

**SMART HOME APPLICATION:
Controlling A
Portable Headphone Amplifier's Potentiometer
Through Parallel Port Via Internet**

Perpustakaan SKTM

**DANIEL LEONARD HENRY
WEK000359
WXES 3182**

**Supervisor: En. YAMANI IDNA
Moderator: En ZAIDI RAZAK**

The submission of this thesis is to fulfill the requirements needed for
acquiring the Bachelor Degree in Computer Science.

**Faculty Of Computer Science
And Information Technology
University Of Malaya
2003/2004**

ABSTRACT

Smart Homes are being build now at present, and there are no doubts that they are having direct impacts on our life, even on the lifestyles of people living in these Smart Homes.

The project uses parallel port programming to implement the Smart Home concept. This project comprises of two components, a software and hardware component.

The software component consists of a client program and a server program. Both of these programs are connected remotely via the Internet or through a network environment. The client program allows the end user to control the application. The server program is connected to the hardware component using the parallel cable. The client program and server program will be written using Microsoft Visual C++.

The hardware component acts as a volume control device. Here the hardware will control the volume of an audio amplifier. To do this, the hardware will resemble like a stepped attenuator. Electronic devices such as transistors and relays will be used.

The integration of both software and hardware will show that the end user is able to control a remotely situated parallel port, where the server program is controlling the data pins. By enabling and disabling the data pins on the parallel port, it is possible to control the volume of an audio amplifier by acting as a volume control switch.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank God for guiding me and giving me the patience and energy in completing this project. In the process of doing this project, I faced many challenges that sometimes seem to be difficult at first, but with perseverance and focus, I was able to complete this thesis on time.

I would like to take this opportunity to thank a few people, Mr. Yamani Idna as my supervisor and lecturer who helped me enormously and guide me through the process of doing this project. Without your knowledge and consultation, there is no way this project will work.

I would like to thank also to Mr. Zaidi Razak, my moderator for this project, who had given me constructive advise so that this project will be even more challenging and produce better results.

Also not forgetting my family at home, my parents who always pray for my success in whatever things I do. This will always be the driving force behind me.

Finally, I would like to dedicate this project to my friends: Asiah, Albert, Zufri, PoG, Barai, Tawau, Mat, Amal and Kobis.

TABLE OF CONTENT

CHAPTER ONE: INTRODUCTION	1
1.1 Introduction.....	2
1.2 What is Smart Homes?.....	3
1.3 How Smart Homes Work?.....	4
1.4 Smart Homes Has Protocols?	5
1.5 What Can Be Control?	6
1.6 Objective of Project.....	7
1.7 Scope of Project.....	7
1.8 Project Planning and Development.....	9
CHAPTER TWO: LITERATION RESEARCH	10
2.1 Purposes of Literation Research.....	11
2.2 How Potentiometers Work?.....	11
2.3 How Stepped Attenuators Work?.....	15
2.4 How Parallel Ports Work?	19
2.5 How Amplifiers Work?.....	25
CHAPTER THREE: METHODOLOGY.....	32
3.1 Purposes of Project Methodology	33
3.2 Project Development Life Cycle	33
3.3 Methods Approach	35
3.3.1 First Approach: Direct Connection	37
3.3.2 Second Approach: Using Logic Gate	38
3.3.3 Third Approach: Using Relay	41
3.4 Project Requirements	44
	62

3.4.1 Hardware Specification	44
3.4.2 Software Specification.....	47
CHAPTER FOUR: PROJECT DESIGN	48
4.1 Introduction	49
4.2 Designs For Controlling Stepped Attenuator	50
4.3 Portable Headphone Amplifier Design	54
4.4 Software Design.....	57
5.2.3 Socket Programming For Client And Server Program	73
5.3 Hardware Development	78
5.3.1 Hardware Installation As A Programmer	80
5.4 More on Input/Output	82
CHAPTER SIX: SYSTEM TEST	83
6.1 System Test Introduction	84
6.2 Software Test	84
6.3 Hardware Test	85
6.4 Integration Test	87
CHAPTER SEVEN: CONCLUSION	88
7.1 Discussion	89
7.2 Problem Solving	90
7.3 Strength Of Project	90
7.4 Weakness	91
7.5 Suggestions and Recommendations	91
7.6 Conclusion	92
BIBLIOGRAPHY	93
APPENDIX 1: Source Code	94

CHAPTER FIVE: SYSTEM DEVELOPMENT AND IMPLEMENTATION.....	61
5.1 Introduction.....	62
5.2 Software Development.....	63
5.2.1 Parallel Port Control With Microsoft Visual C++	65
5.2.2 Coding For Parallel Port Communication	68
5.2.3 Socket Programming For Client And Server Program	72
5.3 Hardware Development	78
5.3.1 Hardware Interface As A Potentiometer.....	80
5.4 More on Inpou32.dll.....	82
CHAPTER SIX: SYSTEM TEST	83
6.1 System Test Introduction	84
6.2 Software Test.....	84
6.3 Hardware Test.....	86
6.4 Integration Test.....	87
CHAPTER SEVEN: DISCUSSION	88
7.1 Discussion.....	89
7.2 Problems Faced	89
7.3 Strength Of Project.....	90
7.4 System Constraint.....	91
7.5 Suggestions and Improvements.....	91
7.6 Conclusion	92
BIBLIOGRAPHY.....	93
APPENDIX 1: Server Scripts	

APPENDIX 2: Client Scripts

APPENDIX 3: User Manual

Table 1.2: Good Chat Phase 1	9
Table 1.3: Good Chat Phase 3	9
Figure 2.1: Schematic of a Potentiometer	12
Figure 2.2: Slide Potentiometer	13
Figure 2.3: Rotary Potentiometer	14
Figure 2.4: Schematic of Stepped Attenuator	16
Figure 2.5: Voltage Divider	16
Figure 2.6: Schematic of Stepped Attenuator 2	17
Table 2.1: BPP Mode	20
Table 2.2: Currents Mode	20
Table 2.3: EPP Mode	21
Table 2.4: RCP Mode	22
Table 2.5: LPT Port Address	22
Figure 2.7: D0-D3 Multiplexer	24
Figure 2.8: C0-C3 Multiplexer	24
Figure 2.9: 12-Bit DAC Model	26
Figure 2.10: 12-Bit Stepped Attenuator	26
Figure 2.11: First Approach Diagram	27
Figure 2.12: AND Gate	28
Table 2.1: AND Truth Table	28
Figure 2.13: Second Approach Diagram	29
Figure 2.14: Relay Connection	32

LIST OF TABLES AND FIGURES

Table 1.1: Summary of project	8
Table 1.2: Gantt Chart Phase 1	9
Table 1.3: Gantt Chart Phase 2	9
Figure 2.1: Schematic of a Potentiometer	12
Figure 2.2: Slider Potentiometer	13
Figure 2.3: Rotary Potentiometer	14
Figure 2.4: Schematic of Stepped Attenuator	16
Figure 2.5: Voltage Divider	16
Figure 2.6: Schematic of Stepped Attenuator 2	17
Table 2.1: SPP Mode	20
Table 2.2: Centronics Mode	20
Table 2.3: EPP Mode	21
Table 2.4: ECP Mode	22
Table 2.5: LPT Port Addresses	23
Figure 2.7: DB-25 Male Connector	24
Figure 2.8: OPA2134	31
Figure 3.1: Waterfall Model	34
Figure 3.2: 12 Stepped Attenuator	36
Figure 3.3: First Approach Diagram	37
Figure 3.4: AND Gate	38
Table 3.1: AND Truth Table	38
Figure 3.5: Second Approach Diagram	40
Figure 3.6: Relay Construction	42

Table 3.2: Minimum Requirement for a Computer	44
Table 3.3: List for Building Amp	45
Figure 4.1: Highest Level of Design	49
Figure 4.2: Design For Controlling Stepped Attenuator	51
Figure 4.3: Controlling and Dimming a Light Bulb	53
Figure 4.4: Basic Amplifier Design.....	54
Figure 4.5: Schematic for Power Supply Unit.....	55
Figure 4.6: Complete Design of Headphone Amp	55
Figure 4.7: Administrator Mode	58
Figure 4.8: User Mode.....	59
Figure 4.9: Login Menu.....	59
Figure 4.10: Administrator Functions	60
Figure 4.11: Control Menu.....	60

Figure 5.1: Positions of Pin	64
Figure 5.2: Female Connector	65
Figure 5.3: Error Message	66
Table 5.1: Command for Pins	71
Figure 5.4: Socket Connection Process	72
Figure 5.5: At initializing	74
Figure 5.6: Running on Server Mode	75
Figure 5.7: Running on Client Mode	76
Figure 5.8: Server's slider changes	77
Figure 5.9: One part of the circuit	78
Figure 5.10: 12 Stepped Attenuator	80
Figure 5.11: Inpout32.dll process flow chart	82
Figure 6.1: Parallel Port Monitor	85
Figure 6.2: GXSPort	86

1.1 Introduction

Smart homes are no longer a design concept of the future. They are being built now, and they are having a direct impact on the lifestyle of people living in them.

Smart homes are all about taking advantage of the power of modern technology to make things that we make our homes. In the 21st century, it is not just about having a smart home, it is about having a smart home that is built on a solid foundation of technology and a strong network of communication.

Smart homes have many benefits. Smart homes can help you to save money, to improve your security, to improve your health, and to improve your quality of life. Smart homes can also help you to improve your productivity, to improve your communication, and to improve your overall well-being.

CHAPTER ONE: INTRODUCTION

Smart homes are a new way of living. They are a new way of thinking about your home. They are a new way of using technology to make your home smarter. They are a new way of making your home a better place to live.

Smart homes are a new way of living. They are a new way of thinking about your home. They are a new way of using technology to make your home smarter. They are a new way of making your home a better place to live.

1.1 Introduction

Smart homes are no longer a design concepts of the future. They are being built now, and they are having a direct impact on the lifestyles of people living in them.

Smart homes is all about taking advantage of the present and upcoming gee-whiz things that can make your home a 21st century castle. It is also about the automatic operation or control of equipment, a process, or a system build in the house.

Smart homes has many benefits. Smart homes save energy and help the environment through intelligent control of lighting, heating and cooling. Smart homes can protect your family and possessions from an increasingly violent and crime-ridden society through sophisticated security and surveillance systems. Whole house audio and video systems allow you to enjoy music and video from anywhere in your house. The elderly and disabled can have full control of the home from their fingertips.

In the future smart homes will become more intelligent and be able to respond to the individual actions, the individual pre-guessing their actions and providing the appropriate responses. The future is Smart Homes and it is the present.

1.2 What is Smart Homes?

It has many names: smart homes, automated homes, domotics, home networks, intelligent homes etc. It is a home that has many devices that can be monitored and controlled either by telephone or by a computer connected to the Internet. For example, you're driving home from work and would like your house to be warm when you arrive. You can call your house and tell the thermostat to warm it up while you drive.

The difference between a normal house and a smart house is that a communication infrastructure is installed that allows the various systems and devices in the house to communicate with each other. The modern home contains a variety of systems, such as central heating, fire and security alarms, and devices, such as televisions and lights. All these usually exist in total isolation from each other. In the smart house, these systems and devices are able to pass information and commands between them so that, for example, the security alarm can turn the lights on or off.

Smart homes are not just merely for the rich and famous people who live in it, but it is more towards suiting the lifestyle of people. For example, Smart homes can be used for older people and those with disabilities, providing safe and secure environments. Simple everyday tasks such as opening windows, drawing curtains or even opening doors, might appear to be common but for many individuals these functions are almost impossible due to their impairments. Also, intelligently designed and operated buildings yield dramatic increase in worker productivity and energy cost savings, and administrative savings within work environments.

1.3 How Smart Homes Work?

Computerized controls have become more and more common in our homes. Computers control our washing machines and microwaves, they turn our heating on and off, and they have provided new ways to monitor the safety and security of our homes. The smart home looks at expanding the use of these computers into other parts of the home, creating a single network that can be easily and conveniently controlled. The use of computer controls removes the need to actually flick a switch or turn a knob to make something work and allows elements of the home to be controlled remotely by, or to respond automatically to, the people living in it.

The smart home relies on a number of small computers distributed around the house that are either used to turn devices and appliances on and off or to send and receive information. These computers are linked together using either a dedicated cable or by sending a special signal through the mains electricity cables.

1.4 Smart Homes Has Protocols?

Smart homes has a variety of communications protocols for operation. These protocols are the rules for all the smart devices to be able to communicate with each other and being control.

LonWorks or the American Echelon Corporation's LonWorks protocol was originally developed for commercial buildings and industrial processes. LonWorks is capable of being used over all communications mediums but is most commonly installed as a dedicated bus or power line system. Echelon licenses its technologies to a large number of manufacturers giving a wide product base for home installations.

Konnex is the result of the convergence of a number of previously separate European standards for communication. It consolidates the protocols of the European Installation Bus Association (EIBA), BatiBus Club International (BCI), and the European Home Systems Association (EHSA). Again Konnex is suitable for use over all mediums and by the nature of the merger of previous consortia has a wide manufacturing and product base.

X-10, developed in the US in 1977, is a simple, low cost protocol for power line communication. Compared with LonWorks and Konnex it is limited in terms of scope with a maximum of 256 devices available to be connected to a network. X10 consists of signals sent over the existing electrical wiring in your house allowing compatible devices throughout the home to communicate with each other. Using X10 it is possible to control virtually any electrical device from anywhere in the house and remote locations with no additional wiring.

1.5 What Can Be Control?

The interface between the smart home and the user is important to ensure that the home is easy to understand and operate. The smart home is made up of five basic components, which can be integrated to work together and be controlled from other sources. The five basic components are:

- Lightings and window treatments
- Security and access control
- Voice and data communication
- Environmental control and energy management
- Audio/video entertainment

Lighting And Window Treatments: Touch a button on your home theater system and the lights will dim. Set front door and driveway lights to turn on at about the time you arrive home from work. Rugs and furniture fades as they are exposed to the sun. You can use timers to open and close shades according to your schedule (or the sun's schedule).

Security And Access Control: With an integrated system, arming your alarm on your way out the door means not just setting the traditional alarm, but also closing all the windows, locking the doors, setting the thermostat, and closing the shades. If a window is left open, the system will take notice and shut off the heat register in that room. Or, as motion sensors detect people entering a room, your system could be programmed to automatically turn on lights or music.

Voice And Data Communication: An integrated system lets you watch your baby's room or see who's at the front door from your computer monitor or television screen.

Environmental Control And Energy Management: A bedtime directive could arm the alarm system, turn off the water heater and lights. Your system can also manage your water. For instance, your water heater can be turned on only when needed.

Audio And Video: Automatically shut off the television or hi-fi system when leaving the house, or when there is no one at the entertainment room. Remotely controlling the television and audio level of sound from a monitor.

1.6 Objective of Project

This project of mine will demonstrate the concept of Smart Home application by applying the knowledge of computer science. With the knowledge and understanding that I have in computer science, I will integrate the concept of parallel ports and cabling that is found in every computer to demonstrate Smart Home application.

1.7 Scope of Project

The scope of smart home application is wide. Therefore, for the scope of this project, I will narrow the scope down to only demonstrate and focus on audio application. By using the parallel port that is part of a computer peripheral, it is possible to automate or control an audio device. In this project, I will use the parallel port to control an amplifier remotely or via Internet.

Due to the size and the complication of a real amplifier, I will build a model or a prototype of a portable headphone amplifier as a substitute to the real amplifier. The medium of control would be the parallel cable.

The summary of my project is shown in Table 1.1 below:

Table 1.1: Summary of project

Title	Description
Objective	Smart Home application: Controlling a portable headphone amp's potentiometer through parallel port via Internet
Controlling Device	A client and a server computer in a network. The client computer (parallel port) serves as the controlling end, while the server computer acts as the user interface
Medium for controlling	A parallel cable
End Device	A portable headphone amplifier
Output testing	A headphone

1.8 Project Planning and Development

Here I will include 2 Gantt charts for the first and second phase of this project.

The first phase will be during the period March-May 2003, and the second phase will be June-October 2003.

Table 1.2: Gantt Chart Phase 1











Activities/Month	March 03	April 03	May 03
Title Selection/Discussion with lecturer			
Research/Reading/Consulting			
Final Discussion/Analysis			
Final Proposal and Viva			
Writing Report/Submission			

Table 1.3: Gantt Chart Phase 2

Activities/Month	June 03	July 03	August 03	Sep 03	Oct 03
Software Development					
Device Development					
Testing/Simulating					
Viva/Documentation					
Writing Report/Finalizing					

2.1 Chapter 1: Literature Research

This chapter will explain the relevance of a variety of literature research, analyzing of topics and concepts related to doing this project. All the research are based on references from books, articles, journals and the Internet.

This process of literature research is important, as a wide scope of research must be carried out before determining the boundaries and scope of this project. Also, a thorough research and reading on the components of this project will allow me to have more knowledge on solving the problems of this project.

CHAPTER TWO: LITERATURE RESEARCH

2.1 Chapter 1: Literature Research

There are many ways where only a portion of the output voltage from a signal source is required. If we allowed the full output voltage from a CD player to be divided into the input of an amplifier, the amplifier would play at or near full power at all times. This would become quite annoying after a very short period of time. Also, this would go against the purpose of an amplifier to reduce the overall volume we need to hear only a fraction of the full output voltage from the amplifier. To control the level of the signal we use a potentiometer.

A potentiometer, also known as a pot, is a variable resistor. It is a three-terminal adjustable, variable, electrical resistor. Potentiometers can be used to share a voltage in a circuit or to divide a voltage. Potentiometers are used in a variety of applications, such as in a volume control, a balance control, a tone control, a filter control, a sensor, and in the case of a volume control, it is a device that is used to control the level of the signal. Potentiometers are also used in a variety of applications, such as in a volume control, a balance control, a tone control, a filter control, a sensor, and in the case of a volume control, it is a device that is used to control the level of the signal. Potentiometers are also used in a variety of applications, such as in a volume control, a balance control, a tone control, a filter control, a sensor, and in the case of a volume control, it is a device that is used to control the level of the signal.

2.1 Purposes of Literature Research

This chapter will explain the outcome of a variety of literature researching, analyzing of topics and concept related to doing this project. All the researches are based on references from books, articles, journals and the Internet.

