change in performance.

When calibrated at 25°C the LM335 has typically less than 1°C error over a 100°C temperature range. Unlike other sensors the LM335 has a linear output. Applications for the LM335 include almost any type of temperature sensing over a -55°C to +150°C temperature range. The low impedance and linear output make interfacing to readout or control circuitry especially easy. The LM335 operates over a -55°C to +150°C temperature range while the LM335 operates over a -40°C to +125°C temperature range. The LM335 operates from -40°C to +100°C. LM335 are available packaged in hermetic TO-46 transistor packages while the LM335 is also available in plastic TO-92 packages.

Features:

1. Directly calibrated in °Kelvin
2. 1°C initial accuracy available
3. Operates from 400 µA to 5 mA
4. Less than 1W dynamic impedance
5. Easily calibrated
6. Wide operating temperature range
7. 200°C over range
8. Low cost

The Analog-to-Digital (A/D) Converter module has five inputs for the 28-pin devices and eight for the other devices. The analog input charges a sample and hold capacitor. The output of the sample and hold capacitor is the input into the converter. The converter then generates a digital result of this analog level via successive approximation.

The A/D conversion of the analog input signal results in a corresponding 10-bit digital number. The A/D module has high and low voltage reference input that is software selectable to some combination of VDD, VSS, RA2, or RA3.The A/D converter has a unique feature of being able to operate while the device is in SLEEP mode. To operate in SLEEP, the A/D clock must be derived from the A/D's internal RC oscillator. The A/D module has four registers. These registers are:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register0 (ADCON0)
- A/D Control Register1 (ADCON1)

The ADCON0 register controls the operation of the A/D module. The ADCON1 register configures the functions of the port pins. The port pins can be configured as analog inputs (RA3 can also be the voltage reference), or as digital I/O.

Additional information on using the A/D module can be found in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023).

The ADRESH : ADRESL registers contain the 10-bit result of the A/D conversion. When the A/D conversion is complete, the result is loaded into this A/D result register pair, the GO/DONE bit (ADCON0<2>) is cleared and the A/D interrupt flag bit ADIF is set. The block diagram of the A/D module .

channel must be acquired before the conversion is started. The analog input channels must have their corresponding TRIS bits selected as inputs. After this acquisition time has elapsed, the A/D conversion can be started.

The micro controllers being semi conducting devices will not be capable of with standing the high system voltages say 230 or 240 volts. So a transformer with the corresponding transformation ratio is used in order to reduce the voltage considerably. The transformer is just used for a coarse variation of the system voltage. The fine tuning of the voltage is achieved using a potential divider of calculated rating. A filtering capacitor is used for reducing the noise encountered in the above explained processes. The micro controller 2 solely performs the action of generating the firing pulse and acceptance of the output of the zero crossing detector. The flow chart explains the sequence of actions carried out during the generation of firing pulse between the two micro controllers.

6.3.4 TIMER0 Module

The micro controller 2 incorporates the TIMER0 module for producing a constant delay of 3.3ms in time scale that is 60 degrees in radians, between the subsequent firing pulses.

The Timer0 module timer/counter has the following features:

• 8-bit timer/counter

• Readable and writable

• 8-bit software programmable prescaler

• Internal or external clock select

• Edge select for external clock

Timer mode is selected by clearing bit T0CS (OPTION_REG<5>). In Timer mode, the Timer0 module will increment every instruction cycle (without prescaler). If the TMR0 register is written, the increment is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0 register.
The prescaler is mutually exclusively shared between the Timer0 module and the Watchdog Timer. The prescaler is not readable or writable.

Timer0 Interrupt

The TMR0 interrupt is generated when the TMR0 register overflows from FFh to 00h. This overflow sets bit T0IF (INTCON<2>). The interrupt can be masked by clearing bit T0IE (INTCON<5>). Bit T0IF must be cleared in software by the Timer0 module Interrupt Service Routine before re-enabling this interrupt. The TMR0 interrupt cannot awaken the processor from SLEEP, since the timer is shut-off during SLEEP.

Prescaler

There is only one prescaler available, which is mutually exclusively shared between the Timer0 module and the Watchdog Timer. A prescaler assignment for the Timer0 module means that there is no prescaler for the Watchdog Timer, and vice-versa. This prescaler is not
readable or writable .The PSA and PS2:PS0 bits (OPTION_REG<3:0>)

the Timer0 module, all instructions writing to the TMR0 register (e.g. CLRF 1, MOVWF 1, BSF 1,x....etc.) will clear the prescaler. When assigned to WDT, a CLRWDT instruction will clear the prescaler along with the Watchdog Timer. The prescaler is not readable or writable.
Note: Writing to TMR0, when the prescaler is assigned to Timer0, will clear the prescaler count, but will not change the prescaler assignment.

6.3.5 TIMER1 Module

For the calculation of the system frequency micro controller 2 uses the TIMER1 module in timer mode. The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L), which are readable and writable. The TMR1 Register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The TMR1 Interrupt, if enabled, is generated on overflow, which is latched in interrupt flag bit TMR1IF (PIR1<0>). This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit TMR1IE (PIE1<0>).
Timer1 can operate in one of two modes:

- As a timer
- As a counter

The operating mode is determined by the clock select bit, TMR1CS (T1CON<1>). In Timer mode, Timer1 increments every instruction Cycle. In Counter mode, it increments on every rising edge of the external clock input. Timer1 can be enabled/disabled by setting/clearing Control bit TMR1ON (T1CON<0>). Timer1 also has an internal "RESET input". This RESET can be generated by either of the two CCP modules.

RC1/T1OSI/CCP2 and RC0/T1OSO/T1CKI pins become inputs. That is, the TRISC<1:0> value is ignored, and these pins read as '0'.
Manual (DS33023).

Timer1 Operation in Timer Mode

Timer mode is selected by clearing the TMR1CS (T1CON<1>) bit. In this mode, the input clock to the timer is FOSC/4. The synchronize control bit T1SYNC (T1CON<2>) has no effect, since the internal clock is always in sync.

6.3.6 Zero Crossing Detector Module

The Zero crossing detector which is connected to one of the phases performs the job of calculating the occurrence of zero time instance for the remaining two phases with respect to the considered one. This scheme eliminates the usage of zero crossing detector for each phases. By counting the total number of pulses between the occurrences of two signals issued by the zero crossing detector enables to calculate the frequency which is nothing but the inverse of the time calculated.

Firing Circuit

A 555 Timer is used in the astable mode mainly to satisfy the purpose of generating the firing pulses. Because the output of the micro controller is just analogous to an delayed signal and so it can not be directly supplied to the thyristors which require a continuous firing pulses. The output of the 555 Timer can be viewed as follows, With respect to the below an input to a two input AND Gate. The output of the micro controller 2 serves as the second input of the mentioned gate. The other advantage of using the high frequency pulses considerably reduces the gate losses that occur during the time of firing the thyristors.When the two inputs described are given to the AND Gate the output waveform is as follows, This clarifies the equivalent conversion of the firing angle requested by the client into the appropriate time delay produced. Pulse transformers are used for providing isolation between the power circuit and the controlling circuit as their level of power with respect to one another differs.

The output of the controlling circuit that is the one which is isolated from the pulse transformers is provided as the input to the power circuit that is the thyristors employed.

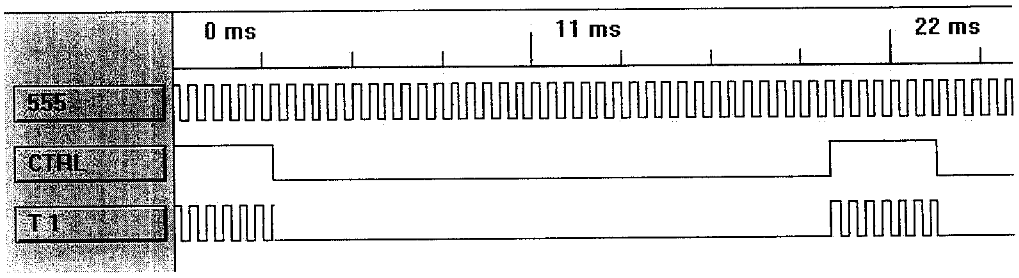


Figure 6.1. Generation of Firing Pulses

MICRO CONTROLLER 1

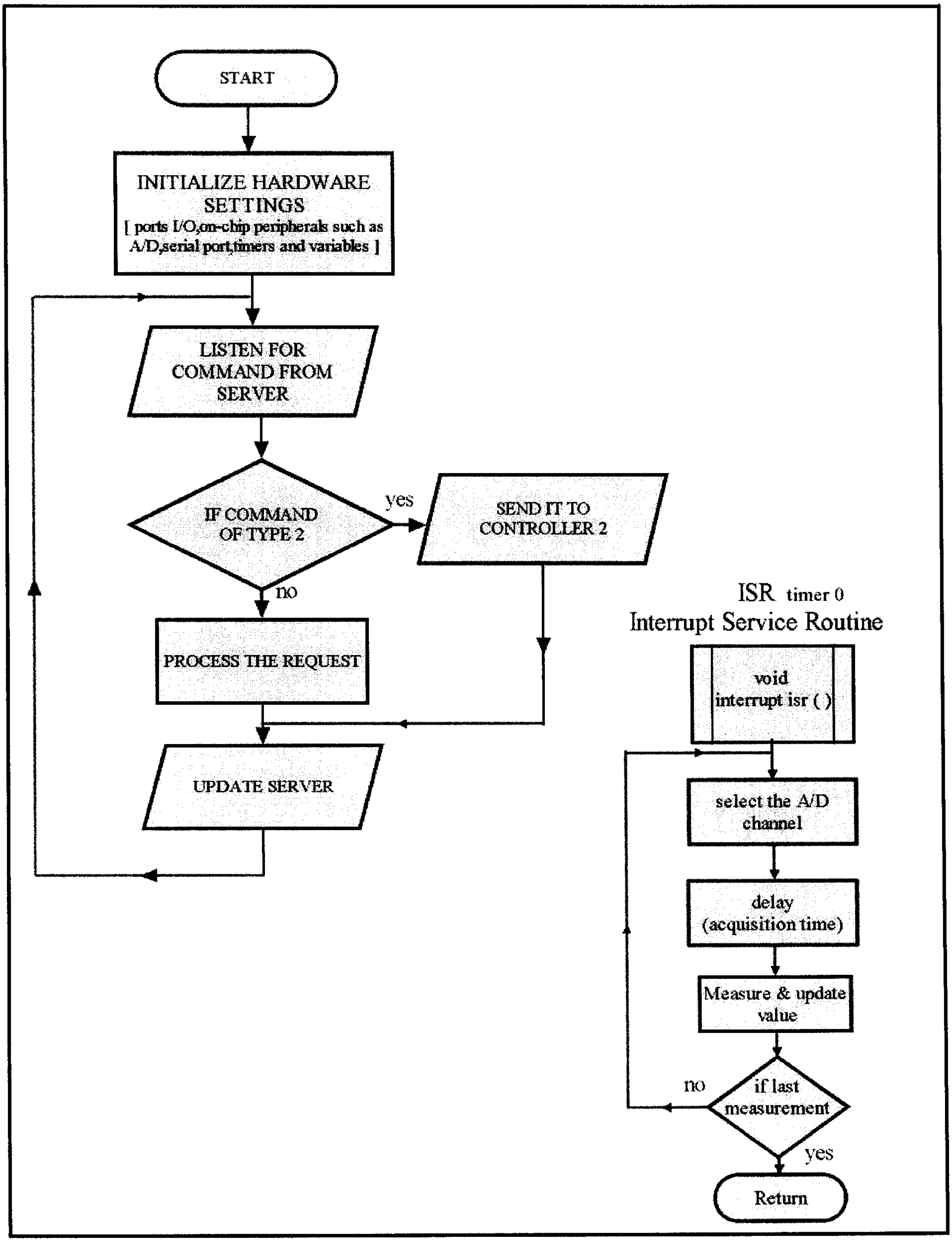


Figure 6.2

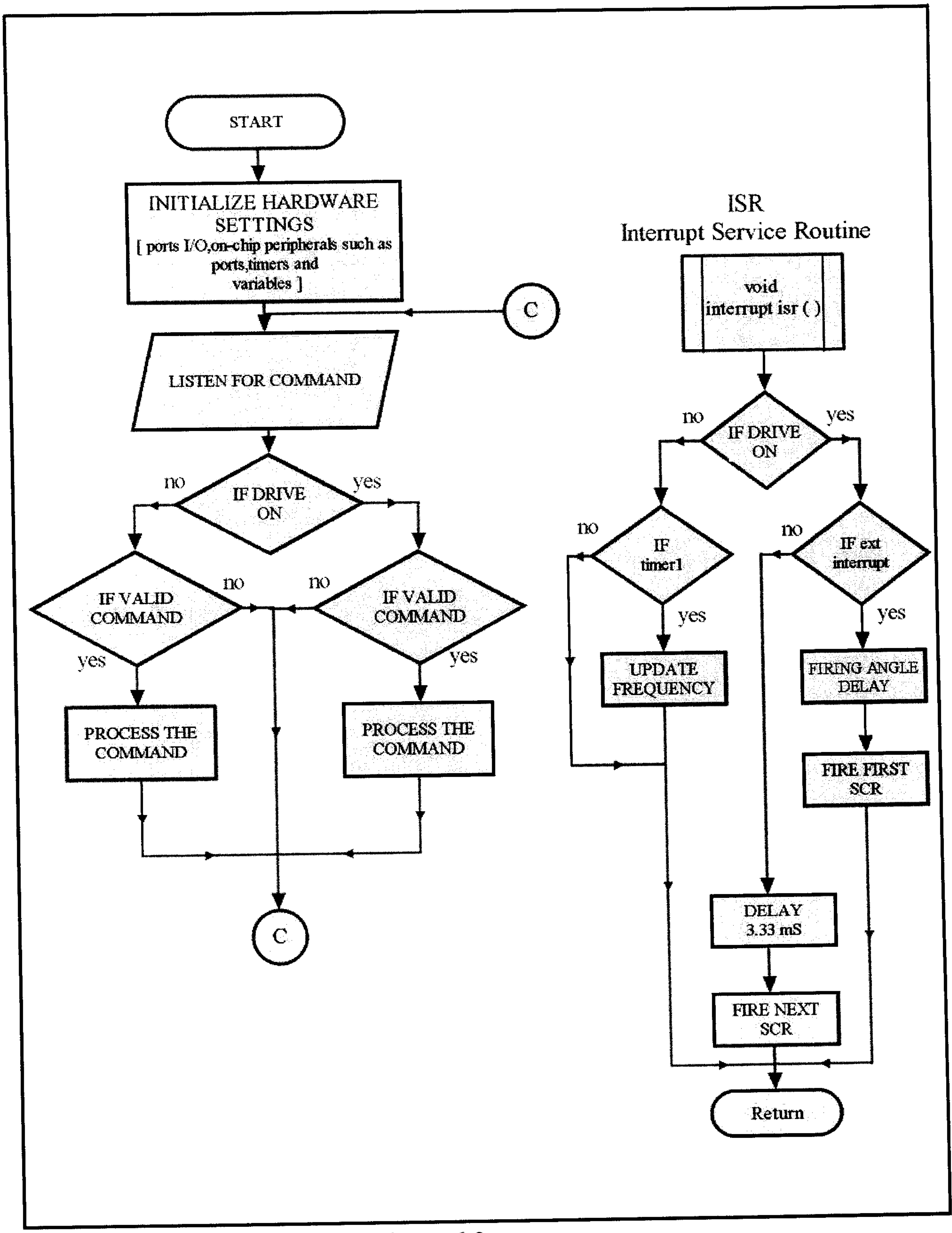

START
INITIALIZE HARDWARE SETTINGS
[ports I/O,on-chip peripherals such as ports,timers and variables]
C
LISTEN FOR COMMAND
IF DRIVE ON
no
yes
IF VALID COMMAND
no
yes
IF VALID COMMAND
no
yes
PROCESS THE COMMAND
PROCESS THE COMMAND
C
ISR
Interrupt Service Routine
void interrupt isr ( )
IF DRIVE ON
no
yes
IF timer1
no
yes
UPDATE FREQUENCY
IF ext interrupt
no
yes
FIRING ANGLE DELAY
FIRE FIRST SCR
DELAY 3.33 mS
FIRE NEXT SCR
Return

Figure 6.3

<u>CONCLUSION</u>

CONCLUSION

"Art is long and Life is short"

7.1 GENERAL FEATURES

The project "e-Control Electric Drives" looks into the arcane world of computer networks and electrical engineering. The working model we have created is a prototype for the future generation to lay their hands on the vast field of networks. The basic theme behind the idea of TCP/IP Protocol in C language is to help the future modifications with light ease. The CGI also uses the C language for program development. The LINUX Operating system is open and so it could be customized by an efficient programmer to his will and wish. The micro controller programming and the hardware circuit is created in such a manner that it could withstand future developments.

7.2 FUTURE ENHANCEMENTS

The project "e-Control of Electric Drives" is developed in such a manner that it could with stand the technological changes that occur in the future. The project we have created is just a prototype model which implements the basics of networking. Home Automation is one of the most exiting field that our project could be enhanced with. Home Automation involves controlling and monitoring the house hold appliances from the remote places by configuring the appliances with the required hardware. Virtual Private Networks which ensure the security between the transmission of data from the involved organization to the roaming computers can be enhanced in our project. The project can also be used in LAN and WAN control of the substations located in remote areas can be efficiently controlled with reduced man power provided the INTERNET Connection is established and the server is configured as in our project. Last but not the least these are only a few future gateways for enhancing our project.

7.3 EPILOUGE

Our Project is an entrance to the field of networks. The entire concept of developing the project lies in the ever-growing field of computer networks which will certainly lead to development of furthermore networking protocols. These protocols and the various encryption techniques available now and will come in the future will certainly help us to enhance our project with much more advanced technologies. The project we have created and tested is just an end of the beginning.

So still there are,

"Miles to go before we sleep
Miles to go before we sleep."

APPENDICES

Client Program Using TCP/IP

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main()
{
        int sockfd,n,r,k,fir;
        char c,temp,state;
        struct sockaddr_in servaddr;
        int power,ss,dir;
        unsigned short int cliarr[5],vv;
        float freq,tt,tempr;
        system("clear");

        printf("\n\n\n\n\t\t\t\t\te-CONTROL OF ELECTRIC DRIVES");

        //Initializing the server socket address structure to 0
        bzero(&servaddr,sizeof(servaddr));

        //creation of the socket
        sockfd=socket(AF_INET,SOCK_STREAM,0);
        if(sockfd==-1)