This process of literature research is important, as a wide scope of research must be carried out before determining and narrowing the scope of this project. Also, a thorough research and reading on the components of this project will allow me to have more knowledge on solving the problems of this project.

2.2 How Potentiometers Work?

There are many cases where only a portion of an output voltage from a signal source is required. If we allowed the full output voltage from a CD player to be driven into the input of an amplifier, the amplifier would play at or near full power at all times. This would become quite annoying in a very short period of time. Also, that would go against the purpose of an amplifier. To reduce the overall volume, we need to allow only a fraction of the full signal through to the amplifier. To control the level of the signal, we use a potentiometer.

A potentiometer, also known as a 'pot' is a modified resistor. It is a manually adjustable, variable, electrical resistor. Potentiometers can be used to allow a change in the resistance in a circuit or as a variable voltage divider, and in the case of a volume control. It has a resistance element that is attached to the circuit by three contacts, or terminals. The ends of the resistance element are attached to two input voltage conductors of the circuit, and the third contact, attached to the output of the circuit, is usually a movable terminal that slides across the resistance element, effectively dividing

it into two resistors. Since the position of the movable terminal determines what percentage of the input voltage will actually be applied to the circuit, the potentiometer can be used to vary the magnitude of the voltage; for this reason it is sometimes called a voltage divider. Typical uses of potentiometers are in radio volume controls and television brightness controls.

A potentiometer generally has 3 terminals. 2 of the terminals are connected to the opposite ends of a resistive element. The 3rd terminal (usually, is physically in-between the other 2 terminals) is called the wiper. The wiper is a contact (actually, generally many very small contacts) that slides along the resistive element. The diagram below shows the schematic symbol for a pot.

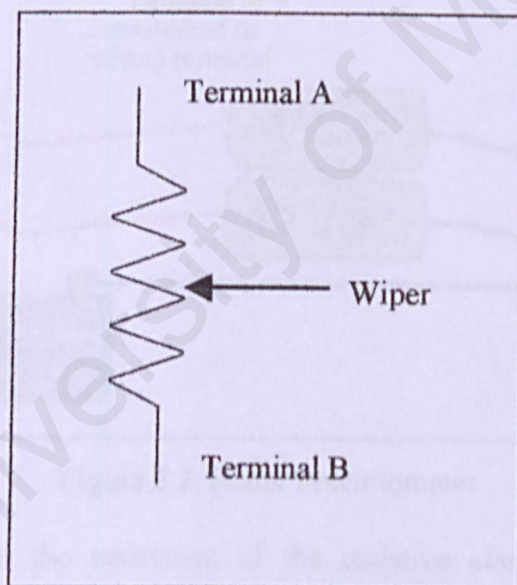


Figure 2.1: Schematic of a Potentiometer

In the diagram below, you can see that the linear taper potentiometer is in the middle of its range of travel. You can also see that 12 volts is applied to terminals 'A' and 'B' are connected to the 12-volt battery. This means that the output from the slider will be 6 volts.

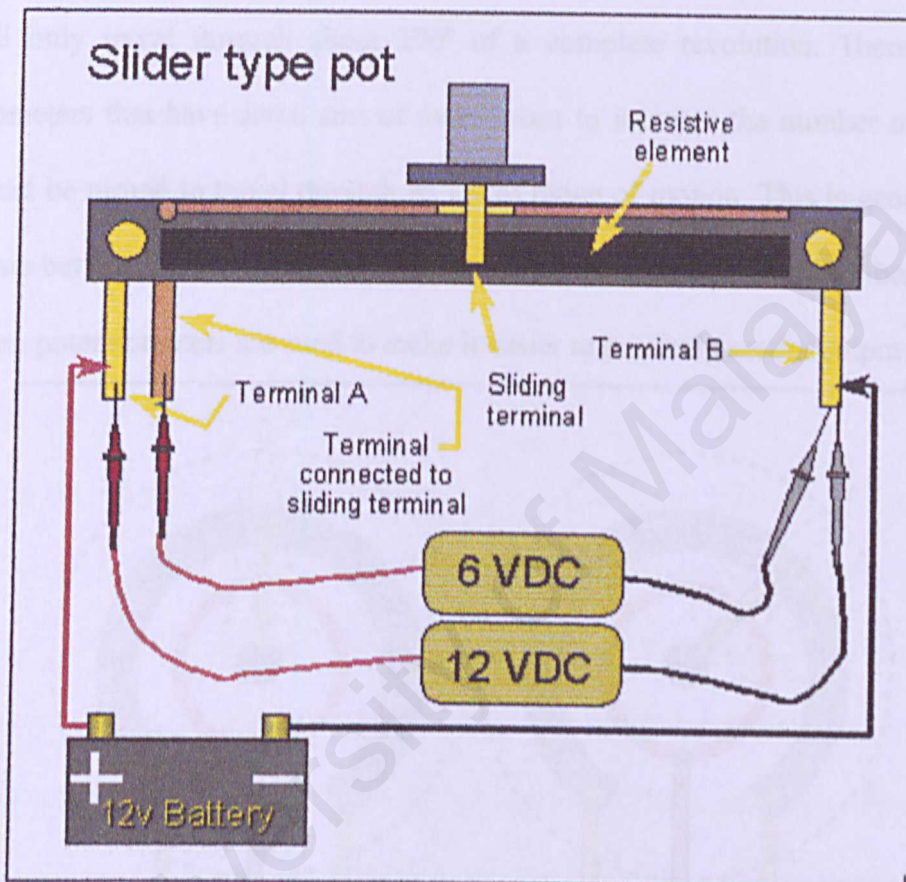


Figure 2.2: Slider Potentiometer

This means that the resistance of the resistive element increases in direct proportion to the distance traveled along the resistive element. In the middle of travel, the resistance from the sliding terminal to either of the other terminals is half of the total resistance. The output is simply the voltage at the point where the wiper contacts the resistive element.

The following diagram shows a rotary potentiometer at 50% and roughly 63% of its range of travel. These positions would correspond to the positions of the slider pot in the previous diagram. This is a single turn potentiometer which means that it can travel its entire range within 1 complete revolution of its shaft (knob). Actually, a single turn pot will only travel through about 270° of a complete revolution. There are other potentiometers that have some sort of mechanism to increase the number of times the shaft must be turned to travel through its entire range of motion. This is generally done with gears but I've seen it done in a planetary gear configuration using ball bearings. The multi-turn potentiometers are used to make it easier to precisely set the output level.

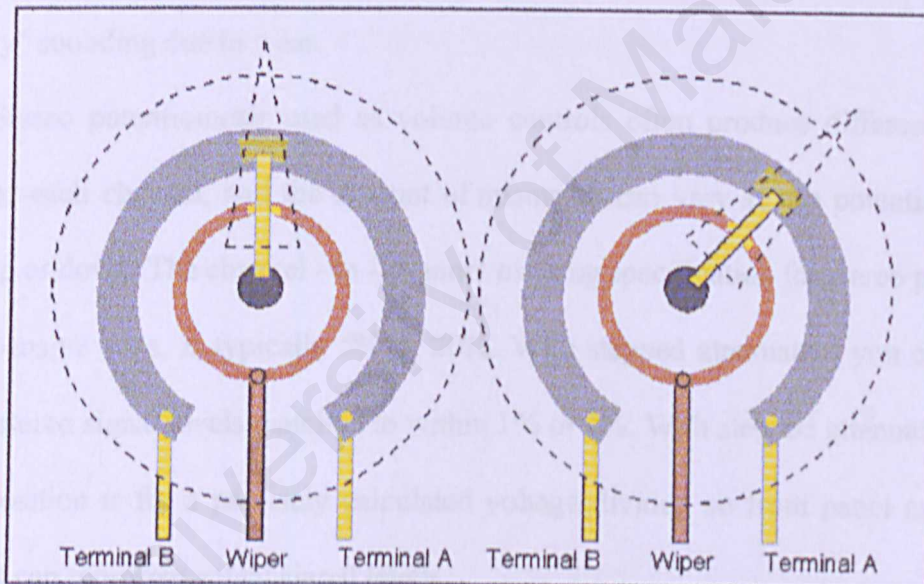


Figure 2.3: Rotary Potentiometer

2.3 How Stepped Attenuators Work?

Stepped attenuators offer distinct sonic advantages over common potentiometers. This is due to their use of discrete resistors and the purest method of signal attenuation that is resistor voltage dividers. Discrete resistors typically have better low noise characteristics than the carbon or cermet resistor elements used in potentiometers.

The total measured resistance for each position of a stepped attenuator remains constant throughout years of use. This is because the wearing (rubbing) parts in stepped attenuators are low resistance switch contacts, not resistive elements with wipers sliding on them as in potentiometers. Potentiometers used as volume controls can become "scratchy" sounding due to wear.

Stereo potentiometer used as volume controls often produce different volume levels for each channel, and the amount of mismatch can vary as the potentiometer is turned up or down. The channel - to - channel tracking specification for stereo pots, even very expensive ones, is typically 5% to 20%. With stepped attenuators you can easily achieve stereo signal levels matched to within 1% or 2%. With stepped attenuators, each switch position is for a precisely calculated voltage divider, so front panel calibration markings can represent actual signal levels.

Figure 2.1: Voltage Divider

Basically, a stepped attenuator works the same as a potentiometer. The only differences are that a stepped attenuator has many precise resistors attached to it. Unlike a potentiometer, it has a resistive element commonly carbon element in it. We can also say that a pot is more to analog and the attenuator is not analog. Each of the resistor's value is different in order to produce the desired attenuation.

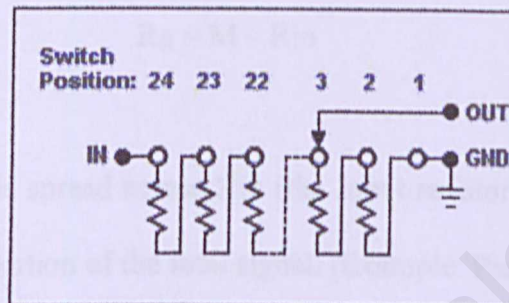


Figure 2.4: Schematic of Stepped Attenuator

Stepped attenuators are essentially simple voltage dividers, and can all be represented by the following circuit:

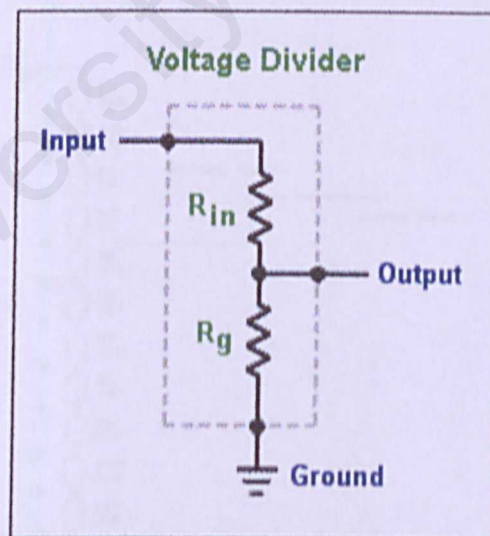


Figure 2.5: Voltage Divider

The amount of attenuation at the output is:

$$[(R_{in} + R_g) / R_g] \times \log \times 20$$

If M is the overall attenuator value or impedance, and D is the amount of attenuation that is desired, the individual resistor values can be calculated.

$$R_{in} = M \times \text{inv.log} \times (D/20)$$

$$R_g = M - R_{in}$$

The total signal (voltage) is spread across **R_{in}** (the Input resistor) and **R_g** (the Ground resistor). The Output is a portion of the total signal. [Example: Per the formula above, if both **R_{in}** and **R_g** are the same value, the Output is -6 decibels down from the total signal level.]

Another way to look at it is the diagram below.

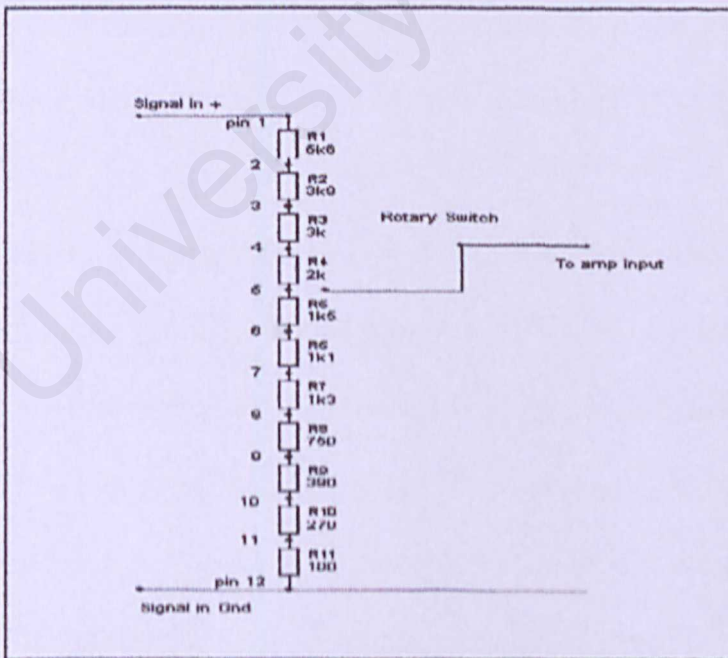


Figure2.6: Schematic of Stepped Attenuator 2

What is most desirable about series type stepped attenuators is that they most closely resemble the operation of a regular variable potentiometer, and so are reliable and will work well almost anywhere. And as the total of $R_{in} + R_g$ (all 23 resistors) is always the same amount, the Input impedance remains constant throughout all 24 switch positions.

2.4 How Parallel Ports Work?

Parallel ports were originally developed by IBM as a way to connect a printer to the computer. When IBM was in the process of designing the personal computer, the company wanted the computer to work with printers offered by Centronics, a top printer manufacturer at the time. IBM decided not to use the same port interface on the computer that Centronics used on the printer.

Instead, IBM engineers coupled a 25-pin connector, DB-25, with a 36-pin Centronics connector to create a special cable to connect the printer to the computer. Other printer manufacturers ended up adopting the Centronics interface, making this strange hybrid cable an unlikely de facto standard.

When a computer sends data to a printer or other device using a parallel port, it sends 8 bits of data or 1 byte at a time. These 8 bits are transmitted parallel to each other, as opposed to the same eight bits being transmitted serially all in a single row through a serial port. The standard parallel port is capable of sending 50 to 100 kilobytes of data per second.

The earliest parallel ports were unidirectional, meaning that data only traveled in one direction for each pin. With the introduction of the PS/2 in 1987, IBM offered a new bi-directional parallel port design. This mode is known as Standard Parallel Port (SPP) and has completely replaced the older design. Bi-directional communication allows each device to receive data as well as transmit it. Many devices use the eight pins (2 through 9) originally designated for data. Using the same eight pins limits communication to only half-duplex communication, meaning that information can only travel in one direction at a time. But pins 18 through 25, originally just used as grounds, can be used

as data pins also. This allows for full-duplex communication or both directions at the same time.

Table 2.1: SPP Mode

<i>Pin</i>	<i>SPP Signal</i>	<i>Pin</i>	<i>SPP Signal</i>	<i>Pin</i>	<i>SPP Signal</i>
1	Strobe	10	Acknowledge	19	Ground
2	Data 0	11	Busy	20	Ground
3	Data 1	12	Paper End	21	Ground
4	Data 2	13	Select	22	Ground
5	Data 3	14	Auto Feed	23	Ground
6	Data 4	15	Error	24	Ground
7	Data 5	16	Initialize	25	Ground
8	Data 6	17	Select In		
9	Data 7	18	Ground		

Table 2.2: Centronics Mode

<i>Pin</i>	<i>Centronics Signal</i>	<i>Pin</i>	<i>Centronics Signal</i>	<i>Pin</i>	<i>Centronics Signal</i>	<i>Pin</i>	<i>Centronics Signal</i>
1	Strobe	10	Acknowledge	19	Ground	28	Ground
2	Data 0	11	Busy	20	Ground	29	Ground
3	Data 1	12	Paper End	21	Ground	30	Ground
4	Data 2	13	Select	22	Ground	31	Initialize
5	Data 3	14	Auto Feed	23	Ground	32	Error
6	Data 4	15	Error	24	Ground	33	Ground
7	Data 5	16	Initialize	25	Ground	34	NC
8	Data 6	17	Select In	26	Ground	35	NC
9	Data 7	18	Ground	27	Ground	36	Select In

Then later, Enhanced Parallel Port (EPP) was created by Intel, Xircom and Zenith in 1991. EPP mode allows for much more data, 500 kilobytes to 2 megabytes, to be transferred each second. It was targeted specifically for non-printer devices that would attach to the parallel port, particularly storage devices that needed the highest possible transfer rate.

Table 2.3: EPP Mode

<i>Pin</i>	<i>EPP Signal</i>	<i>Pin</i>	<i>EPP Signal</i>	<i>Pin</i>	<i>EPP Signal</i>
1	Write	10	Interrupt	19	Ground
2	Data 0	11	Wait	20	Ground
3	Data 1	12	Spare	21	Ground
4	Data 2	13	Spare	22	Ground
5	Data 3	14	Data Strobe	23	Ground
6	Data 4	15	Spare	24	Ground
7	Data 5	16	Reset	25	Ground
8	Data 6	17	Address Strobe		
9	Data 7	18	Ground		

Microsoft and Hewlett Packard jointly announced a specification called Extended Capabilities Port (ECP) in 1992. While EPP was geared toward other devices, ECP was designed to provide improved speed and functionality for printers. In 1994, the IEEE 1284 standard was released. It included the two specifications for parallel port devices, EPP and ECP. In order for them to work, both the operating system and the device must support the required specification. This is seldom a problem today since most computers support SPP, ECP and EPP and will detect which mode needs to be used, depending on

the attached device. If you need to manually select a mode, you can do so through the BIOS on most computers.