                printf("\n Error in the creation of the socket");

        //Initializzing the socket to server details
        servaddr.sin_family=AF_INET;
        servaddr.sin_port=htons(9888);

        //set the presentation format
        inet_pton(AF_INET,"192.168.1.1",&servaddr.sin_addr);

        printf("\n\n\nconnecting to the server...................");

        //connecting to the server
        n=connect(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
        printf("\nError in the connection with the
        server\n\t\tconnection failed");
else
        printf("\n\nconnection established");

printf("\n\n\n\t\t Press ENTER to continue.....");
menu:
c=getchar();
if(c=='\n')
{
system("clear");
printf("\n\n\n\t\t\t\t\te-CONTROL OF ELECTRIC DRIVE");
printf("\n\n\n\n\n\n\t1.SEND FIRING ANGLE");
printf("\t\t2.STOP");
printf("\t\t  3.START");
printf("\t4.FORWARD");
printf("\t5.REVERSE");
printf("\t6.GET CURRENT STATUS");
printf("\n\n\t7.UPDATE DRIVE PARAMETERS");
printf("\n\n\t8.CLOSE THE CONNECTION");


printf("\n\n\n\n\n\n\nEnter command:");
scanf("%d",&k);
}

//sending the corresponding details to and fro from the server
write(sockfd,&k,sizeof k);
switch(k)
{
        case 1://firing data
                r=read(sockfd,&temp,sizeof temp);
                printf("\nBytes received from controller: %d",r);
                (temp==1)?printf("\n##### operation success
##### "):printf("\n########## operation failure ############");
                printf("\nEnter the firing angle:");
                scanf("%d",&fir);
                write(sockfd,&fir,sizeof fir);
                r=read(sockfd,&temp,sizeof temp);
                printf("\n Bytes received from controller: %d",r);
new firing angle is set ##### "):printf("\n########### operation failure
############");
                              break;


                  case 2://stop
                              r=read(sockfd,&temp,sizeof temp);
                              printf("\n\nBytes received from controller:
%d",r);
                              (temp==1)?printf("\noperation success ####
drive stopped"):printf("\noperation failure ##### drive is already idle (or)
error in transmission");
                              break;


                  case 3://start
                              r=read(sockfd,&temp,sizeof temp);
                              printf("\n Bytes received from controller %d",r);
                              (temp==1)?printf("\noperation success ####
drive started"):printf("\noperation failure ##### drive is already running
(or) error in transmission");
                              break;

                  case 4://forward
                              r=read(sockfd,&temp,sizeof temp);
                              printf("\n Bytes received from controller %d",r);
                              (temp==1)?printf("\noperation success ####
drive set to forward mode"):printf("\noperation failure ##### drive is
already running ..cannot change direction while running(or) error in
transmission");
                              break;

                  case 5://reverse
                              r=read(sockfd,&temp,sizeof temp);
                              printf("\n Bytes received from controller %d",r);
                              (temp==1)?printf("\noperation success ####
drive set to reverse mode"):printf("\noperation failure ##### drive is
already running ..cannot change direction while running(or) error in
transmission");
                              break;
                  case 6://status
                              r=read(sockfd,&temp,sizeof temp);
```