Table 2.4: ECP Mode

<i>Pin</i>	<i>ECP Signal</i>	<i>Pin</i>	<i>ECP Signal</i>	<i>Pin</i>	<i>ECP Signal</i>
1	HostCLK	10	PeriphCLK	19	Ground
2	Data 0	11	PeriphAck	20	Ground
3	Data 1	12	NAckReverse	21	Ground
4	Data 2	13	X-Flag	22	Ground
5	Data 3	14	Host Ack	23	Ground
6	Data 4	15	PeriphRequest	24	Ground
7	Data 5	16	nReverseRequest	25	Ground
8	Data 6	17	1284 Active		
9	Data 7	18	Ground		

Each pin from the parallel port is associated with an address. The Parallel Port has three commonly used base addresses. The 3BCh base address was originally introduced used for Parallel Ports on early Video Cards. This address then disappeared for a while, when Parallel Ports were later removed from Video Cards. They now reappeared as an option for Parallel Ports integrated onto motherboards, upon which their configuration can be changed using BIOS.

LPT1 is normally assigned base address 378h, while LPT2 is assigned 278h. However this may not always be the case. 378h & 278h have always been commonly used for Parallel Ports. The lower case h denotes that it is in hexadecimal. These addresses may change from machine to machine.

Table 2.5: LPT Port Addresses

<i>Printer</i>	<i>Data Port</i>	<i>Status</i>	<i>Control</i>
LPT1	0x03bc	0x03bd	0x03be
LPT2	0x0378	0x0379	0x037a
LPT3	0x0278	0x0279	0x027a

When the computer is first turned on, BIOS (Basic Input/Output System) will determine the number of ports you have and assign device labels LPT1, LPT2 & LPT3 to them. The BIOS first looks at address 3BCh. If a Parallel Port is found here, it is assigned as LPT1. Then it searches at location 378h. If a Parallel card is found there, it is assigned the next free device label. This would be LPT2 if a card wasn't found at 3BCh or LPT3 if a card was found at 3BCh. The last port of call, is 278h and follows the same procedure than the other two ports. Therefore it is possible to have a LPT2 that is at 378h and not at the expected address 278h.

What can make this even confusing is that some manufacturers of Parallel Port Cards have jumpers that allow you to set your Port to LPT1, LPT2, LPT3. Now what address is LPT1? - On the majority of cards LPT1 is 378h, and LPT2, 278h, but some will use 3BCh as LPT1, 378h as LPT2 and 278h as LPT3.

The assigned devices LPT1, LPT2 & LPT3 should not be a worry to people wishing to interface devices to their computer. Most of the time the base address is used to interface the port rather than LPT1 etc. However should you want to find the address of LPT1 or any of the Line Printer Devices, you can use a lookup table provided by BIOS. When BIOS assigns addresses to your printer devices, it stores the address at specific locations in memory, so we can find them.

The following shows an example of the male connector and its description.

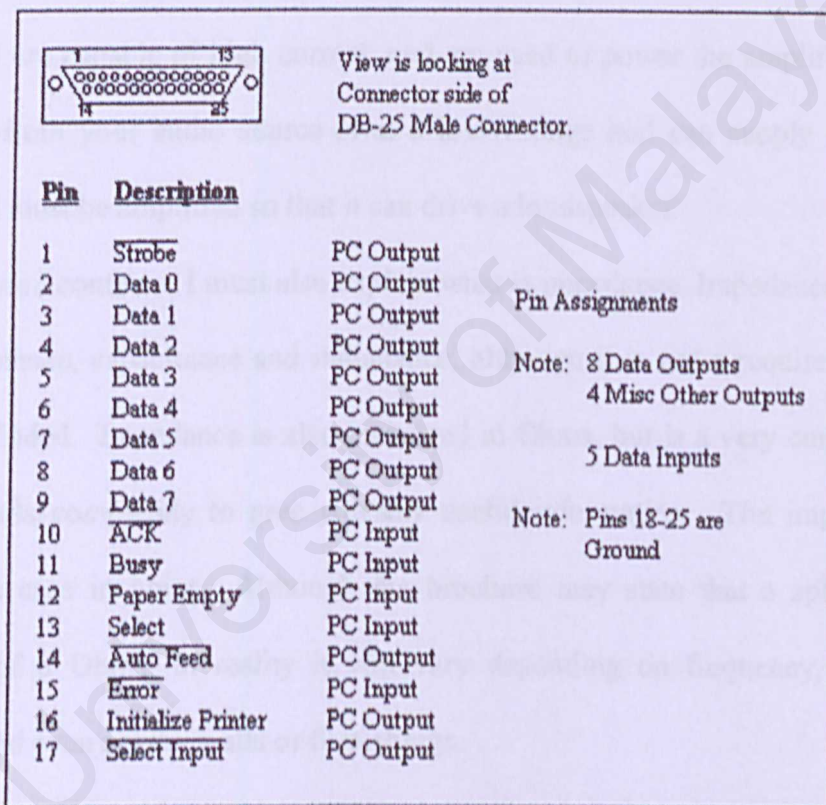


Figure 2.7: DB-25 Male Connector

By enabling either one of the data pins, the enabled pin will produced a voltage while the disabled pins will not have any voltage. With this feature, it is possible to make the parallel port to act as a controlling device.

2.5 How Amplifiers Work?

The term "amplifier" has become generic, and is often thought by some to mean a power amplifier for driving loudspeakers. This is not the case. Well, it is, but it is not the only case.

When it comes to volts and amperes, we have alternating current and direct current, AC and DC respectively. The power from a wall outlet is AC, and so is the output from a CD or cassette player. The mains from the wall outlet are at a high voltage and are capable of high current, and are used to power the amplifying circuits. The signal from your audio source is at a low voltage and can supply only a small current, and must be amplified so that it can drive a loudspeaker.

Before I continue, I must also explain what is impedance. Impedance is a derived unit of resistance, capacitance and inductance, although it is not a requirement that all three be included. Impedance is also measured in Ohms, but is a very complex figure, and often fails completely to give you any useful information. The impedance of a speaker is a case in point. Although the brochure may state that a speaker has an impedance of 8 Ohms, in reality it will vary depending on frequency, the type of enclosure, and even nearby walls or furnishings.

Some audio amplifiers provide several different outputs, each characterized by the impedance of its expected load, for example the impedance of the speaker that you should attach to that output. This impedance measures the relationship between voltage and current that the load needs to function optimally. The higher the impedance, the more voltage the amplifier must provide to propel a particular electric current through the speaker. If the speaker that you attach to the amplifier has the wrong impedance, the

amplifier won't be able to deliver its maximum audio power to the speaker and you may damage the amplifier, speaker, or both.

Since a typical household speaker has an impedance of 8 ohms, you should connect it to an amplifier's 8 ohm output. However, if you connect more than one speaker to the same output, you should be careful to determine the combined impedance. For example, two 8-ohm speakers in series have a combined impedance of 16 ohms while two 8-ohm speakers in parallel have a combined impedance of 4 ohms. Many amplifiers are designed to accommodate these arrangements.

When a distribution amplifier must send current long distances through thin wires, it will often use higher voltages and lower currents to minimize power losses in the wires. Such an amplifier expects its load to have unusually large impedance. In this situation, the speaker that is used must either have large impedance, so that it can use this high voltage/low current power directly, or there must be an impedance matching transformer between the amplifier and the speaker.

The term "amplify" basically means to make stronger. The strength of a signal in terms of voltage is referred to as amplitude. To understand how any amplifier works, you need to understand the following.

- Voltage Amplifier – an amp that boosts the voltage of an input signal
- Current Amplifier – an amp that boosts the current of a signal
- Power Amplifier – the combination of the above two amplifiers

In the case of a voltage amplifier, a small input voltage will be increased, so that for example a 20mV or 0.02V input signal could be amplified to the output of 2 Volt. This represents a gain of 100. The output voltage is 100 times as great as the input voltage. This is called the voltage gain of the amplifier.

In the case of a current amplifier, an input current of 20mA or 0.02A could be amplified to give an output of 2A. Again, this is a gain of 100, and is the current gain of the amplifier.

If now lets combine the two amplifiers, then calculate the input power and the output power, we will measure the power gain:

$$P = V \times I, \text{ where } I \text{ is current.}$$

The input and output power can be calculated:

$$P_{in} = 0.02V \times 0.02A$$

$$= 400\mu W$$

$$P_{out} = 2V \times 2A$$

$$= 4W$$

The power gain is therefore 10,000, which is the voltage gain multiplied by the current gain. Somewhat surprisingly perhaps, we are not interested in power gain with audio amplifiers. There are good reasons for this. Having said this, in reality all amplifiers are power amplifiers, since a voltage cannot exist without power unless the impedance is infinite. This is never achieved, so some power is always present, but it is convenient to classify amplifiers as above.

Amplifiers will be quoted as having specific input impedance. This only tells us the sort of load it will place on preceding equipment, such as a preamplifier. The load is that resistance or impedance placed on the output of an amplifier. In the case of a power amplifier, the load is most commonly a loudspeaker. Any load will require that the source or the preceding amplifier is capable of providing it with sufficient voltage and current to be able to perform its task. In the case of a speaker, the power amplifier must be capable of providing a voltage and current sufficient to cause the speaker cones to move. This movement is converted to sound by the speaker.

Even though an amplifier might be able to make the voltage great enough to drive a speaker cone, it will be unable to do so if it cannot provide enough current. This has nothing to do with its output impedance. An amplifier can have very low output impedance, but only be capable of a small current, for example an operational amplifier, or opamp is a case in point.

The output impedance of an amplifier is a measure of the impedance or resistance looking back into the amplifier. It has nothing to do with the actual loading that may be placed at the output. For example, an amplifier has an output impedance of 10 Ohms. This is verified by placing a load of 10 Ohms across the output, and the voltage can be seen to decrease by 1/2. However, unless this amplifier is capable of substantial output current, we might have to make this measurement at a very low output voltage indeed, or the amplifier will be unable to drive the load. Another amplifier might have an output impedance of 100 Ohms, but be capable of driving 10A into the load. Impedance and current are completely separate, and must not be seen to be in any way equivalent.

Feedback in its broadest sense means that a certain amount of the output signal is fed back into the input. An amplifier or an element of an amplifying device is presented with the input signal, and compares it to a small-scale replica of the output signal. If there is any difference, the amp corrects this, and ideally ensures that the output is an exact replica of the input, but with greater amplitude. Feedback may be as a voltage or current, and has a similar effect in either case.

In many designs, one part of the complete amplifier circuit usually the input stage acts as an error amplifier, and supplies exactly the right amount of signal to the rest of the amp to ensure that there is no difference between the input and output signals, other than amplitude. This is of course an ideal state, and is never achieved in practice. There will always be some difference, however slight.

Some of the electronic formulae are essential in understanding the concept of amplification. The first of these is Ohm's Law, which states that a voltage of 1 V across a resistance of 1 Ohm will cause a current of 1 Amp to flow. The formula is as follow:

$$R = V / I \quad , R \text{ is resistance, } V \text{ is voltage, } I \text{ is current.}$$

Then there is impedance or reactance of a capacitor, which varies inversely with frequency as frequency is increased, the reactance falls and vice versa.

$$X_c = 1 / (2 * \pi * f * C) \quad , \text{ where } X_c \text{ is capacitive reactance, } f \text{ is frequency, } C \text{ is capacitance.}$$

Inductance reactance, being the reactance of an inductor is proportional to frequency.

$$X_i = 2 * \pi * f * L \quad , \text{ where } X_i \text{ is inductive reactance, } L \text{ is inductance.}$$

No discussion of amplification would be complete without a discussion of opamps. Although not a single device, the opamp is considered to be a building block, just like a valve or any transistor.

The operational amplifier or opamp was originally used for analogue computers, although at that time they were made using discrete components. Modern opamps are so good, that it is difficult or impossible to achieve results even close with discrete transistors or FETs. However, there are still some instances where opamps are just not suitable, such as when high supply voltages are needed for large voltage swings.

The majority of power amplifiers whether bipolar or MOSFET are in fact discrete opamps, with a +ve input and a -ve input. Unlike the other devices, opamps are primarily designed as voltage amplifiers, and their versatility comes from their input circuitry. Opamps have two inputs, designated as the non-inverting and inverting or simply + and -.

Modern opamps are as close as anyone has ever got to the ideal amplifier. The bandwidth is very wide indeed, with very low distortion, 0.00003% for one of the Burr Brown devices, and low noise. Although it is quite possible to obtain an output impedance of far less than 10 Ohms, the current output is usually limited to about +/- 20mA or so. Supply voltage of most opamps is limited to a maximum of about +/-18V, although there are some that will take more, and others less.

Depending on the opamp used, gains of 100 with a frequency response up to 100kHz are easily achieved, with noise levels being only very marginally worse than a dedicated discrete design using all the noise reducing tricks known. The circuits shown below have frequency response down to DC, with the upper frequency limit determined by device type and gain.

For the purpose of this project, I will explain the portable headphone amplifier. The headphone amp basically works the same as the normal amplifier found at hi-fi system. The only difference is that the headphone amp drives a headphone and not a speaker.

Basically a headphone amp consists of 2 departments, the power supply unit and the amplifier circuit. The power supply unit (PSU) will either be powered by a power adaptor or batteries. The PSU job is to supply the appropriate power or voltage to the amp circuit. This is important because the amp will not function if the voltage supplied is insufficient.

The main component of the amp circuit is the operational amplifier IC chip. The op-amp will act as the heart of the entire amp to amplify the input signal.

Below shows a diagram for the Burr Brown OPA2134 opamp that will be used in this project.

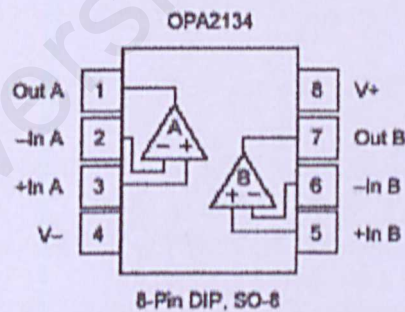


Figure 2.8: OPA2134

OPA2134 is a dual channel opamp. This is an ideal example opamp because we need 2 channels to produce stereo output. The voltage supply range is as low as 2.5V and as high as 18V.

1.1 Purpose of Project Methodology

This chapter will explore the methodology used in this project, as well as explain the reasons for choosing this methodology. Methodology is defined as a collection of procedures, techniques, tools and documentation aids that can help to speed up the development of a project.

Here I will explain some of the approaches that I used in this project. The first and second approaches are not feasible because of various reasons and I will explain why. The 3rd approach is the one that I used in this project and I will explain why. I will also explain why I chose this approach and I will explain why I chose this approach.

CHAPTER THREE: METHODOLOGY

1.2 Project Development Life Cycle

For this project, planning is a key part of the project. The project work is done in a structured way. Therefore, a development life cycle is used to make sure that the process of developing this project is done in a structured way. The life cycle is given as follows:

Here I adopted the Project Development Life Cycle based on software engineering. Although the project is not fully a software project, it is possible to use the Project Development Life Cycle in this project. The project is divided into four phases: Planning, Analysis, Design and Implementation. The project is divided into four phases: Planning, Analysis, Design and Implementation. The project is divided into four phases: Planning, Analysis, Design and Implementation.

In this project, the project is divided into four phases: Planning, Analysis, Design and Implementation. The project is divided into four phases: Planning, Analysis, Design and Implementation. The project is divided into four phases: Planning, Analysis, Design and Implementation. The project is divided into four phases: Planning, Analysis, Design and Implementation.

3.1 Purposes of Project Methodology

This chapter will explain the methodology used in developing this project, as well as solutions to the problem faced while planning for the project. Methodology is defined as a collection of procedures, techniques, tools and documentation aids that can help to speed up the development process smoothly.

Here I will explain some of the approaches earlier decided to do this project. The first and second approaches are not feasible because of certain difficulties and uncertainties found during the earlier planning. The 3rd approach as the one chosen and I personally think is best to do this project.

3.2 Project Development Life Cycle

For this project, planning is as important as making sure the final product work. Therefore, a development life cycle is essential to make sure that the process of developing this project runs smoothly and accordance to the time period given.

Here I adapted the System Development Life Cycle based on software engineering. Although my project is not fully a software project, but it is possible to adapt the development life cycle concept into this project because the processes involved in making this project is just the same as developing software.

In this project, the model chosen to do this project is the Waterfall Model. I've chosen this model because of its simplicity and easy to implement to my project. Also, this model is straightforward and easy to follow.

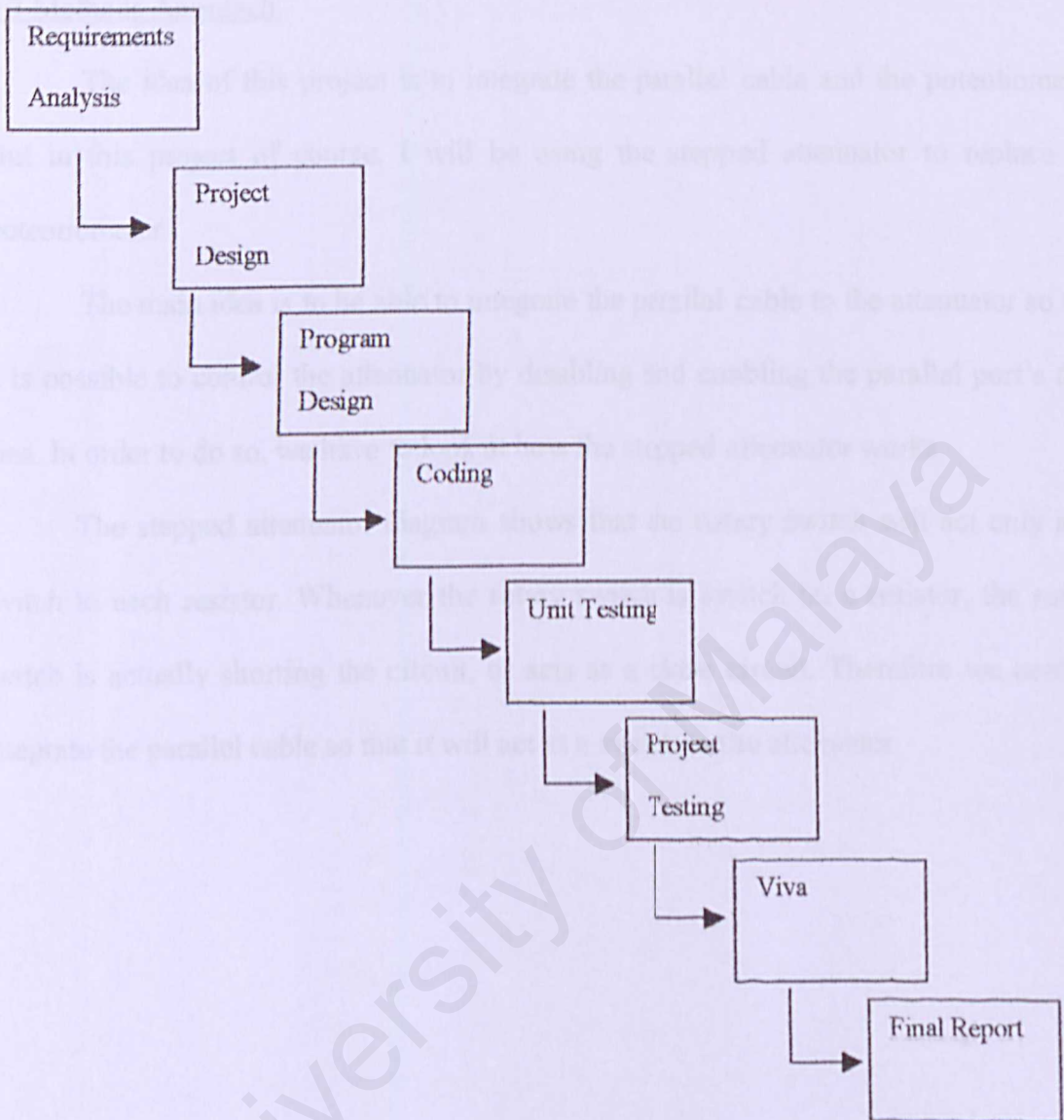


Figure 3.1: Waterfall Model

3.3 Methods Approach

The idea of this project is to integrate the parallel cable and the potentiometer. But in this project of course, I will be using the stepped attenuator to replace the potentiometer.