```
                                state=temp;
                                power=(state&0x04)?1:0;
                                ss=(state&0x02)?1:0;
                                dir=(state&0x01)?1:0;
                                power?printf("\n\n\n##########--Power---OFF-
-###########"):printf("\n\n\n###########--Power---ON--#########");
                                ss?printf("\n############--IDLE---
########"):printf("\n############---RUNNING---########");
                                dir?printf("\n##########--REVERSE--
#########"):printf("\n############--FORWARD--##########");
                                break;
                     case 7:
                                r=read(sockfd,&cliarr,sizeof cliarr);
                                printf("\n The no.of bytes received is %d",r);
                                freq = (1/(cliarr[0]*(0.4)))*(1000000);
                                freq=(freq>52||freq<48)?tt:freq;
                                tt=freq;
                                tempr=cliarr[3]*4.8828/1000.0;
                                tempr=((tempr-2.6500)/0.01)+31.0;
                                vv=cliarr[4];
                                vv=vv>>2;
                                printf("\n\tFREQUENCY     : %2.4f Hz",freq);
                                //printf("\n\tSPEED         : %d rpm",cliarr[1]);
                                printf("\n\tFIRING DATA   : %d
 Degrees",cliarr[2]);
                                printf("\n\tTEMPERATURE   : %2.4f degree
 centigrade",tempr);
                                printf("\n\tVOLTAGE       : %f
 Volts",(0.34633*vv));
                                break;
                     case 8:
                                printf("\nClosing connection....\n");
                                close(sockfd);
                                return 0;
                     default:
                                printf("\n INCORRECT COMMAND...");
                                printf("\nPlease Give correct command");
                                break;
              }
              c=getchar();
              goto menu;
   }
```

Server Program Using TCP/IP

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>

int main()
{
        int sockfd,n,k,fd,tmp,cmd,i,j=0,fire;
        char send[50],buff[10],d,g,serialbuff[20];
        struct sockaddr_in clienaddr,servaddr;
        struct termios options;
        socklen_t clientaddrlen;
        unsigned short int servarr[5];

        //creation of server socket
        sockfd=socket(AF_INET,SOCK_STREAM,0);
        if(sockfd==-1)
        printf("\n Error in the creation of the socket");

        //Initializing the socket address structure
        servaddr.sin_family=AF_INET;
        servaddr.sin_port=htons(9888);
        servaddr.sin_addr.s_addr=INADDR_ANY;

        //Bimd the socket address atructure to a unique name
        bind(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));

        //Listening for the connection from the server
        listen(sockfd,10);

        //Doing the task
        clientaddrlen=sizeof(clienaddr);
        n=accept(sockfd,(struct sockaddr*)&clienaddr,&clientaddrlen);
```

```
    //opening the serial port
    fd=open("/dev/ttyS0",O_RDWR|O_NOCTTY);
    if(fd==-1)
    {
            printf("\n There is an error in opening the port");
    }

    //configuring the serial port with baud rate
    tcgetattr(fd,&options);
    options.c_cflag |= (CLOCAL|CREAD);
    options.c_lflag &= ~(ICANON|ECHO|ISIG);
    cfsetispeed(&options,B19200);
    cfsetospeed(&options,B19200);
    options.c_cflag &= ~PARENB;
    options.c_cflag &= ~CSTOPB;
    options.c_cflag &= ~CSIZE;
    options.c_cflag |= CS8;
    options.c_iflag |= (IXON|IXOFF|IXANY);
    tcsetattr(fd,TCSANOW,&options);

    system("clear");
    printf(" \n\t\t\te-CONTROL OF ELECTRIC DRIVES\n\n\n\n");
    printf("\t\tCLIENT SERVER \t\t\t DRIVE SERVER\n");
    printf("\t\t  COMMUNICATION\t\t\t   COMMUNICATION\n\n\n");

getcmd:

     k=read(n,&cmd,sizeof cmd);
     bzero(serialbuff,sizeof serialbuff);
     switch(cmd)
     {
             case 1:
             printf("\t Command Received   : Firing\n");
             printf("\t                     Angle");
             bzero(serialbuff,sizeof serialbuff);
             d=cmd&0xFF;
             write(fd,&d,1);
             k=read(fd,serialbuff,sizeof serialbuff);
             printf("\t   Acknowledgement Recieved   :%d\n",k);

             g=serialbuff[0];
```

```
read(n,&fire,sizeof fire);
printf("\t Entered Firing Angle  :%d",fire);
d=fire&0xFF;
write(fd,&d,1);
k=read(fd,serialbuff,sizeof serialbuff);
printf("\t   Acknowledgement Recieved  :%d\n",k);
g=serialbuff[0];
write(n,&g,sizeof g);
break;


case 2:
printf("\t Command Received   : Stop");
bzero(serialbuff,sizeof serialbuff);
d=cmd&0xFF;
write(fd,&d,1);
k=read(fd,serialbuff,sizeof serialbuff);
printf("\t   Acknowledgement Recieved   :%d\n",k);
g=serialbuff[0];
write(n,&g,sizeof g);
break;


case 3:
printf("\t Command Received   : Start");
bzero(serialbuff,sizeof serialbuff);
d=cmd&0xFF;
write(fd,&d,1);
k=read(fd,serialbuff,sizeof serialbuff);
printf("\t   Acknowledgement Recieved   :%d\n",k);
g=serialbuff[0];
write(n,&g,sizeof g);
break;


case 4:
printf("\t Command Received   : Forward");
bzero(serialbuff,sizeof serialbuff);
d=cmd&0xFF;
write(fd,&d,1);
k=read(fd,serialbuff,sizeof serialbuff);
printf("\t   Acknowledgement Recieved   :%d\n",k);
```

```
write(n,&g,sizeof g);
break;


case 5:
printf("\t Command Received   : Reverse");
bzero(serialbuff,sizeof serialbuff);
d=cmd&0xFF;
write(fd,&d,1);
k=read(fd,serialbuff,sizeof serialbuff);
printf("\t   Acknowledgement Recieved   :%d\n",k);
g=serialbuff[0];
write(n,&g,sizeof g);
break;


case 6:
printf("\t Command Received   : Status");
bzero(serialbuff,sizeof serialbuff);
d=cmd&0xFF;
write(fd,&d,1);
k=read(fd,serialbuff,sizeof serialbuff);
printf("\t   No. Of Bytes Recieved   :%d\n",k);
g=serialbuff[0];
write(n,&g,sizeof g);
break;


case 7:
printf("\t Command Received   : Update\n");
printf("\t                     Server");
bzero(serialbuff,sizeof serialbuff);
d=cmd&0xFF;
write(fd,&d,1);
k=read(fd,serialbuff,sizeof serialbuff);
printf("\t   No. Of Bytes Recieved   :%d\n",k);
g=serialbuff[1];
servarr[0]=g<<8;
servarr[0]=servarr[0]+serialbuff[0];
bzero(serialbuff,sizeof serialbuff);
d=8;
 write(fd,&d,1);
```

```
            printf("\t\t\t\t\t   No. Of Bytes Recieved   :%d\n",k);
            servarr[1]=serialbuff[1]<<8;
            servarr[1]=servarr[1]+serialbuff[0];
            bzero(serialbuff,sizeof serialbuff);
            d=9;
            write(fd,&d,1);
            k=read(fd,serialbuff,sizeof serialbuff);
            printf("\t\t\t\t\t   No. Of Bytes Recieved   :%d\n",k);
            servarr[2]=serialbuff[0];
            bzero(serialbuff,sizeof serialbuff);
            d=10;
            write(fd,&d,1);
            k=read(fd,serialbuff,sizeof serialbuff);
            printf("\t\t\t\t\t   No. Of Bytes Recieved   :%d\n",k);
            servarr[3]=serialbuff[2]<<8;
            servarr[3]=servarr[3]+serialbuff[1];
            bzero(serialbuff,sizeof serialbuff);
            d=11;
            write(fd,&d,1);
            k=read(fd,serialbuff,sizeof serialbuff);
            printf("\t\t\t\t\t   No. Of Bytes Recieved   :%d\n\n\n",k);
            servarr[4]=serialbuff[1]<<8;
            servarr[4]=servarr[4]+serialbuff[0];
            write(n,servarr,sizeof servarr);
            break;



            case 8:
            printf("\t\t Command Received   : Terminate Connection");
            printf("\nClient requested to close the connection.\nClosing
the connection...\n");
            close(fd);
            close(sockfd);
            close(n);
            return 0;
        }
        goto getcmd;
}
```