The main idea is to be able to integrate the parallel cable to the attenuator so that it is possible to control the attenuator by disabling and enabling the parallel port's data pins. In order to do so, we have to look at how the stepped attenuator works.

The stepped attenuator diagram shows that the rotary switch will act only as a switch to each resistor. Whenever the rotary switch is switch on a resistor, the rotary switch is actually shorting the circuit, or acts as a close circuit. Therefore we need to integrate the parallel cable so that it will act as a switch to the attenuator.

Series Type Stepped Attenuator

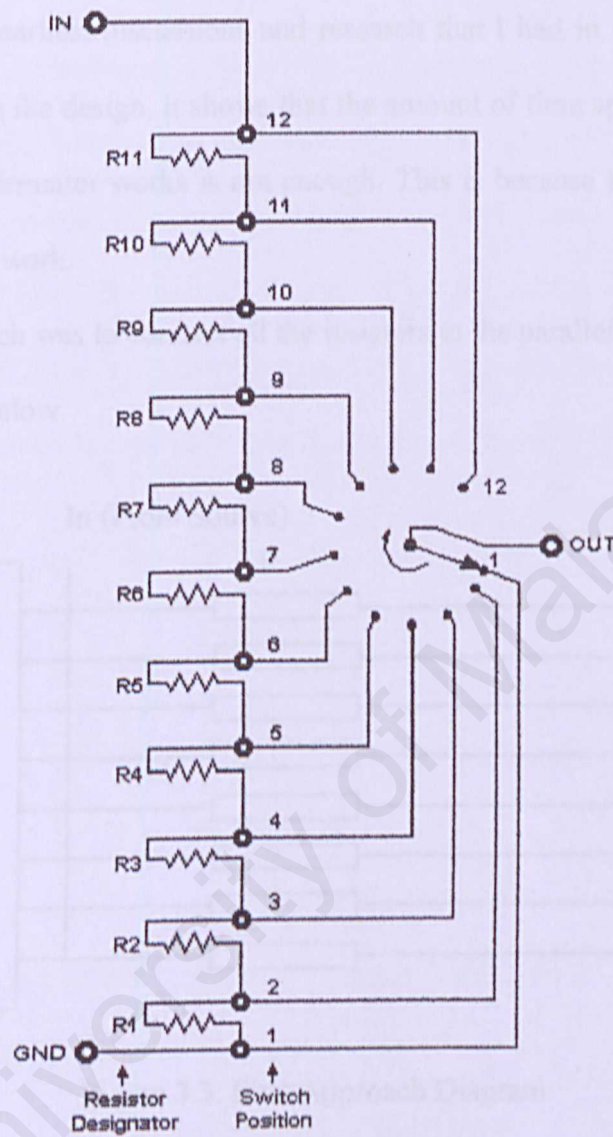


Figure 3.2: 12 Stepped Attenuator

3.3.1 First Approach: Direct Connection

This is the earliest discussions and research that I had in order to come to this approach. Based on the design, it shows that the amount of time spent on understanding the way that the attenuator works is not enough. This is because the first approach is a failure and will not work.

This approach was to connect all the resistors to the parallel cable's data pin. The diagram is shown below.

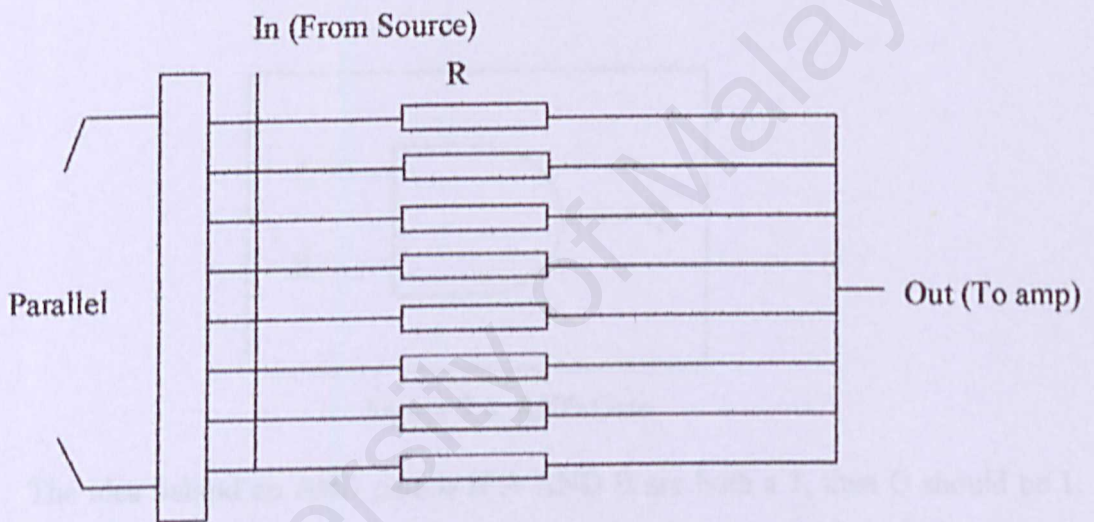


Figure 3.3: First Approach Diagram

This design will not produce an amplified signal at the output. This layout will not work because even though the data pins of the parallel port is disabled, the signal from the source will still be able to flow thru to the amp.

3.3.2 Second Approach: Using Logic Gate

After more researching and seeking consultations from lecturers from other faculty, a second plan was formed. The second plan was almost feasible. This involves the usage of a logic gate IC. The logic gate here is the AND logic.

Logic AND has 2 inputs and produces 1 output, or the results from the AND operation. With this gate, a combination of outputs can be achieved. The AND gate performs a logical “and” operation on two inputs A and B, as shown below:

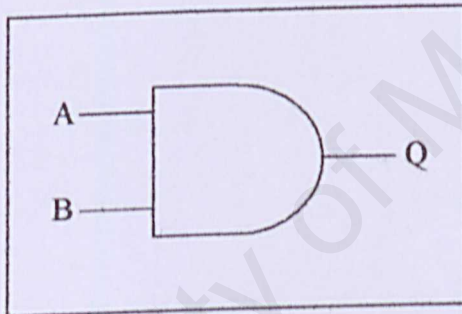


Figure 3.4: AND Gate

The idea behind an AND gate is if A AND B are both a 1, then Q should be 1. This behavior can be shown in the truth table below:

Table 3.1: AND Truth Table

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

The design of the second approach is shown below:

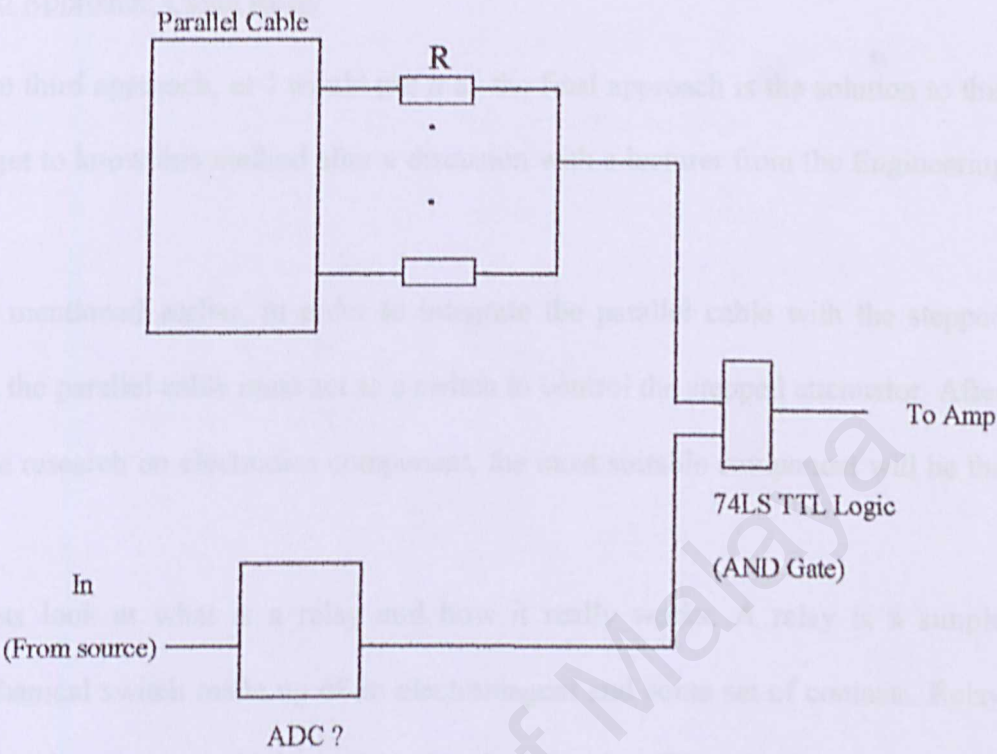


Figure 3.5: Second Approach Diagram

This design too has problem. It seems that the idea behind this design is correct, but can a signal be AND with a voltage or current from the parallel cable? The signal from the source seems to be an analog signal, should use analog to digital converter? This idea seems to produce more questions of uncertainties. Therefore, a third approach is needed in order to find a better solution.

3.3.3 Third Approach: Using Relay

The third approach, or I would put it as the final approach is the solution to this project. I get to know this method after a discussion with a lecturer from the Engineering Faculty.

As mentioned earlier, in order to integrate the parallel cable with the stepped attenuator, the parallel cable must act as a switch to control the stepped attenuator. After doing some research on electronics component, the most suitable component will be the relay.

First let's look at what is a relay and how it really works. A relay is a simple electromechanical switch made up of an electromagnet and some set of contacts. Relays are simple devices and are widely used in electrical devices. There are four components in a relay:

- An electromagnet
- Armature that can be attracted by the electromagnet
- A spring to pull back the armature to its original place
- And a set of electrical contacts

As in the figure below, you can see that a relay consists of two separate and independent circuits. The lower circuit will drive the electromagnet. A switch is controlling the power to the electromagnet. When the switch is turned on, the electromagnet will act as a magnet and will pull the armature towards it. The armature will thus close on the inner circuit in the relay. The armature can be seen as a switch to the second circuit. When this happens, there will be readings in the voltmeter because the

armature completes the upper circuit. When the electromagnet is not energized or the switch is off, the spring attached to the armature will pull the armature away and the inner circuit is not complete. In this case, no reading on the voltmeter.

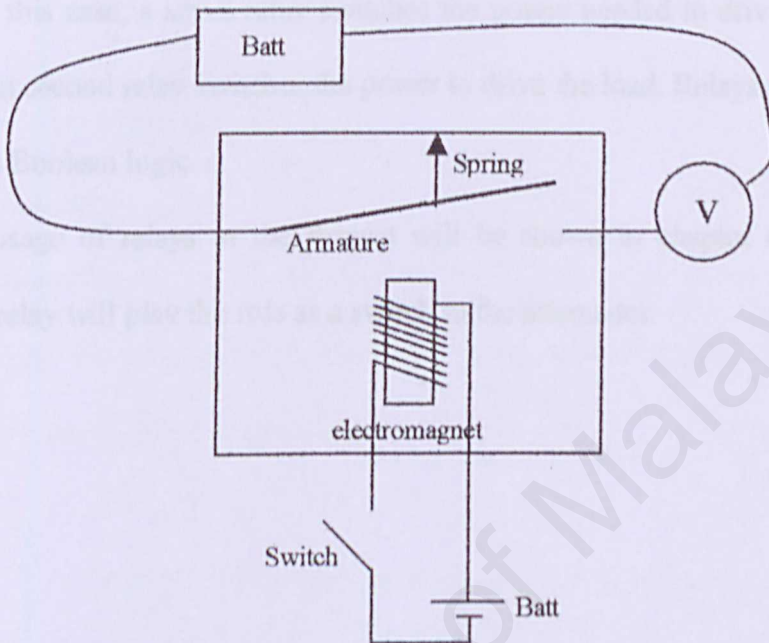


Figure 3.6: Relay Construction

With a relay, it is possible to control several variables. A relay has different voltage and current that is needed to activate the armature. A relay has different maximum voltage and current that can run through the armature and the armature contacts. Relays come in different kinds of shape and sizes, with different numbers of armatures, generally one or two. Also, the numbers of contacts for the armature too are different. Generally there are one or two, in which sometimes one is not used. And relays come in whether the contact is normally open or normally closed.

Relays are quite common in home appliances where there is an electronic control turning on something like a motor or a light. They are also common in cars, where the 12V supply voltage means that just about everything needs a large amount of current. In

later model cars, manufacturers have started combining relay panels into the fuse box to make maintenance easier.

In places where a large amount of power needs to be switched, relays are often cascaded. In this case, a small relay switches the power needed to drive a much larger relay, and that second relay switches the power to drive the load. Relays can also be used to implement Boolean logic.

The usage of relays in the Project will be shown in chapter 4 under Project Design. The relay will play the role as a switch to the attenuator.

Component	Approximate Cost
Processor	\$10.00
Memory	\$5.00
Hard Disk	\$15.00
Network Interface	\$10.00

3.4 Project Requirements

Project requirements include the tools and devices needed to develop this project.

Here I've put the requirements under two categories, which is hardware and software.

3.4.1 Hardware Specification

1. Personal Computer

A personal computer is needed to develop the software for the project. Also the computer will be the simulation platform for testing out the system. The table below shows the minimum requirements for a computer.

Table 3.2: Minimum Requirement for a Computer

<i>Components</i>	<i>Requirements</i>
Processor	Pentium 2 550 MHz
Memory	128 MB
Hard Disk	10 Giga
Network Interface Card	10/100 Mbps

2. Electronic components

In this project, a few of the electronic components will be needed to build the parallel cable interface and also the headphone amp. The components include some resistors, capacitors, stereo socket, operational amplifier, buffer, rail-splitter and so on. The table below shows the list needed to build the amplifier:

Table 3.3: List for Building Amp

RS			
Components	Description	Qty	ID
	Power Supply		
315-0726	220 uF 35V Electrolytic Cap, 105 degC, Panasonic	2	C1/C2
365-4133	220 uF 25V Electrolytic Cap, 105 degC, Panasonic	2	
365-4060	330uF, 16V, 105 degC, Panasonic	2	
148-736	10K ohm 1/4W metal film resistor	1	R1
148-663	Resistor, metal film, 0.25W, 4.7K/4.75K ohm(4K7)	2	R2/R3
588-386	LED, indicator, low current, 3mm dia, red	1	D1
	Amp		
115-023	Cap, min, encapsulated, polyster film, 63Vdc, 0.1uF	2	C1
312-1469	Cap, metallised polyster, 63V, 0.1uF	2	
148-972	Resistor, metal film, 0.25W, 100K	2	R2
148-506	Resistor, metal film, 0.25W, 1K	2	R3
148-736	1/4W metal film resistor, 10K ohm	2	R4
218-8281	IC, Op Amp, Voltage Feedback, dual, OPA2132PA	1	OPA
285-8069	IC, Op Amp, Voltage Feedback, dual, OPA2134PA	1	OPA
813-115	IC, socket, low profile, DIL, turned pin, solder, 8 way	1	
148-174/269	Resistor, metal film, 0.25W, 47R, 1% (47 Ohm)	2	R5
148-269	Resistor, metal film, 0.25W, 100R, 1% (100 Ohm)	2	
	PCB & Stuff		
330-840	Switch, Toggle, ultra min, silver contacts, SPCO, on-on	1	SWI
206-5841	PCB, single sided	1	
449-348	Connector, audio, 0.25in, PCB mount socket 3 way	2	

173-849	Potentiometer, Cermet, 12mm, 1W 10K	1	
	Panasonic EVJ Y10 Series 10K		
173-631	Potentiometer, panel mount, plastic, 10K, 0.5W		
259-6941	Knob, control, collect, plastic, 16.2mm, 1/4in shaft, black cab	1	

This will be the programming language used to write the system software.

2. Pascal

This will be the software used to describe the architecture of the system.

3. Microsoft Windows XP

This will be the operating system platform used to run the system. It will be the main platform.

3.4.2 Software Specification

1. Microsoft Visual Studio 6 / Visual C++ 6.0

This will be the programming language used to code the system software.

2. Pspice

This will be the software used to simulate the schematic of the amplifier.

3. Microsoft Windows XP

This will be the operating system platform used to run the above software. It will be the main platform.

4.1 Introduction

This phase is important. Designing a project will determine the outcome of the project. Producing a good design will ensure that the final product will not only work properly, but the process of developing it will be easy and smooth.

In this chapter, I will include the design for the entire project, as well as the design of the possible hardware modules. Figure shows the highest level of design for the project.