Server Configuration Through CGI

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>


//Motor Query and set command codes
#define FIRING          1
#define STOP            2
#define START           3
#define FORWARD                 4
#define REVERSE         5
#define STATUS          6
#define FREQ            7       // Values of update server
#define SPEED           8       //      "
#define FIRE_DATA       9       //      "
#define TEMPERATURE             10      //      "
#define VOLTAGE                 11      //      "

//Bits allocation for Status result(command 6)
#define POWER           4
#define MSTATUS                 2
#define      MDIR               1

#define SERIAL_PORT  "/dev/ttyS0"

#define REV      0
#define FWD      1

#define OFF0
#define ON  1

struct termios options;
int ser_port_fd;
int motor_dir = FWD;
int speed;
int motor_status = ON;
int power_status;
int fire_data;
double freq,voltage,temperature,tt;
int query_motor_status, query_fire_data, query_motor_dir;

int prev_cmd_failed=0;

int initPort( );
void getMotorStatus( );
int parseQueryString(char *qs);
int setValue(char *t1,char *t2);
int setMotorStatus(int ms);
int setFiringAngle(int fa);
int setMotorDir(int md);

#define closePort()close(ser_port_fd)
#define DELAY()          usleep(10000);
#define START_PARA()printf("<P>");
#define END_PARA()    printf("</P>");

int main() {
        int i;
        int query_str_len=0;
        char *query_string=NULL;
        FILE *fp;

        query_string = getenv("QUERY_STRING");
        if( query_string != NULL)
                query_str_len = strlen(query_string);


        printf("Content-Type: text/html\n\n");
        printf("<HTML>");
        printf("<TITLE>e-Control of Motors</TITLE>\n");
        printf("<BODY align>\n");
        printf("<CENTER>");
        printf("<H1>KUMARAGURU COLLEGE OF
  TECHNOLOGY</H1><BR>\n");
```

```
ENGINEERING</H2><BR>\n");
        printf("<H3>e-Control of Electric Drives</H3>\n");
//      printf("%s<BR>\n",query_string);
        if( initPort() == -1 ) {
                printf("Sorry could not open serial port...:(<BR>");
                exit(EXIT_FAILURE);
        }

        getMotorStatus(); //Read the status of motor

        parseQueryString(query_string); //Parse the query string provided
 by CGI

        if( query_motor_status != motor_status && query_motor_status !=
 0xFF)
                setMotorStatus(query_motor_status);

        if( query_fire_data != fire_data && !prev_cmd_failed &&
 query_fire_data != -1)
                setFiringAngle(query_fire_data);

        if( query_motor_dir != motor_dir && !prev_cmd_failed &&
 query_motor_dir != 0xFF)
                setMotorDir(query_motor_dir);

        getMotorStatus();

        printf("<form action=\"http://server.motorcontrol.net/cgi-
 bin/motor.cgi\">");

        START_PARA();
        printf("<b>Firing data: </b>");
        printf("<input type=text name=FireData size=10 maxlength=10
 value=\"%d\">",fire_data);
        printf(" (Please enter the firing angle in degrees)<BR>");
        END_PARA();

        START_PARA();
        printf("<BR><b>Motor On/Off status</b><BR><BR>");
        printf("<input type=radio %s name=On_Off value=\"1\"> On",
 (motor_status==ON)?"checked":"");
```

```
(motor_status==OFF)?"checked":"");
        END_PARA();

        START_PARA();
        printf("<b>Motor direction</b><BR><BR>");
        printf("<input type=radio %s name=Dir value=\"F\">
Forward",(motor_dir==FWD)?"checked":"");
        printf("<input type=radio %s name=Dir value=\"R\">
Reverse",(motor_dir==REV)?"checked":"");
        END_PARA();

        START_PARA();
        printf("-----------------------[ MOTOR STATUS ]---------------------
--<BR><BR>");
        printf("Power           :
%s<BR>",(power_status==ON)?"ON":"OFF");
        printf("Motor run status :
%s<BR>",(motor_status==OFF)?"IDLE":"RUNNING");
        printf("Run direction    :
%s<BR><BR>",(motor_status==OFF)?"NA":((motor_dir==FWD)?"For
ward":"Reverse"));

        printf("Input voltage frequency : %lf<BR>",freq);
//      printf("Speed of motor          : %d<BR>",speed);
        printf("Firing angle            : %d<BR>",fire_data);
        printf("Temperature             : %lf<BR>",temperature);
        printf("Input voltage           : %lf<BR>",voltage);
        printf("<BR><BR><input type=submit>");
        printf("</CENTER>");
        printf("</form");
        printf("</BODY>\n");
        printf("</HTML>");

        closePort(); //close serial port connection
  }

  int initPort()
  {
        //Open the serial port
        ser_port_fd = open(SERIAL_PORT,O_RDWR|O_NOCTTY);
        if(ser_port_fd == -1)
                return ser_port_fd;
```

```
	//Set the serial port options
	tcgetattr(ser_port_fd,&options);
	options.c_cflag |= (CLOCAL|CREAD);
	options.c_lflag &= ~(ICANON|ECHO|ISIG);
	cfsetispeed(&options,B19200);
	cfsetospeed(&options,B19200);
	options.c_cflag &= ~PARENB;
	options.c_cflag &= ~CSTOPB;
	options.c_cflag &= ~CSIZE;
	options.c_cflag |= CS8;
	options.c_iflag |= (IXON|IXOFF|IXANY);
	tcsetattr(ser_port_fd,TCSANOW,&options);

	return 0;
}

void getMotorStatus()
{
	unsigned char serial_data[10];
	int ret;
	unsigned short int value;
	char option;

	// Read the status using option 6
	option = STATUS;
	write(ser_port_fd,&option,1);
	DELAY();
	ret = read(ser_port_fd,serial_data,sizeof serial_data);
	if( ret == -1 ) {
		printf("Error obtaining values\n");
		exit(EXIT_FAILURE);
	}
	power_status = (serial_data[0] & POWER)?OFF:ON;
	motor_status = (serial_data[0] & MSTATUS)?OFF:ON;
	motor_dir = (serial_data[0] & MDIR)?REV:FWD;

	//Obtain the frequency of input voltage
	option = FREQ;
	write(ser_port_fd,&option,1);
//	DELAY();
	ret = read(ser_port_fd,serial_data,sizeof serial_data);
	value = serial_data[1] * 256 + serial_data[0];
```

```
        freq=(freq>51.0)?tt:freq;
        tt=freq;

        //Obtain the speed of motor
        option = SPEED;
        write(ser_port_fd,&option,1);
//      DELAY();
        ret = read(ser_port_fd,serial_data,sizeof serial_data);
        speed = (serial_data[1]<<8) + serial_data[0];

        //Obtain the firing angle
        option = FIRE_DATA;
        write(ser_port_fd,&option,1);
//      DELAY();
        ret = read(ser_port_fd,serial_data,sizeof serial_data);
        fire_data = serial_data[0];

        //Obtain temperature
        option = TEMPERATURE;
        write(ser_port_fd,&option,1);
//      DELAY();
        ret = read(ser_port_fd,serial_data,sizeof serial_data);
        value = (serial_data[2]<<8) + serial_data[1];
        temperature = (value * 4.8828)/1000.0;
        temperature = ((temperature-2.6500)/0.01) + 31.0;

        //Obtain input voltage
        option = VOLTAGE;
        write(ser_port_fd,&option,1);
//      DELAY();
        ret = read(ser_port_fd,serial_data,sizeof serial_data);
        value = (serial_data[1]<<8) + serial_data[0];
        voltage = (value >> 2) * 0.34633;
}

int parseQueryString(char *qs)
{
        char *s = strdup(qs);
        char *token1,*token2;
        if(strlen(s)==0)
                return -1;
        token1 = strtok(s,"=");
```

```
    if(token1 == NULL || token2 == NULL)
            return -1;
    setValue(token1,token2);

    token1 = strtok(NULL,"=");
    if( !token1 )
            return 1;
    token2 = strtok(NULL,"&");
    setValue(token1,token2);

    token1 = strtok(NULL,"=");
    if( !token1 )
            return 1;
    token2 = strtok(NULL,"&");
    setValue(token1,token2);
    return 1;
}

int setValue(char *t1,char *t2)
{
     if(strcmp(t1,"FireData")==0){
             if( strlen(t2) > 0 )
                     query_fire_data = atol(t2);
             else
                     query_fire_data = -1;
             return 1;
     } else if(strcmp(t1,"On_Off")==0) {
             if(*t2 == '0')
                     query_motor_status = OFF;
             else if (*t2== '1')
                     query_motor_status = ON;
             else
                     query_motor_status = 0xFF;
     } else if(strcmp(t1,"Dir")==0){
             if(*t2 == 'F')
                     query_motor_dir = FWD;
             else if (*t2== 'R')
                     query_motor_dir = REV;
             else
                     query_motor_status = 0xFF;
     }
}
```

```
int setMotorStatus(int ms)
{
        int ack;
        char option = (ms)?START:STOP;
        write(ser_port_fd,&option,1);
        DELAY();
        read(ser_port_fd,&ack,2);
}

int setMotorDir(int md)
{
        int ack;
        char option = (md)?FORWARD:REVERSE;
        write(ser_port_fd,&option,1);
        DELAY();
        read(ser_port_fd,&ack,2);
}

int setFiringAngle(int fa)
{
        int ack;
        char fire_ang = fa & 0xFF;
        char option = FIRING;

        write(ser_port_fd,&option,1);
        DELAY();
        read(ser_port_fd,&ack,2);

        write(ser_port_fd,&fire_ang,1);
        DELAY();
        read(ser_port_fd,&ack,2);
}
```

Micro Controller Serial Communication Using C

```
#include"pic1687x.h"


unsigned char count,tcount,sh,sl,tmprh,tmprl,currh,currl,voltl,volth;
unsigned char com,fdata,state,rpmh,rpml,fl,fh;
unsigned int s;
unsigned char channel;

union storage
{
unsigned int v;
unsigned char lh[2];
}

union storage speed;

void delayu(unsigned char d)
{unsigned char i;
for(i=d;d>0;d--);
}



void main()
{

TRISA=0XFF;
TRISE=0XFF;
TRISD=0X00;
TRISB=0X00;
TRISC=0XC0;

ADCON1=0X82;
ADCON0=0X81;

RCSTA=0X90;
TXSTA=0X24;
```

```
INTCON=0X60;
OPTION=0XE8;
PIE1=0X01;
                //RCIF=0;
TMR1IF=0; //whether to remove interrupt for serial reception
T1CON=0X01;
GIE=1;
count=0;
tcount=0;
fdata=0;

while(1)
{
while(RCIF)
{       com=RCREG;

        switch(com)
        {
                case 0x01:
                        TXREG=0X01;
                        while(!TRMT);
                        while(!RCIF);
                        fdata=RCREG;
                        PORTD=fdata;
                        PORTB=0X01;
                        while(!PORTE);
                        TXREG=PORTE;
                        while(!TRMT);
                        PORTB=0;
                        break;
                case 0x02:
                        PORTB=0X02;
                        while(!PORTE);
                        TXREG=PORTE;
                        while(!TRMT);
                        PORTB=0;
                        break;
                case 0x03:
                        PORTB=0X03;
                        while(!PORTE);
                        TXREG=PORTE;
                        while(!TRMT);
        break;
case 0x04:
        PORTB=0X04;
        while(!PORTE);
        TXREG=PORTE;
        while(!TRMT);
        PORTB=0;
        break;
case 0x05:
        PORTB=0X05;
        while(!PORTE);
        TXREG=PORTE;
        while(!TRMT);
        PORTB=0;
        break;
case 0x07:
        TRISD=0XFF;
        PORTB=0X06;
        while(!PORTE);
        fl=PORTD;
        PORTB=0;
        while(PORTE);
        PORTB=0X06;
        while(!PORTE);
        fh=PORTD;
        PORTB=0;
        while(PORTE);
        TRISD=0X00;
        TXREG=fl;
        while(!TRMT);
        TXREG=fh;
        while(!TRMT);
        break;

case 0x06:
        TRISD=0XFF;
        PORTB=0X07;
        while(!PORTE);
        state=PORTD;
        PORTB=0;
        while(PORTE);
        TRISD=0X00;
```

```
                while(!TRMT);
                break;
        case 0x08:
                s=sh;
                s=s<<8;
                s+=sl;
                speed.v=((65535*tcount)+s)*1.2;
                TXREG=speed.lh[0];
                while(!TRMT);
                TXREG=speed.lh[1];
                while(!TRMT);
                TXREG=sh;
                while(!TRMT);
                TXREG=sl;
                while(!TRMT);
                TXREG=tcount;
                while(!TRMT);
                break;
        case 0x09:
                TXREG=fdata;
                while(!TRMT);
                break;
        case 0x0A:
                TXREG=tmprl;
                while(!TRMT);
                TXREG=tmprh;
                while(!TRMT);
                break;

        case 0x0B:
                TXREG=voltl;
                while(!TRMT);
                TXREG=volth;
                while(!TRMT);
                break;
        default:
                TXREG=0X05;
                while(!TRMT);
    }
}
}
}
```