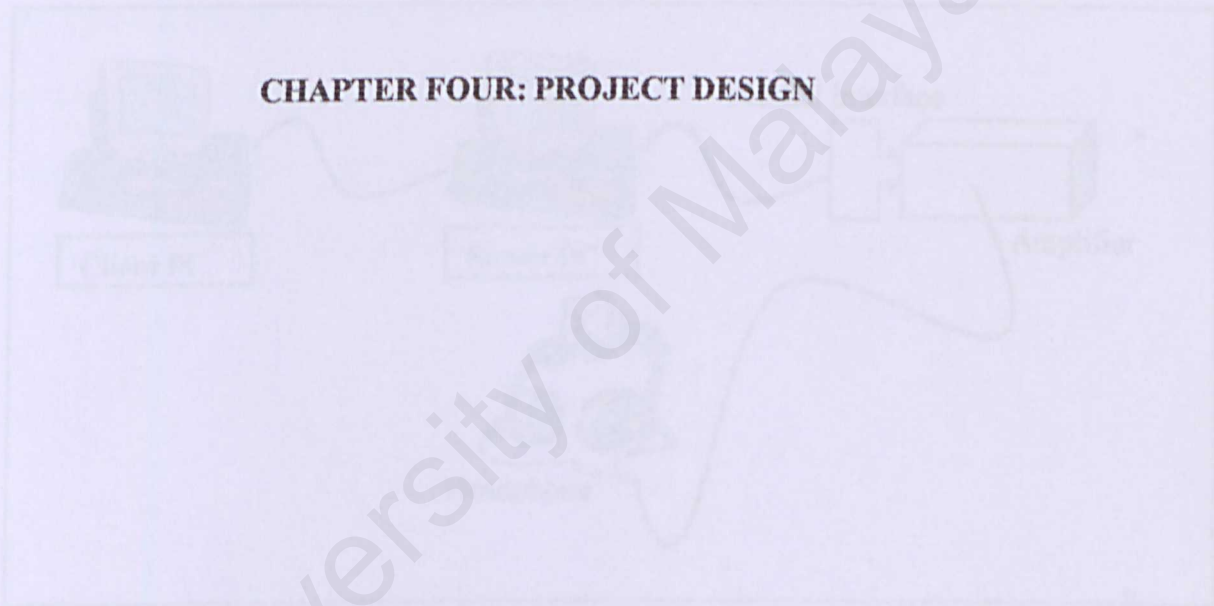


Fig 4.1 Highest Level of Design

4.1 Introduction

This phase is important. Designing a project will determine the outcome of the project. Producing a good design will ensure that the final product will not only work properly, but the process of developing it will be easy and smoothly.

In this chapter, I will include the design for the entire project, as well as the design of the portable headphone amplifier. Below shows the highest level of design for the project:

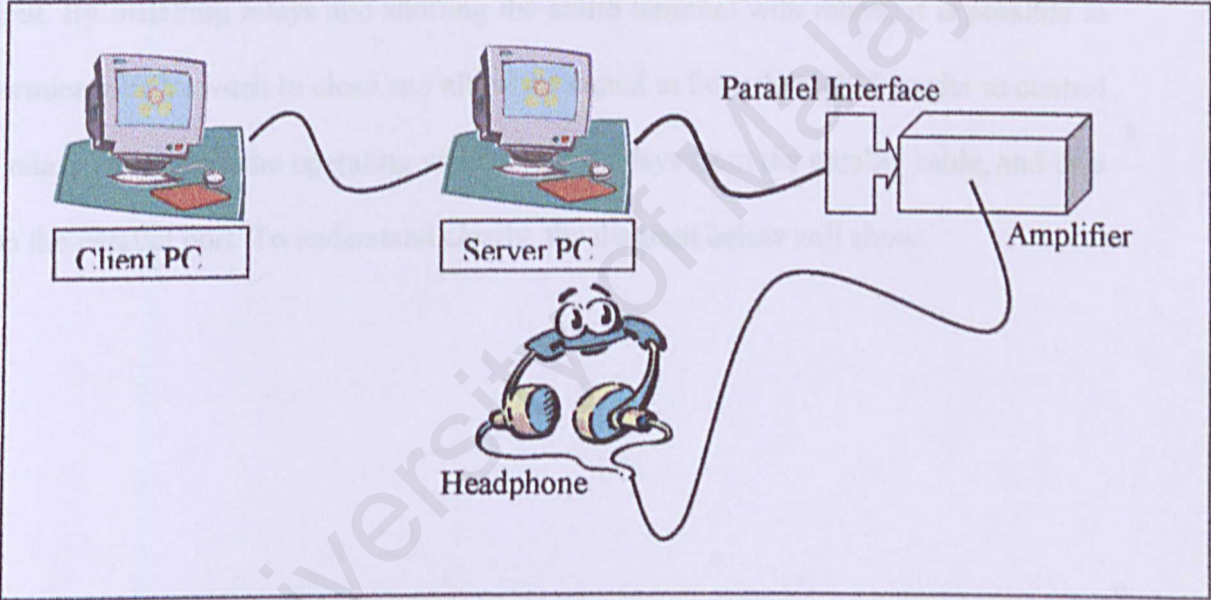


Figure 4.1: Highest Level of Design

4.2 Designs For Controlling Stepped Attenuator

Based on the three approaches mentioned in chapter 3, the verdict will be the third approach that is by using the Relay. As mentioned also in chapter 3, the relay will play the role as a switch on the stepped attenuator. This is due to the concept on how a stepped attenuator works.

On the actual attenuator, the rotary switch act as a switch, whereby when we rotate the switch, it will short on the terminals to allow the signal flows through into the output. By installing relays and shorting the entire terminal with relays, it is possible to determine which switch to close and allow the signal to flow through. In order to control the relays, we supply the operating voltage to the relays from the parallel cable, and thus from the parallel port. To understand clearly, the diagram below will show:

Series Type Stepped Attenuator

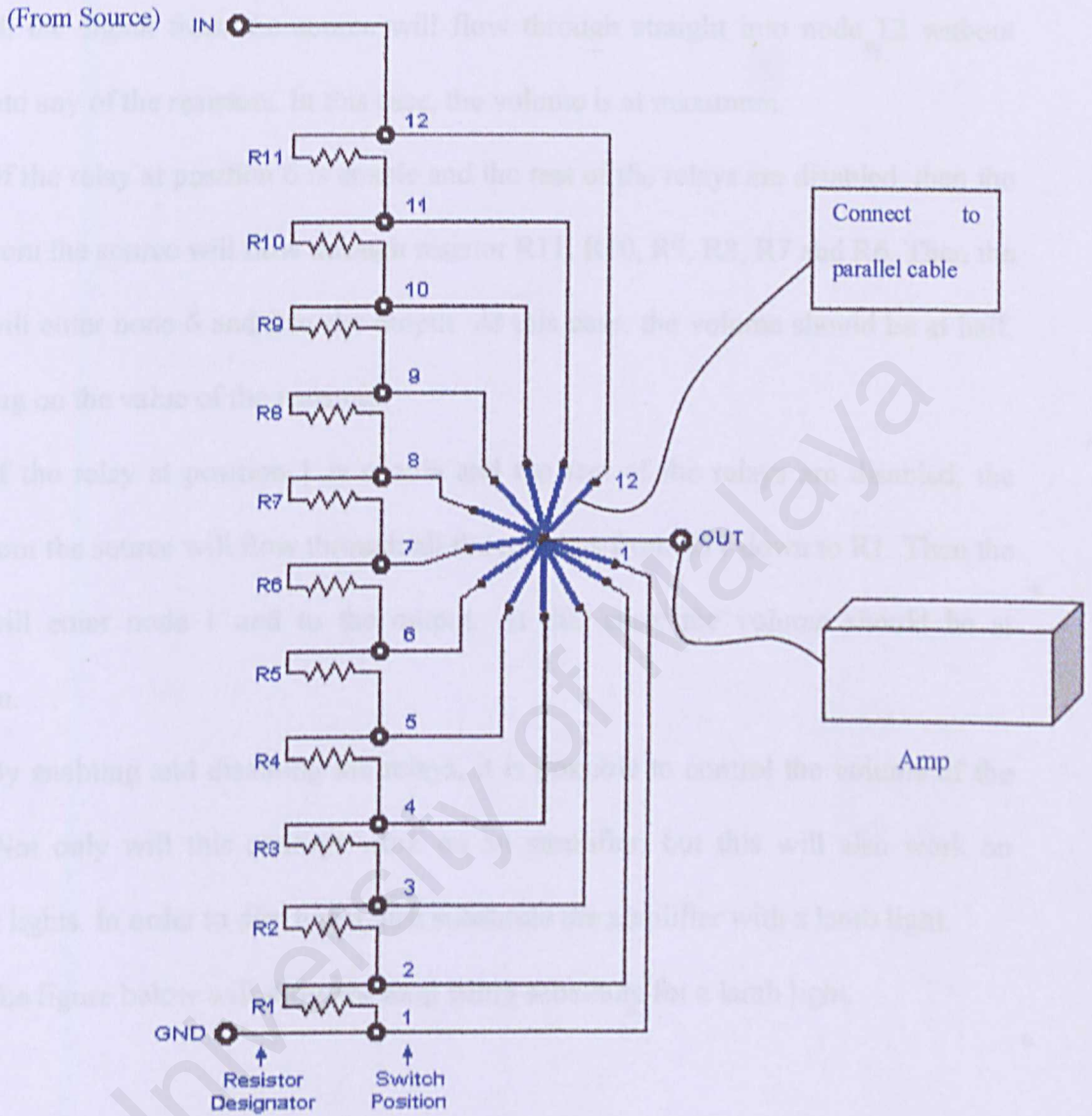


Figure 4.2: Design For Controlling Stepped Attenuator

The Blue bold lines represent relays connected from the stepped attenuator to the amp. The relays power supplies are from the parallel cable. Whenever the data pins are enabled, it will supply power to the relays.

For example, if relay at position 12 is enable and the rest of the relays are disabled, the signal from the source will flow through straight into node 12 without going into any of the resistors. In this case, the volume is at maximum.

If the relay at position 6 is enable and the rest of the relays are disabled, then the signal from the source will flow through resistor R11, R10, R9, R8, R7 and R6. Then the signal will enter node 6 and into the output. At this case, the volume should be at half, depending on the value of the resistors.

If the relay at position 1 is enable and the rest of the relays are disabled, the signal from the source will flow through all the resistors from R11 down to R1. Then the signal will enter node 1 and to the output. At this case, the volume should be at minimum.

By enabling and disabling the relays, it is possible to control the volume of the source. Not only will this concept work on an amplifier, but this will also work on dimming lights. In order to dim lights, just substitute the amplifier with a lamb light.

The figure below will show the amp being substitute for a lamb light.



Figure 3.2: Controlling and Dimming a Light Bulb

But in the context of this project, the 1.5V will be used instead of a light bulb.

Series Type Stepped Attenuator

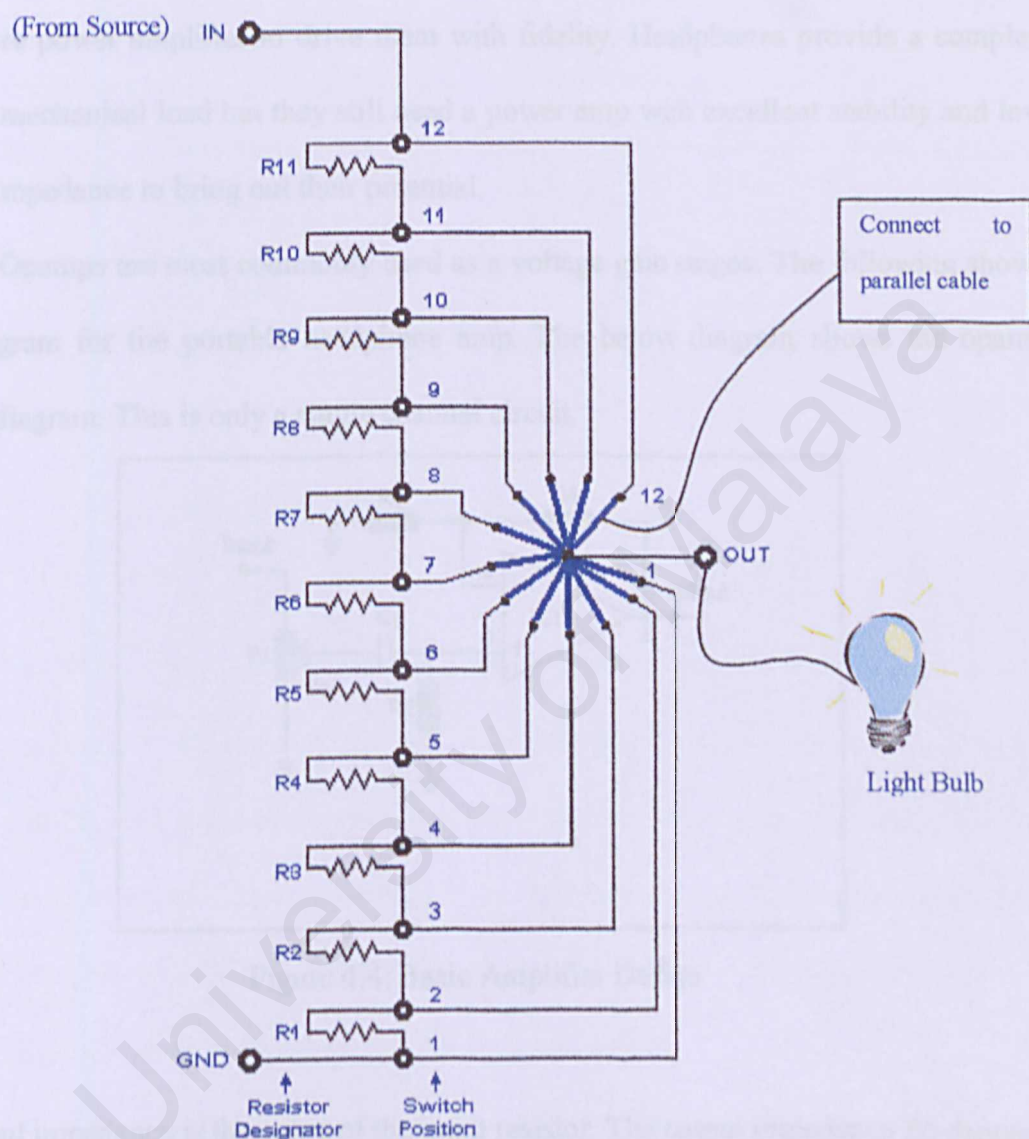


Figure 4.3: Controlling and Dimming a Light Bulb

But in the case of this project, the LED will be used instead of a light bulb.

4.3 Portable Headphone Amplifier Design

The fact is that headphones are actually miniature speakers and require a miniature power amplifier to drive them with fidelity. Headphones provide a complex electro-mechanical load but they still need a power amp with excellent stability and low output impedance to bring out their potential.

Opamps are most commonly used as a voltage gain stages. The following shows the diagram for the portable headphone amp. The below diagram shows the opamp circuit diagram. This is only a single channel circuit.

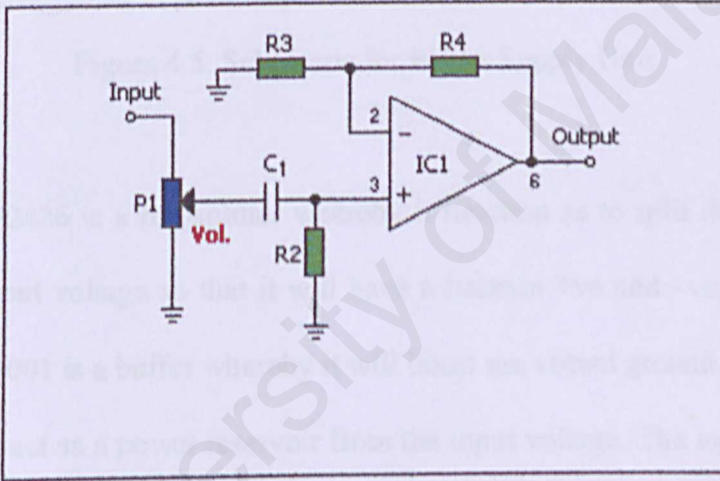


Figure 4.4: Basic Amplifier Design

The input impedance is the value of the input resistor. The output impedance Z_o depends on the particular opamp, but will decrease with decrease in gain. The gain of the amp is shown in the equation below;

$$G = (R3 + R4) / R3$$

P1 will be substitute for the stepped attenuator.

opamp. **Tip** to lower the current. This excess of current could drive the headphones higher

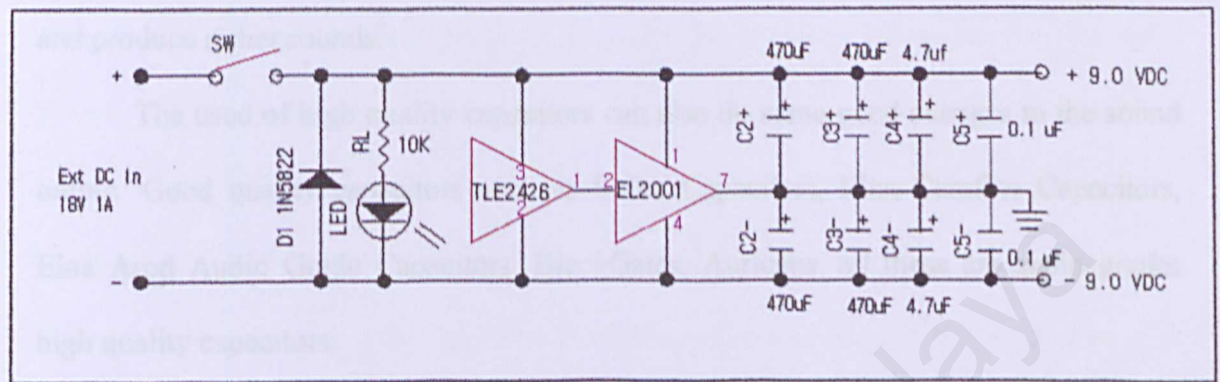


Figure 4.5: Schematic for Power Supply Unit

The TLE2426 is a rail-splitter whereby it function as to split the virtual ground and also the output voltage so that it will have a balance +ve and -ve to supply to the opamp. The EL2001 is a buffer whereby it will boost the virtual ground current. The rest of the capacitors act as a power reservoir from the input voltage. The input voltages will be batteries powered. D1 or 1N5822 is a diode and act as a precaution for reserved power. This will prevent the damage to the semiconductors.