```
{
        if(TMR1IF==1)
        {TMR1IF=0;
        count+=1;

        if(!T0IE)
        {
        TMR0=0XF5;
        OPTION=0XE8;
        T0IF=0;
        T0IE=1;
        }
        channel=0;
        ADCON0=(channel<<3)+0x81;
        delayu(200);
        ADGO=1;
        while(ADGO);
        tmprh=ADRESH;
        tmprl=ADRESL;
        delayu(22);


        channel=2;
        ADCON0=(channel<<3)+0x81;
        delayu(200);
        delayu(200);
        delayu(200);
        ADGO=1;
        while(ADGO);
        volth=ADRESH;
        voltl=ADRESL;
        }
        if(T0IF==1)
        {
        T0IF=0;
        tcount=count;
        count=0;
        sl=TMR1L;
        sh=TMR1H;
        T0IE=0;
        }
    }
```

Micro Controller Program to generate Firing Pulse Using Assembly Language

```
fdata equ 0x21
fangle equ 0x22
count equ 0x23
ang_count equ 0x24
flag equ 0x25                ;=====
reverse equ 0x26
seq equ 0x27
seq_temp equ 0x28
trig_count equ 0x29
comm_check equ 0x2A
fh equ 0x2B
fl equ 0x2C
state equ 0x2D
flag1 equ 0x70                    ;=====
w_temp equ 0x71
status_temp equ 0x72

#include p16f877.inc
org 0x00
goto main
org 0x04
                                    ;CONTEXT SAVING
        movwf w_temp       ;copy w to temp register
        swapf STATUS,0 ;swap status to be saved into w
        clrf STATUS                  ;bank 0, regardless of current bank,
clears irp,rp1,rp0
        movwf status_temp ;save status to bank zero status_temp register

        btfsc PIR1,00
        goto nopwr
        bcf reverse,02

        btfsc flag1,00
        goto fskip

        btfsc flag1,01
        goto fskip
        movf TMR1H,0
        movwf fh
        movf TMR1L,0
        movwf fl
fskip:clrf TMR1H
        clrf TMR1L
        movlw b'00010001'
        movwf T1CON
        bcf flag1,00

        btfsc INTCON,01            ;set as power ON
        bcf reverse,02

        btfsc reverse,01      ;check if stop then return otherwise proceed
        goto s_eoi
        btfsc INTCON,01            ;check RB0 interrupt
        goto eint
next:
        btfss INTCON,02            ;ISR of timer0
        goto eoi
        bcf INTCON,02
        goto firing

main:                              ;HARDWARE SETTINGS
        bsf STATUS,05                      ;ports i/o declraration
        clrf TRISC                 ;port c as output
        clrf TRISE                 ;port E as output=-=-=-=-=-=--TRISC
 TRISE TRISB 2 alter==-=-=-=
        movlw b'11111111'          ;hkhlkh
        movwf TRISD                        ;port D as input
        movlw b'00000001'          ;port b 6,7 as input
        movwf TRISB                        ;pin 6,7 as output and all other pin
 as input
        movlw 0x06                 ;config all analog pin as digital
        movwf ADCON1
        movlw 0xff
        movwf TRISA
        bsf PIE1,00                ;switch ON timer 1 interrupt

        clrf INTCON                ;GIE,INTE,,TMR0IE is temprly disabled
        movwf OPTION_REG    ;AND also intrrupt edge is selected
rising=1,falling=0
        bcf STATUS,05
        clrf PORTC
        clrf PORTA
        clrf PORTB
        clrf PORTE
        movlw b'00000010'
        movwf reverse       ;set direction forward and default as
stop(stopbit01=1--dirbit00=0)
        movlw 0x00
        movwf seq_temp          ;set default sequence as 0
        movlw 0x00
        movwf fangle        ;set firing angle to default 0 degree

        movlw b'00010001'       ;==\=\=\=\=temporarily switch off timer1
for debugging
        movwf T1CON
        movlw b'00000101'
        movwf flag1
        movlw b'11010000'
        movwf INTCON    ;load finally settings GIE=INTE=1
TMROIE=0,INTF=0,TMROIF=0

comm_state:
        clrf comm_check
        movf PORTA,0    ;command protocol start here
        btfsc STATUS,Z
        goto $-2            ;infinite loop to whether any command
        movwf comm_check        ;load command from porta to
comm_check

        btfss reverse,01    ;check the status of motor whether it is running
or idle
        goto run_1              ; and branch accordingly 2 loops
        goto idle_1
        goto $
eint:bcf INTCON,01
        movf seq_temp,0                         ;update firing settings
seq,angle
        movwf seq
        movwf ang_count

        movf ang_count,1                    ;check if angle is zero or equal to 1
and adjust to 1
        btfsc STATUS,Z              ;/./././././././././././lower limit of firing
angle
        goto dend

l2:     movlw 0x5B                  ;-=-=-=-=-====-=---=-=-=re-adjust count
        movwf count
l1:     decfsz count,1
        goto l1
        decfsz ang_count,1
        goto l2
dend:clrf trig_count          ;clearing trigger count so that first group is
executed at cmp interrupt
        goto firing                 ;start firing
eis:
        clrf TMR0
        bsf STATUS,05
        movlw b'10000101' ;==-=-=-=prescalar for tmr0 is set back here=-
=-=-=
        movwf OPTION_REG
        bcf STATUS,05
        movlw b'11100000'
        movwf INTCON
        goto eoi

s_eoi:
         movlw b'11010000'                ;(ON)
GIE,PEIE,INTE=1,,,,,,(OFF)TMROIE,TMROIF=0
         movwf INTCON
eoi:                                 ;CONTEXT RESTORING
        swapf status_temp,0  ;swap status_temp register into w---(sets bank
to original state)
        movwf STATUS              ;move w into status register
        swapf w_temp,1            ;swap w_temp
        swapf w_temp,0            ;swap w_temp into w
        retfie

firing:
        btfss reverse,00
        goto seqf12
seqf12:
        btfss seq,00
        goto seqf1
        goto seqf2
seqf1:
        btfsc flag1,02
        goto startf1
        movf trig_count,0
        addwf trig_count,0
        addwf PCL,1
        call fg1
        goto eis
        call fg2
        goto eoi
        call fg3
        goto eoi
        call fg4
        goto eoi
        call fg5
        goto eoi
        call fg6
        goto tmis
startf1:bsf PORTC,00
          bsf PORTC,04
          bcf flag1,02
          incf trig_count,1
          goto eis
seqf2:btfsc flag1,02
        goto startf2
        movf trig_count,0
        addwf trig_count,0
        addwf PCL,1
        call fg6
        goto eis
        call fg1
        goto eoi
        call fg2
        goto eoi
        call fg3
        goto eoi
        call fg4
        call fg5
        goto tmis
startf2:bsf PORTC,04
        bsf PORTC,02
        bcf flag1,02
        incf trig_count,1
        goto eis
fg1:    bcf PORTC,02
        bsf PORTC,00
        bcf PORTC,04
        incf trig_count,1
        return
fg2:    bcf PORTC,04
      bsf PORTC,05
        bcf PORTC,00
        incf trig_count,1
        return
fg3:bsf PORTC,01
        bcf PORTC,05
        incf trig_count,1
        return
fg4:bsf PORTC,03
        bcf PORTC,01
        incf trig_count,1
        return
fg5:bsf PORTC,02
        bcf PORTC,03
        incf trig_count,1
        return
fg6:bsf PORTC,04
        bcf PORTC,02
        incf trig_count,1
        return
seqr12:
        btfss seq,00
        goto seqr1
        goto seqr2
seqr1:
        btfsc flag1,02
        goto startr1
        movf trig_count,0
        addwf trig_count,0
        call rg1
        goto eis
        call rg2
        goto eoi
        call rg3
        goto eoi
        call rg4
        goto eoi
        call rg5
        goto eoi
        call rg6
        goto tmis
startr1:bsf PORTC,00
        bsf PORTB,06
        bcf flag1,02
        incf trig_count,1
        goto eis
seqr2:btfsc flag1,02
        goto startr2
        movf trig_count,0
        addwf trig_count,0
        addwf PCL,1
        call rg6
        goto eis
        call rg1
        goto eoi
        call rg2
        goto eoi
        call rg3
        goto eoi
        call rg4
        goto eoi
        call rg5
        goto tmis
startr2:bsf PORTB,06
        bsf PORTC,07
        bcf flag1,02
        incf trig_count,1
        goto eis
rg1: bcf PORTC,07
        bsf PORTC,00
        bcf PORTB,06
        return
rg2: bcf PORTB,06
        bsf PORTB,07
        bcf PORTC,00
        incf trig_count,1
        return
rg3:bsf PORTC,06
        bcf PORTB,07
        incf trig_count,1
        return
rg4:bsf PORTC,03
        bcf PORTC,06
        incf trig_count,1
        return
rg5:bsf PORTC,07
        bcf PORTC,03
        incf trig_count,1
        return
rg6:bsf PORTB,06
        bcf PORTC,07
        incf trig_count,1
        return
tmis:
        movlw b'11010000'
        movwf INTCON   ;switch off tmr0 interrupt
        goto eoi

run_l:
        movlw 0x02                      ;loop when motor is running
        xorwf comm_check,0              ; check if command for stop
        btfsc STATUS,Z
        goto stop_f

        movlw 0x01                      ;check if command for firing data
reception
        xorwf comm_check,0
        btfsc STATUS,Z
        goto fdata_f

        movlw 0x06                      ;check if command for request of
frequency
        xorwf comm_check,0
        goto freq_f

        movlw 0x07                      ;check if command for request of
state
        xorwf comm_check,0
        btfsc STATUS,Z
        goto state_f

        movlw 0x02                      ;command not applicable
        movwf PORTE
        goto ak1


idle_l:                                 ;loop when motor is idle
        movlw 0x01                      ;check if command for fdata
reception
        xorwf comm_check,0
        btfsc STATUS,Z
        goto fdata_f

        movlw 0x06                      ;check if command for request of
frequency
        xorwf comm_check,0
        btfsc STATUS,Z
        goto freq_f

        movlw 0x03                      ;check if command for start
        xorwf comm_check,0
        btfsc STATUS,Z
        goto start_f

        movlw 0x04                      ;check if command for forward
direction
        xorwf comm_check,0
        btfsc STATUS,Z
        goto fwd_f
        movlw 0x05                      ;check if command for reverse
direction
        xorwf comm_check,0
        btfsc STATUS,Z
        goto rvs_f
```

```
state
        xorwf comm_check,0
        btfsc STATUS,Z
        goto state_f

        movlw 0x02
        movwf PORTE
        goto ak1
start_f:
        bcf reverse,01
        goto ack
stop_f:
        bsf reverse,01                                  ;=-=-=-check
        goto ack
fwd_f:
        bcf reverse,00
        goto ack
rvs_f:
        bsf reverse,00
        goto ack
fdata_f
        movf PORTD,0                ;-=-=-=what happens if firing angle = 0
deg for both seq-=-=-=-=-
        movwf fdata                 ;get data from PORT D and store it in
fdata
        sublw 0x3C
        btfss STATUS,C              ;check if <=60 and branch
        goto f_sl                   ;=-=-=goto for smaller-larger
        btfsc STATUS,Z              ;check if =60 and branch
        call eq                             ;if =60 goto eq
        movlw 0x3A                          ;;upper limit of range check value
        subwf fdata,0
        btfsc STATUS,C
        call eq
        movlw 0x00
        movwf seq_temp              ;set sequence as 00
        movf fdata,0
        movwf fangle                ;copy fdata 2 fangle
        goto ack

 eq: movlw 0x3A                     ;;upper limit of range check value
        movwf fdata
```

```
f_sl:movlw 0x3C                    ;subr for angle greater than 60
        subwf fdata,1
        movf fdata,0
        sublw 0x3C              ;similar to less than 60 case
        btfss STATUS,C          ;check if <=60 and branch
        call eq                         ;means smaller-larger (greater then
120 deg) so adjust
        btfsc STATUS,Z          ;check if =60 and branch
        call eq                         ;if =60 goto eq
        movlw 0x3A                      ;upper limit of range check value
        subwf fdata,0
        btfsc STATUS,C
        call eq
        movlw 0x01
        movwf seq_temp          ;set sequence as 01
        movf fdata,0
        movwf fangle            ;copy fdata 2 fangle
        goto ack
freq_f:
        bsf flag1,01
        bsf STATUS,05
        clrf TRISD
        bcf STATUS,05
        movf fl,0               ;testinng changrd
        movwf PORTD
        movlw 0x01
        movwf PORTE
        movf PORTA,0
        btfss STATUS,Z
        goto $-2
        clrf PORTE

        movf PORTA,0
        movwf comm_check
        movlw 0x06                      ;check if same command
        xorwf comm_check,0
        btfss STATUS,Z
        goto $-5
        movf fh,0                       ;testinng changrd
        movwf PORTD
        bcf flag1,01
```

```
state_f:
        movf reverse,0
        movwf state
        bsf STATUS,05
        clrf TRISD
        bcf STATUS,05
        movf state,0
        movwf PORTD
spl_ack:movlw 0x01
        movwf PORTE
        movf PORTA,0
        btfss STATUS,Z
        goto $-2
        bsf STATUS,05
        movlw 0xFF
        movwf TRISD
        bcf STATUS,05
        clrf PORTE
        clrf comm_check
        goto comm_state
ack:
        movlw 0x01
        movwf PORTE
ak1:movf PORTA,0
        btfss STATUS,Z
        goto $-2
        clrf PORTE
        clrf comm_check
        goto comm_state

nopwr:bcf PIR1,00
        bsf flag1,02
        bsf reverse,02
        bsf reverse,01
        clrf PORTC
        bcf PORTB,06
        bcf PORTB,07
        goto eoi
end
```

REFERENCES

REFERENCES

1. Behrouz .A. Forouzan (2000) 'Data Communications and Networking'
2. Douglas .E. Comer (1995) 'Internetworking with TCP/IP',Vol.1,Vol.2 and Vol.3
3. Dubey .G.K 'An Introduction to Electric Drives',Naroasa Publishing House
4. Fredrick .M. Cady 'Microcontrollers and Microcomputers principles of Software and Hardware Engineering',Oxford University Press 2003.
5. John .B. Peatman 'Design with PIC Micro controllers', Pearson Education 2004.
6. Richard Stevens .W 'UNIX Network Programming-Networking APIs : Sockets and XTI'

WEBSITES:

1. www.linux.org
2. www.microchip.com
3. www.national.com
4. www.weballey.com