The overall design for the portable amp will be as follow:

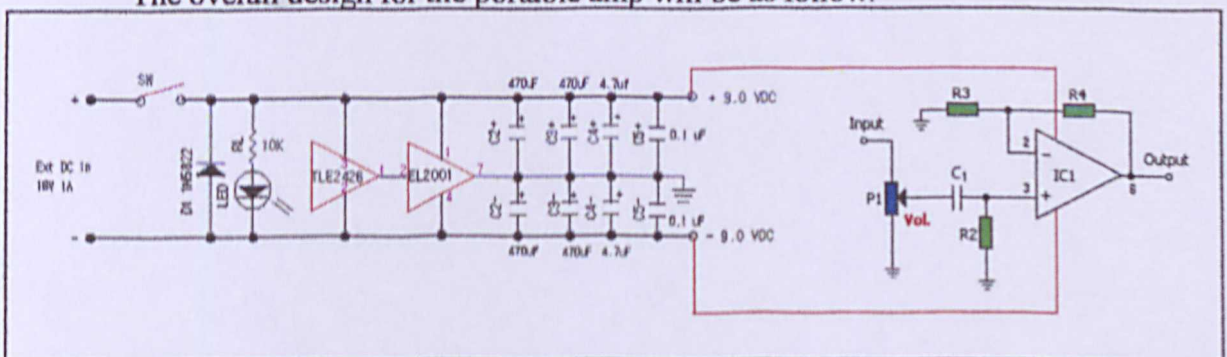


Figure 4.6: Complete Design of Headphone Amp

Over the stages of building the amp, they might be some changes to the amp as they are still rooms for improvement. For example, we could add a buffer at the output of the amp to boost the current. This boost of current could drive the headphone better and produce richer sounds.

The used of high quality capacitors can also do some good changes to the sound output. Good quality capacitors such as Solen Capacitors, Elna Cerafine Capacitors, Elna Arod Audio Grade Capacitors, BlackGates, Auricaps, all these are audio grades high quality capacitors.

Also, they can be varieties of opamps to use too. Instead of using the Burr Brown OPA2134, we can also use the Burr Brown OPA2132.

For the buffer, we can also opt for the BUF634 which is cheaper than the EL2001.

4.4 Software Design

In order to control the stepped attenuator through the parallel port, it is actually the software that is playing with the parallel ports data pins. The software role is to provide a graphical user interface or GUI to the user to control the volume of the audio system.

The software will be written in 2 programs, one for the client, and another for the controlling end. Therefore, in writing the codes, some network programming is needed, for example socket programming. The controlling end will control the parallel port by allowing the user to select which data pin to disable and enable.

The software will be fully written in Visual C++. The next section will show the data flow diagram for the software. The software will be named Administrator Mode and User Mode. The Administrator Mode will only allow the administrator or whoever has the admin password to enter this mode. The features in this mode include the administrator features such as adding users, deleting users and log file updates.

The User Mode is the most basic mode where there are only basic functions available. Basic functions such as controlling the volume and activating the amp.

The data flow diagrams are as follows:

a) Administrator Mode

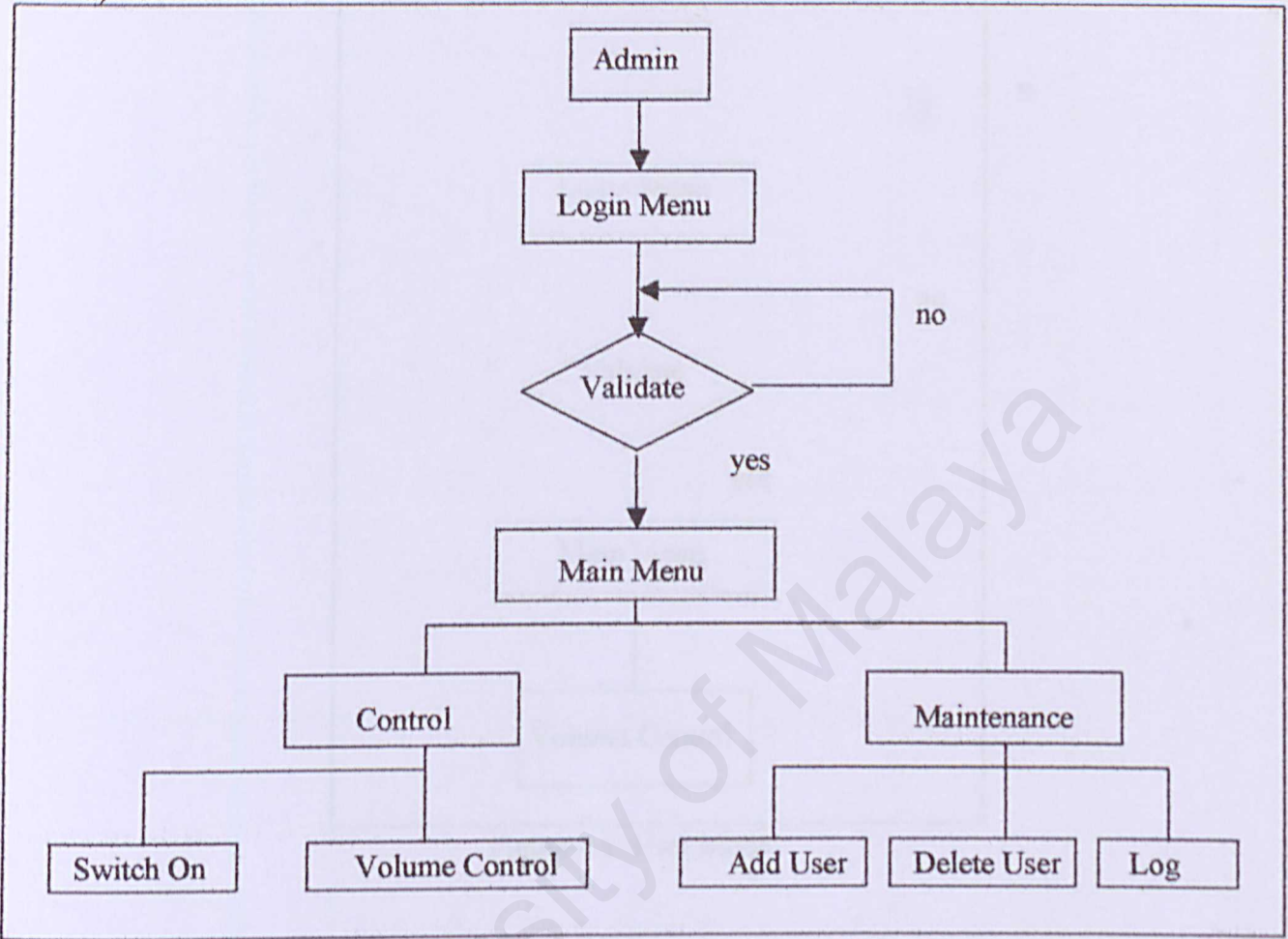


Figure 4.7: Administrator Mode

b) User Mode

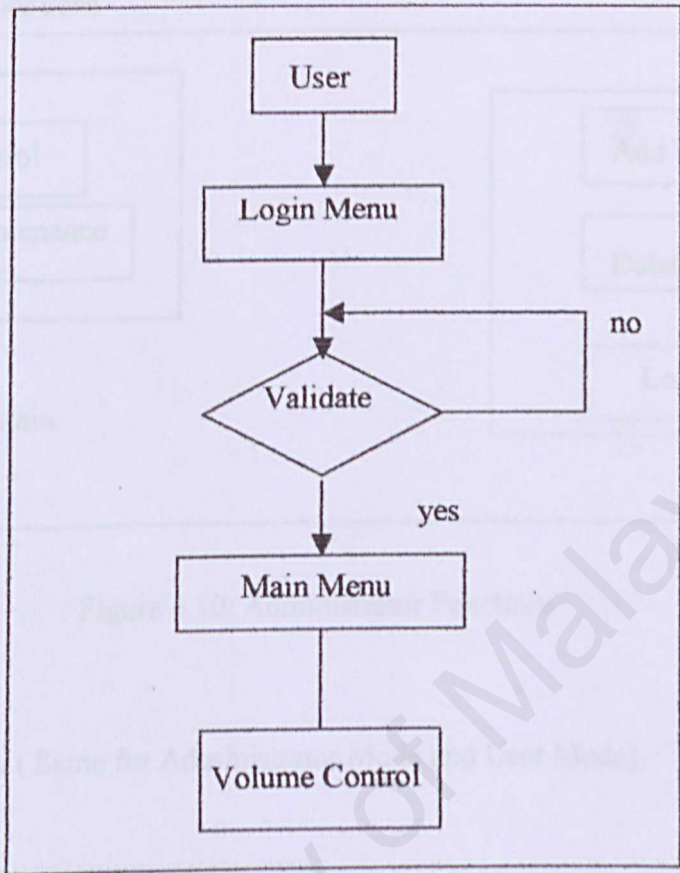


Figure 4.8: User Mode

c) Login Menu

The Login Menu form contains two input fields: 'Login:' and 'Password:'. Below these fields are two buttons: 'Ok' and 'Cancel'.

Figure 4.9: Login Menu

d) Administrator Functions

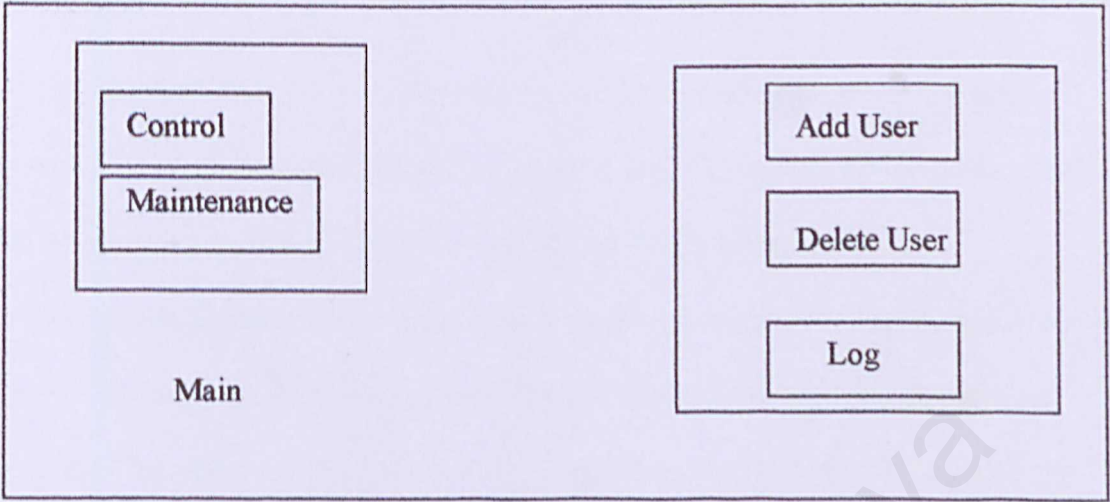


Figure 4.10: Administrator Functions

e) Controlling Menu (Same for Administrator Mode and User Mode)

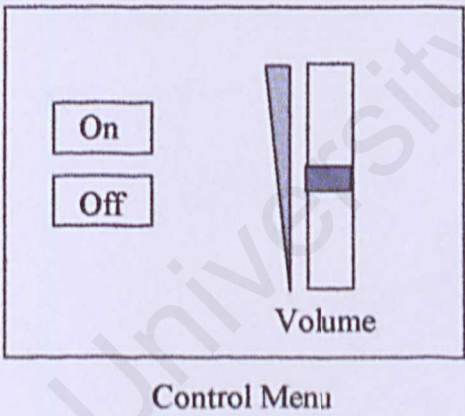


Figure 4.11: Control Menu

1.1 Introduction

This chapter will explain the development of the system or product. Which is based on the integration of modules and the system derived from the system design plan. The module functions by coding a data software and hardware programs.

This system consists of 2 components, the software component and the hardware component. The software component affects the data flow to be entered and processed in the system. The hardware component controls the hardware and will find the application based on the data flow.

CHAPTER FIVE: SYSTEM DEVELOPMENT AND IMPLEMENTATION

The following section of the chapter will discuss the system development and implementation. The system development is the process of creating a system that meets the requirements of the user. The implementation is the process of putting the system into operation.

5.1 Introduction

This chapter will explain the development of the system or model, which is based on the transition of modules and algorithms derived from the system design phase, into feasible instructions by coding it into software and hardware programs.

This system consists of 2 components, the software component and the hardware component. The software component allows the end user to have access and control into the system. The software also controls the hardware and will further control the application desired by the end user. The hardware functions as a controlling device, triggered by the software. The hardware component enables the end user to control the application based on the functions of the software.

The following sections of this chapter will explain the development process of both the software and hardware components. Then later will show how both software and hardware combines and implement it on the application.

5.2 Software Development

This section will explain the software development process. But before going into the details, let's first look into what does the software component do.

The software component consists of two programs running simultaneously. One part of the software is the client program, which is running remotely on a personal computer. And the second part of the software is the server program, which is also running remotely on a personal computer. The software is running across a network, meaning that both client and server programs on 2 separate personal computers are connected via the Internet or a local area network.

The client program communicates with the server program through a network environment. The client program, which resides remotely, will be able to communicate with the server program. The system is designed to allow the end user to have control over the client program and communicate with the server program to control the application.

Basically the client and server program are the same. Both of these programs have options to allow the user to select if the program should be running under the client mode or the server mode. This is to allow easy software development so that only one program is written. Also this allows the flexibility of the single program to run under multiple modes set by the user. During the coding process, both client and server functions must be written into the program. Therefore, the coding process should be longer and tedious.

Apart from the mode selection function, the program also allows text messaging and control functions on the hardware. The client and server program are able to communicate using text message. This is to enable both parties to exchange simple information such as text.

The control function of the software is to control the parallel port at the server computer. The function of the program is to be able to *enable* and *disable* the data pins of the parallel port. There are 8 data pins out of the 25 pins of the parallel port. Enabling the data pins is when there is a voltage present at the pins, where disabling the pins is when there is no voltage presence on the pins. The figure below shows the position of the data pins. The connector shown is a male connector.

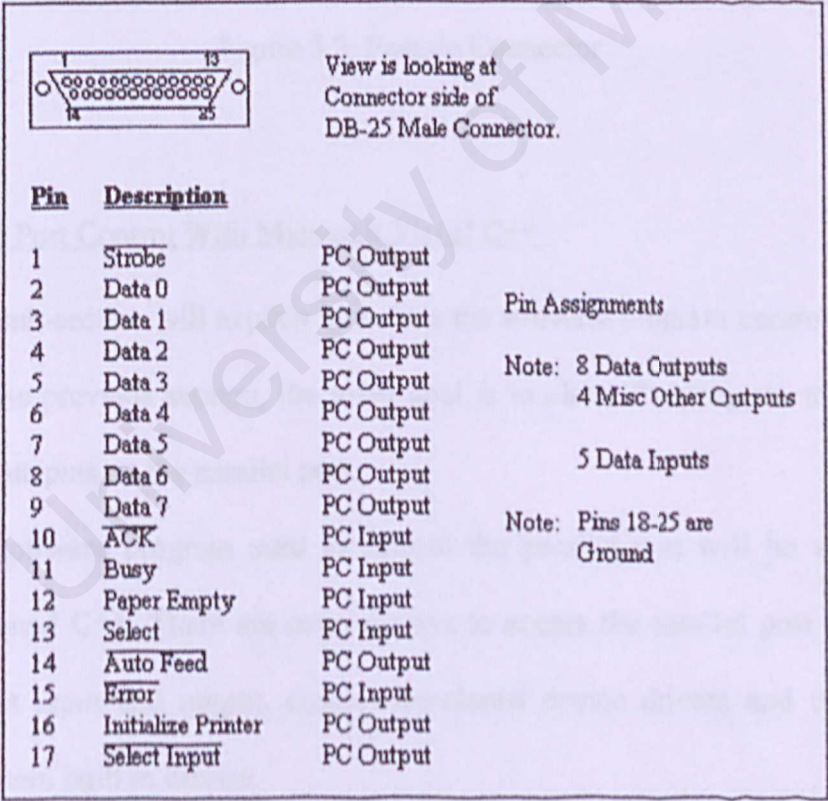


Figure 5.1: Positions of Pin

Based on the above figure, pin 2, 3, 4, 5, 6, 7, 8, and 9 are the data pins. There are a total of 8 data pins. This system will only deal with these 8 pins, and omitting the rest of the pins.

The figure below shows the female connector for the 25 pins parallel port.

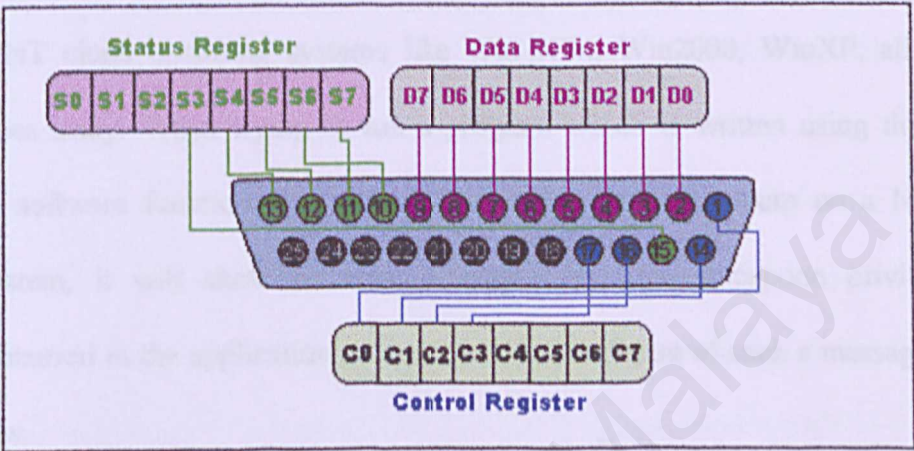


Figure 5.2: Female Connector

5.2.1 Parallel Port Control With Microsoft Visual C++

This sub-section will explain how does the software program control the parallel port. From the previous section, the main goal is to allow the program to enable and disable the data pins on the parallel port.

The software program used to control the parallel port will be written using Microsoft Visual C++. There are several ways to access the parallel port. These ways include direct input and output, custom developed device drivers and the Windows operating system built in drivers.

Almost all programming languages allow programmers to access parallel port using some library functions. For instances, Borland C is providing *Inportb* and *Outportb* functions to read and write IO mapped peripherals. In Microsoft Visual C++,

there are 2 functions to access IO mapped peripherals, `_inp` for reading and `_outp` for writing. These functions are declared in "conio.h" [1].

By using `Inporb` and `outportb` or `_inp()` or `_outp` functions in our program, there should be without any problem if running the program on Dos or Win95/98. But with the new era of NT clone operating systems like Win NT4, Win2000, WinXP, all this simplicity goes away. When trying to run a program which is written using the conventional software functions like `Inporb`, `outportb`, `_inp()` or `_Outp` on a NT or Win2000 system, it will show an error message that "The exception privileged instruction occurred in the application at location". The figure of such a messagebox is given below.

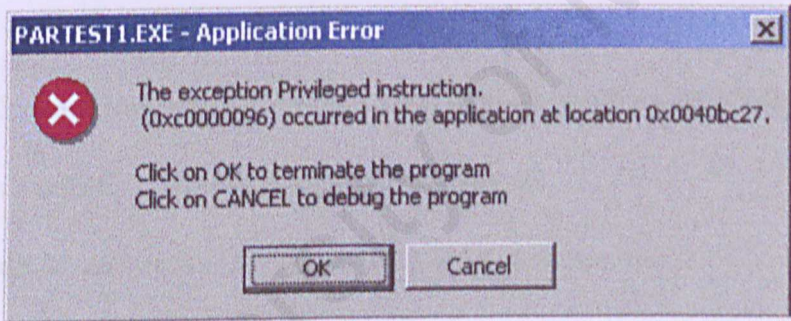


Figure 5.3: Error Message

The above error message only happens under the NT operating system, but the program is running perfectly flawless under the Windows 98 operating system. This is because being a very secure operating system, Windows NT assigns some privileges and restrictions to different types of programs running on it. It classifies all the programs into two categories, User mode and Kernel mode, for example running in ring3 and ring0 modes. User mode programs are running in ring3 mode and Kernel mode programs are running in ring0 mode. The program that will be written falls in the user mode category.

The user mode program is restricted to use certain instructions such as IN, OUT. Whenever the operating system finds that a user mode program is trying to execute such instructions, the operating system stops execution of that program and will display an error message. Eventually the interfacing program stops executing IN or OUT instructions to read or write data to parallel port. But in the same time Kernel mode program are in no way restricted in executing such instructions.

Device drivers are capable of running in kernel mode. So the workaround for the above stated problem is to write a kernel mode driver capable of reading and writing data to parallel port and let the user mode program to communicate with it [2]. The device driver is referring to the *inpout32.dll* for the Windows NT/2000/XP operating systems. The *inpout32.dll* has the following features:

- It works seamless with all versions of Windows including 98/NT/2000/XP
- It uses a kernel mode driver embedded in the dll
- No addition software or driver installation is required
- Driver will be automatically installed and configured automatically when the dll is loaded
- No special APIs are required, only 2 functions *inp32* and *out32*
- Can be easily used with Visual Basic and Visual C++

5.2.2 Coding For Parallel Port Communication

The following codes are written into the server program to allow the user to have control over the parallel port. The code utilizes the inpout32.dll device driver to have access into the parallel port's pins. The codes are as follows:

```
typedef UINT (CALLBACK* LPFNDLLFUNC1)(INT,INT);
typedef UINT (CALLBACK* LPFNDLLFUNC2)(INT);

HINSTANCE hDLL; // Handle to DLL

LPFNDLLFUNC1 Output; // Function pointer
LPFNDLLFUNC2 Input; // Function pointer

hDLL = LoadLibrary("Inpout32");

if (hDLL != NULL)
{
    Output = (LPFNDLLFUNC1)GetProcAddress(hDLL,"Out32");
    Input = (LPFNDLLFUNC2)GetProcAddress(hDLL,"Inp32");

    if (!Output || !Input)
    {
        // handle the error FreeLibrary(hDLL);
    }
}

Output(int, int);
```


decimal value will be 0. And for all 8 pins to be enabled the binary number will be

In order to write to the data pins at the parallel port, the command used is the *Output(int, int)*, where the first *int* represents the address of the parallel port, and the second *int* represents the value of the pins. The command for reading data from the parallel port is the *Input(int)*, where the *int* represents the address of the parallel port. However, for the purpose of this system, the *Input(int)* command will not be used because the program only has the writing function to the parallel port.

For each parallel port, there consist of three port addresses, namely the data port, status port, and the control port. These addresses are in sequential order. For instance, if the data port is at address 0x0378, the next status port will be at 0x0379 and the control port will be at 0x037a. The addresses are in hexadecimal numbers. Thus, only the data port address is needed to write into the data pins. Because a parallel port can have different modes, namely LPT1, LPT2 or LPT3, the port address for each mode is different. In order to make sure the mode of the parallel port, the user can check the modes in the System Configuration, under the Resource Settings. But normally the parallel port will be set to LPT1 in the BIOS, and the resources settings will be from 0378 to 037F. Therefore, for the purpose of this system, the data port address will be **0x0378**.

The value in *Output(int, int)* will determine which pin to activate. Let's look at the 8 data pins as binary numbers. Each data pin can only be either enable or disable. Thus let's assign the binary value of 1 for enable, and the binary value 0 for disable. Since there are 8 data pins, in the state where all pins are disabled the binary number will look like 00000000. Let's assume that the left most bit is pin 8 and the right most bit is pin 1. When the binary numbers are converted into decimal numbers, for 00000000 the

decimal value will be 0. And for all 8 pins to be enabled the binary number will be 11111111, and the decimal value will be 255. The following shows each pin being enabled accompanied by the binary and decimal values:

- All pins disabled

Binary: 00000000

Decimal: 0

- Pin 1 enabled

Binary: 00000001

Decimal: 1

- Pin 2 enabled

Binary: 00000010

Decimal: 2

- Pin 3 enabled

Binary: 00000100

Decimal: 4

- Pin 4 enabled

Binary: 00001000

Decimal: 8

- Pin 5 enabled

Binary: 00010000

Decimal: 16

- Pin 6 enabled

Binary: 00100000

Decimal: 32

- Pin 7 enabled

Binary: 01000000

Decimal: 64

- Pin 8 enabled

Binary: 10000000

Decimal: 128

- All pins enabled

Binary: 11111111

Decimal: 255

This system only has functions to enable each pin at a time. The system does not require 2 or more pins to be enabled. The following table shows the function:

Pin State	Decimal Values	Command in codes
All disabled	0	Output(0x0378, 0);
Pin 1	1	Output(0x0378, 1);
Pin 2	2	Output(0x0378, 2);
Pin 3	4	Output(0x0378, 4);
Pin 4	8	Output(0x0378, 8);
Pin 5	16	Output(0x0378, 16);
Pin 6	32	Output(0x0378, 32);
Pin 7	64	Output(0x0378, 64);
Pin 8	128	Output(0x0378, 128);

Table 5.1: Command for Pins

5.2.3 Socket Programming For Client And Server Program

The program will utilize socket programming in order to have communication between the client program and the server program. Socket programming allows more applications to have the ability to communicate with each other applications over a network, including the Internet.

Socket programming uses the same principles and functionality to perform communication. One program will sit on a computer, waiting for another program to open a communication connection. This program is in *listening* mode. Another program most likely on another computer will try to connect to the remote listening program. The connecting program has to know the network location, or network address of the remote program. Once the connection has been made, messages can pass back and forth between the two programs. This connection is a two-way communication channel, where both sides can send information.

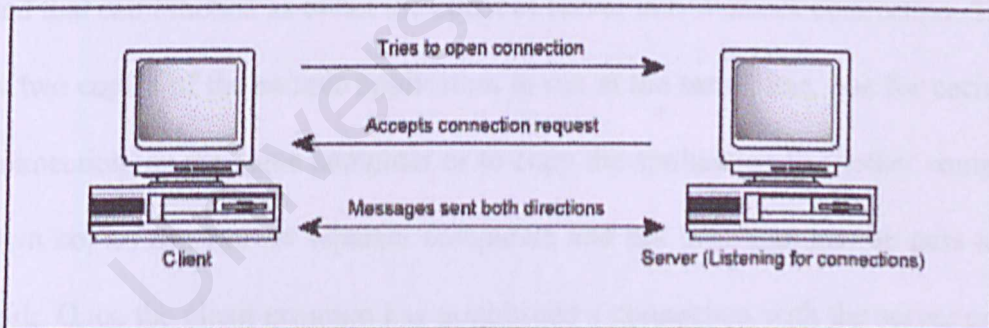


Figure 5.4: Socket Connection Process

The basic object used by applications to perform most network communications is called a socket. Sockets were first developed on UNIX at the University of California at Berkley. Sockets were designed so that most network communications between

applications could be performed in the same way that these same applications would read and write files.

When a file is to be read or written, a file object must be used to point to the file. A socket is similar; it is an object used to read and write messages that travel between applications.

In socket programming, only one application may be listening on any specific port on a single computer. Although numerous applications may listen for connection requests on a single computer at the same time, each of these applications must listen on a different port.

This program will be developed using Visual C++, and by using the MFC Winsock classes to add network communications capabilities. The base class, CAsyncSocket, provides complete, event-driven socket communications.

For the development of the socket program, a simple dialog application will be created that can function as either the client or server in a Winsock connection. This will allow two copies of the sample application to run at the same time, one for each end of the connection, on the same computer or to copy the application to another computer so that two copies can run on separate computers and see messages can be pass across a network. Once the client program has established a connection with the server program, text messages can be enter and send to the other program. When the message has been sent, it will be added to a list of messages sent. Each message that is received will be copied into another list of all messages received. This will allow the user to see the complete list of what is sent and received.

The few functions present on the program is to allow the user to set if the program is to be run as the client or server. Another important function is the slider

function, which allows the user to *increase* the slider and *decrease* the slider. Whenever the user changes the slider on the client side, the slider on the server side will changes too. The slider will integrate with the changes on the data pins of the parallel port. The slider will have 8 positions. Each position of the slider represents the data pins of the parallel port.

The figures below show a screen shot of both the client program and the server program at development time:

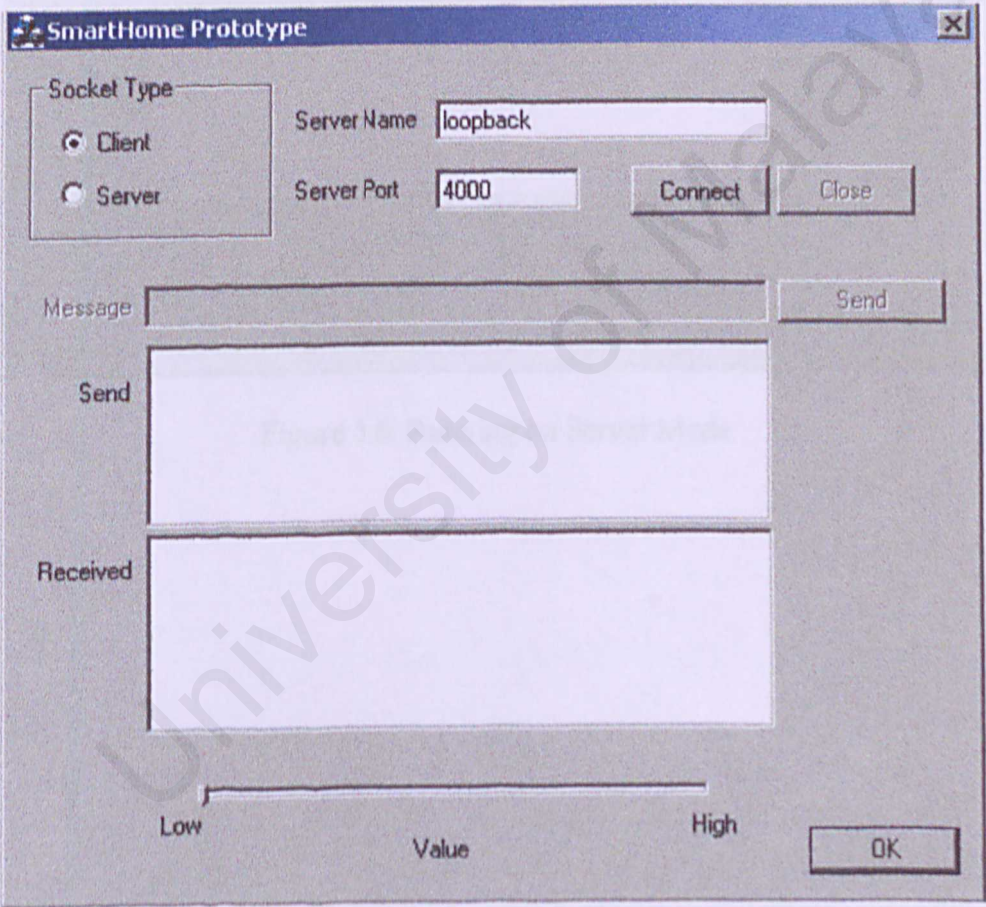


Figure 5.5: At initializing

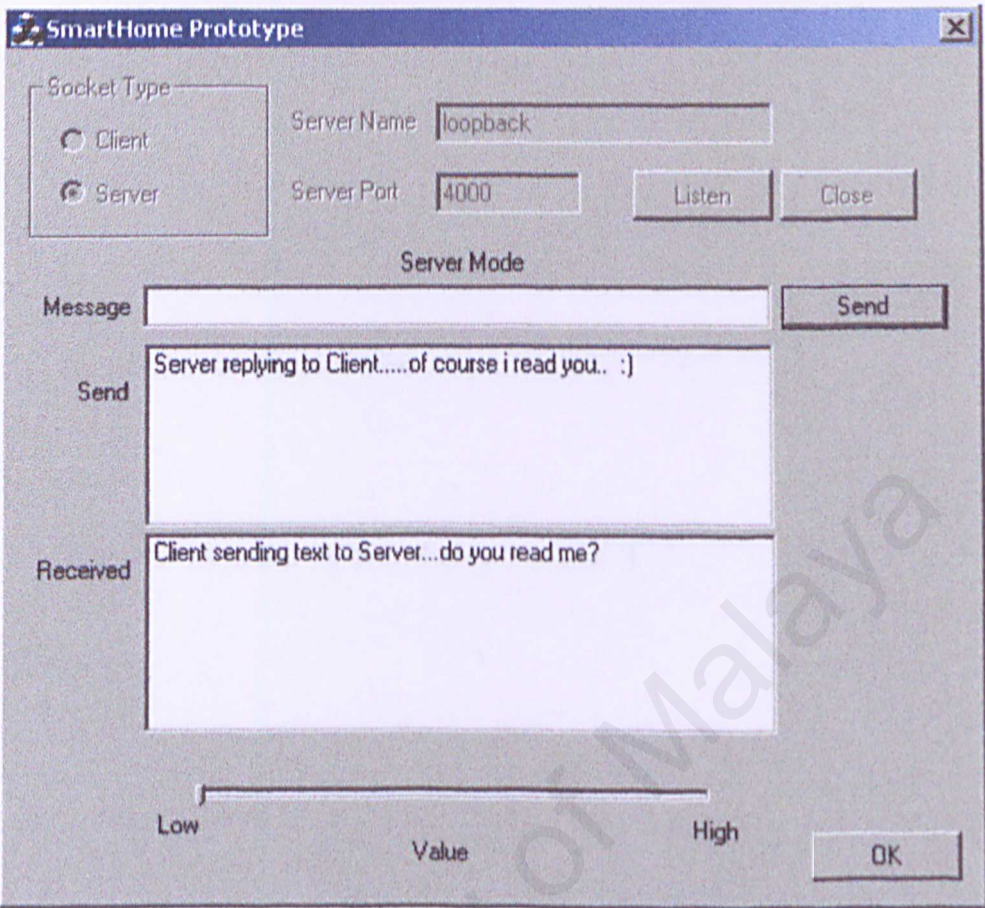


Figure 5.6: Running on Server Mode

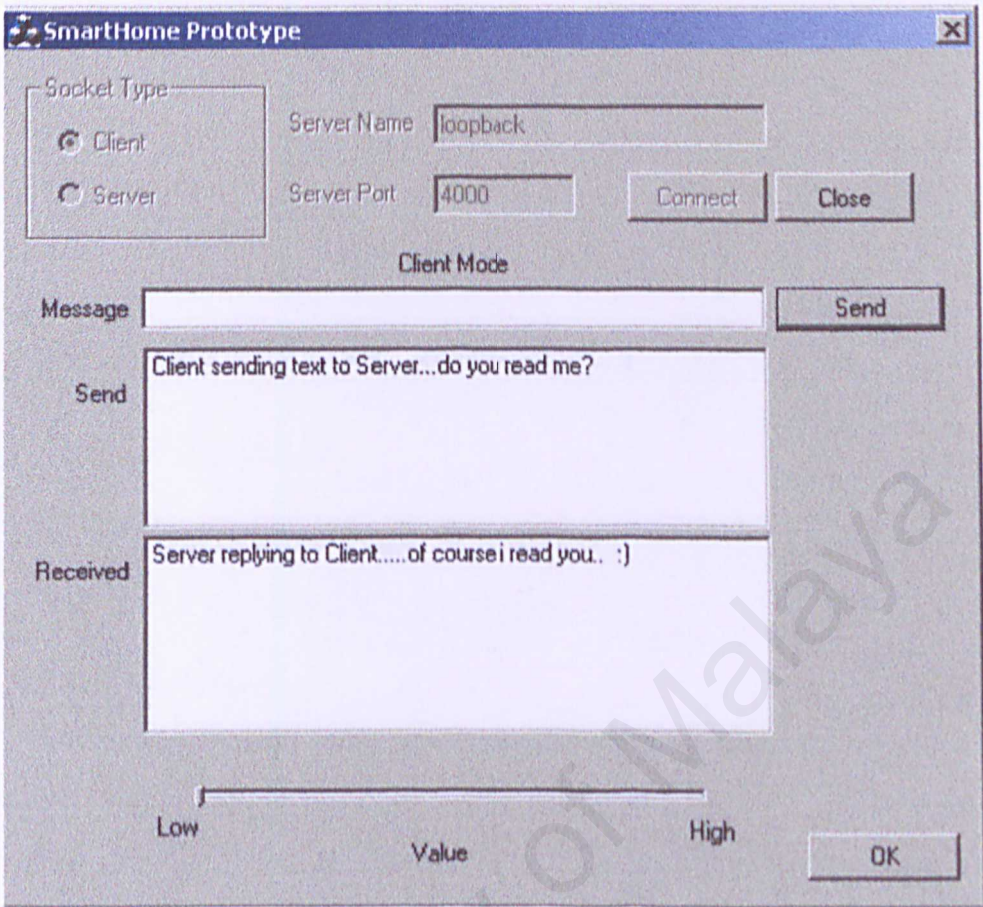


Figure 5.7: Running on Client Mode

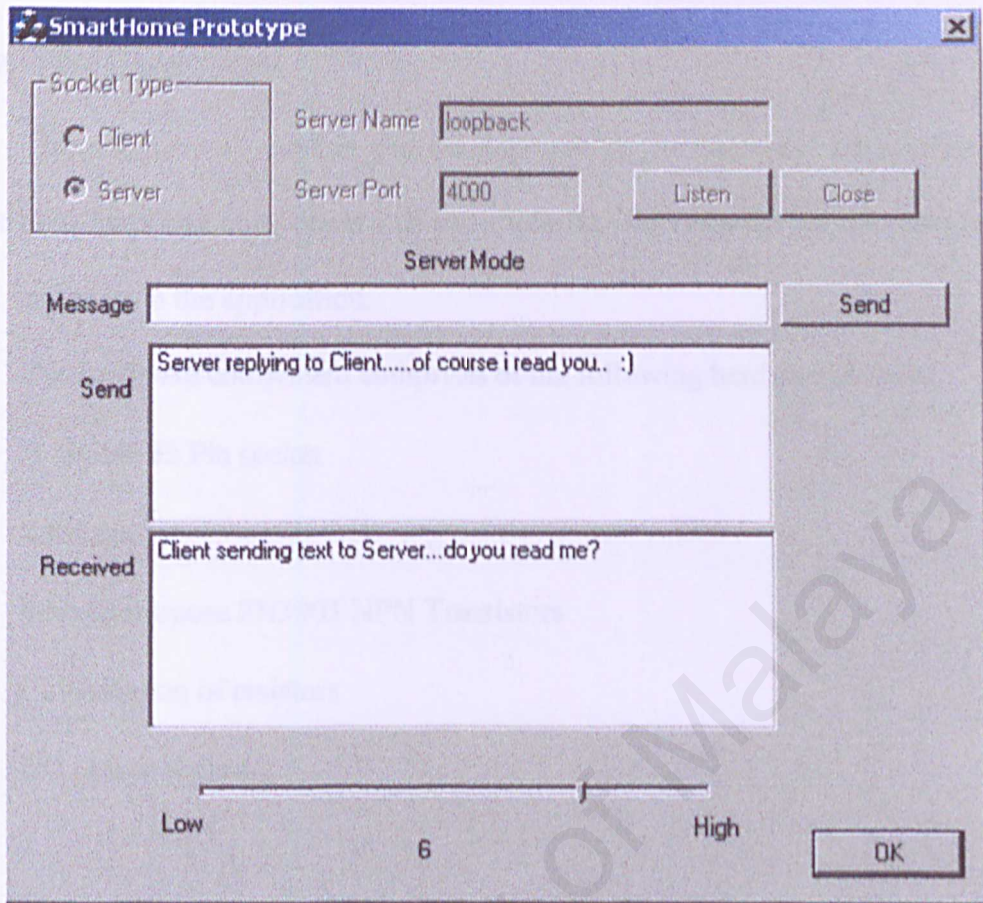


Figure 5.8: Server's slider changes

When the slider is at position 0, all the data pins are disabled. When the slider is increased to 1, pin 1 will be enabled. When the slider changes to position 2, pin 2 will be enabled and the rest of the pins will be disabled. When the slider changes to position 3, pin 3 will be enabled and so on until the maximum position of 8, and then pin 8 will be enabled.

5.3 Hardware Development

This section will explain the development of the hardware component of the system. The hardware component acts as an interface between the parallel port from the server computer to the application.

The hardware component comprises of the following hardware devices:

- A female 25 Pin socket
- 8 Relays
- 8 Multi-purpose 2N3903 NPN Transistors
- Combination of resistors
- DC power source

The figure below shows one part of the circuit from the parallel port interface:

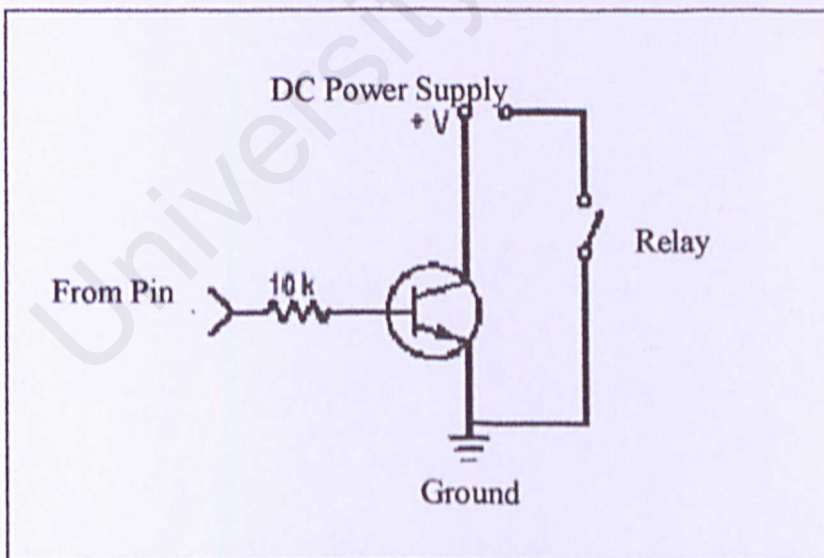


Figure 5.9: One part of the circuit

Based on the above figure, this is only one circuit connection for one pin. Therefore, there are 8 pins and there should be 8 circuit connections similar to the above figure.

The 2N3904 NPN multi-purpose transistor acts as a switch to enable the relay. When the pin at the parallel port is enabled, there is a small voltage present at the pin. The value of the voltage is around 5V. When this 5V is present at the Base of the transistor, this will allow current to flow from the Emitter of the transistor to the Collector of the transistor. Thus, the Base of the transistor acts as a switch to enable a close circuit for the Emitter and the Collector of the transistor. The direction of the current is from the Emitter down across the Base, to the Collector of the transistor.

When the pin is enabled at the parallel port, a voltage present at Base will complete the circuit at the Relay and the DC power source will power up the relay, causing the switch in the relay to close.

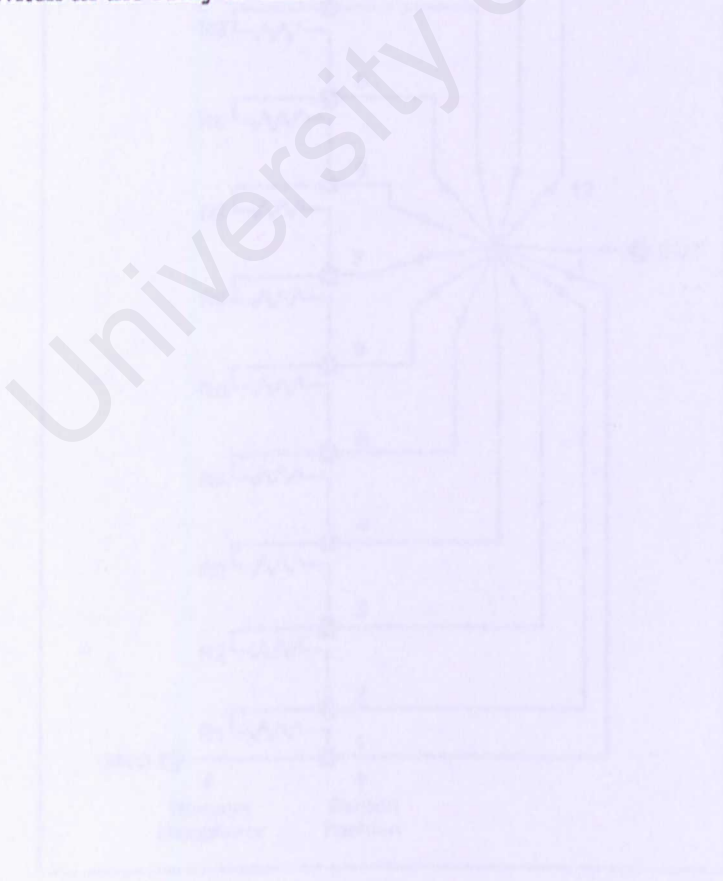


Figure 2.10.11 Relay Activation

5.3.1 Hardware Interface As A Potentiometer

The application here is to allow the end user to control a remotely situated audio amplifier. In order to control the audio amplifier, the hardware interface will have to act like a potentiometer, or a volume control device.

Before going into the details of the interface development, lets look at the figure below.

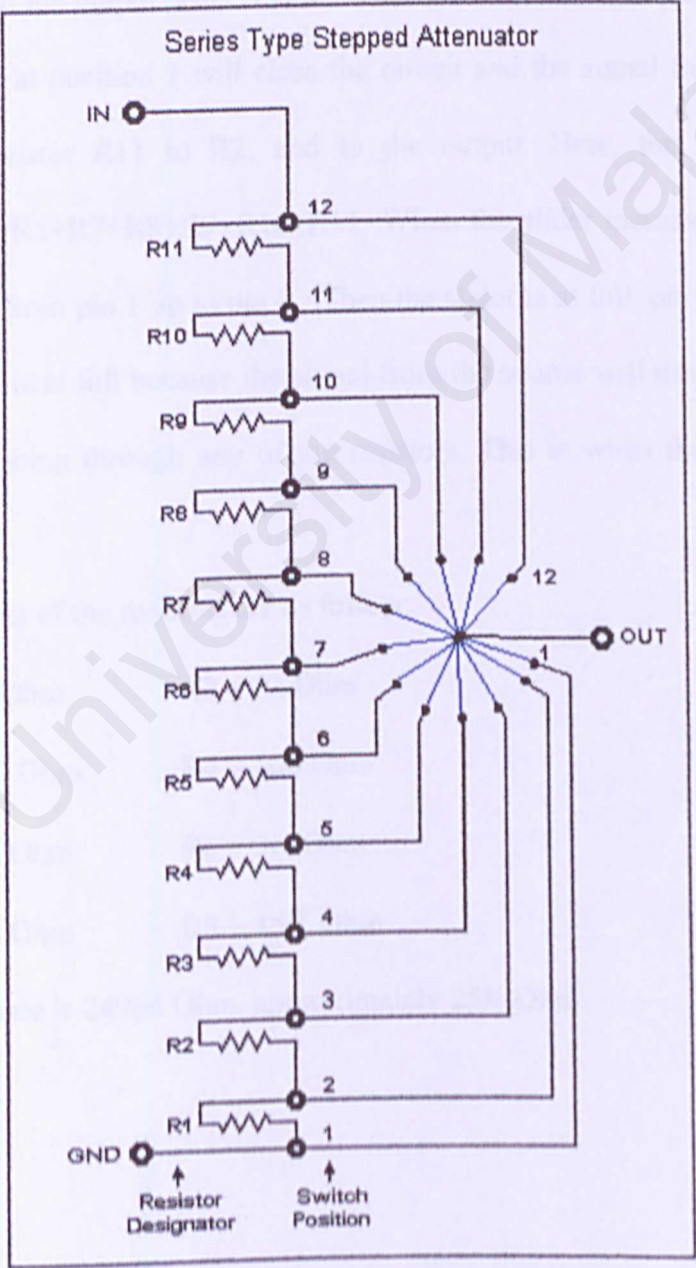


Figure 5.10: 12 Stepped Attenuator

The above figure shows a 12-stepped attenuator. Although for the purpose of this system, instead of developing a 12 stepped attenuator, an 8-stepped attenuator will be developed. This is because there are only 8 data pins used from the parallel port.

The blue coloured lines represent the relays. Each time the relay is enabled, it will act as a switch to short the circuit from the input source to the output. When all the data pins are disabled, none of the 8 relays will enable. Thus, the source signal will not be able to flow to the output. This is when the volume is at the lowest level. When pin 1 is enabled, relay at position 1 will close the circuit and the signal from the source will travel across resistor R11 to R2, and to the output. Here, the total resistance is $R2+R3+R4+R5+R6+R7+R8+R9+R10+R11$. When the slider increases its position, the pins will enable from pin 1 up to pin 8. When the slider is at full, pin 8 will enable. This time the volume is at full because the signal from the source will travel straight into the output without going through any of the resistors. This is when the volume is at full blast.

The values of the resistors are as follow:

- $R1 = 82 \text{ Ohm}$ $R2 = 82 \text{ Ohm}$
- $R3 = 240 \text{ Ohm}$ $R4 = 560 \text{ Ohm}$
- $R5 = 1\text{K Ohm}$ $R6 = 3\text{K Ohm}$
- $R7 = 5\text{K Ohm}$ $R8 = 15\text{K Ohm}$

The total resistance is 24964 Ohm, approximately 25K Ohm.

5.4 More on Inpout32.dll

The feature of Inpout32.dll is it can work with all the Windows versions without any modification in user code or the *dll* itself. The *dll* will check the operating system version when functions are called, and if the operating system is Win9X, the *dll* will use *_inp()* and *_outp* functions for reading and writing to the parallel port. If the operating system is Windows NT, 2000 or XP, it will install a kernel mode driver and talk to parallel port through that driver. The user code will not be aware of the OS version on which it is running. The *dll* can be used in Windows NT clone operating systems as if it is Win9X. The flow chart of the program is given below.

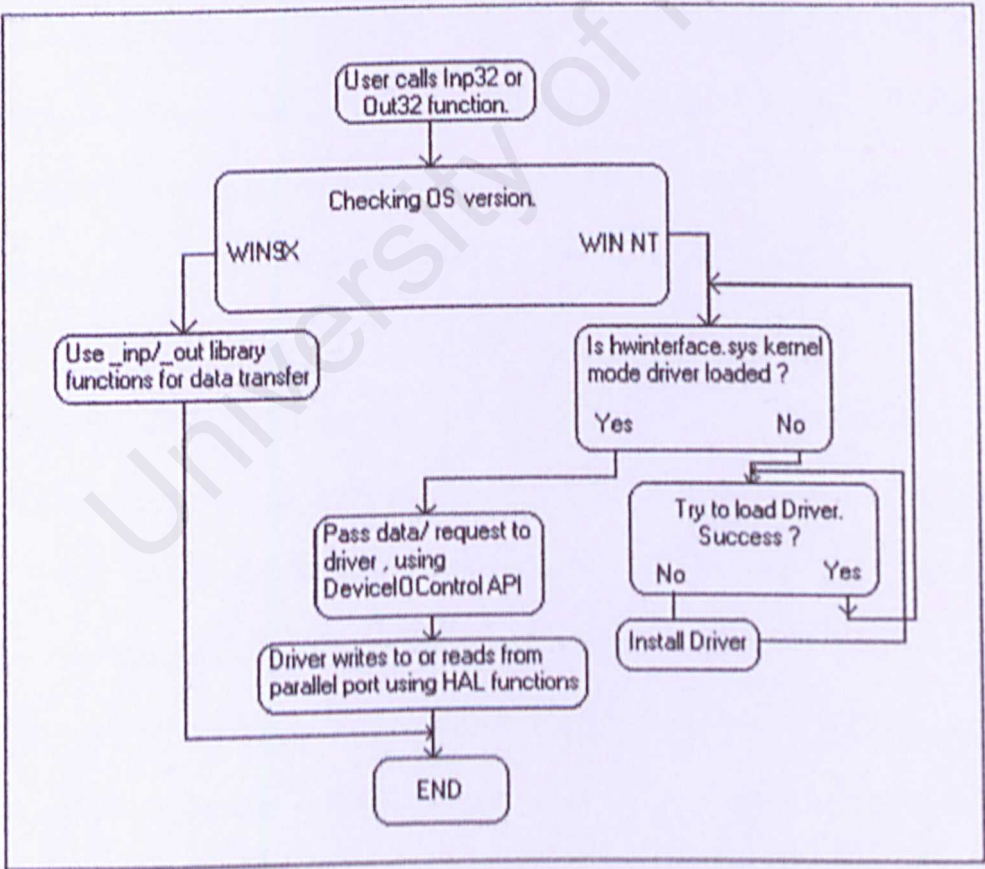


Figure 5.11: Inpout32.dll process flow chart

6.1 System Test

This chapter explains the test method carried out to test the final product of the system. The system consists of components, the software component and the hardware component. Therefore there should be at least two test method to check the performance of the system.

6.2 Software Test

CHAPTER SIX: SYSTEM TEST

In order to produce a quality product, it is important to have a systematic approach to testing the software. The software test is a process of checking the software for errors and defects. During the writing of the program, the programmer may make some mistakes and errors. These mistakes and errors may be in the form of syntax errors, logical errors, or runtime errors. It is also important to test the software to ensure that it meets the requirements and that it is reliable. The software test is a process of checking the software for errors and defects. The software test is a process of checking the software for errors and defects. The software test is a process of checking the software for errors and defects.

To test the software, the programmer should first test the program on a small scale. This is done by testing the program on a small number of data. The programmer should also test the program on a large scale. This is done by testing the program on a large number of data. The programmer should also test the program on a variety of data. This is done by testing the program on data that is different from the data used in the development of the program.

To test the program, the programmer should first test the program on a small scale. This is done by testing the program on a small number of data. The programmer should also test the program on a large scale. This is done by testing the program on a large number of data. The programmer should also test the program on a variety of data. This is done by testing the program on data that is different from the data used in the development of the program.

6.1 System Test Introduction

This chapter explains the test method carried out to test the final product of the system. The system contains 2 components, the software component and the hardware component. Therefore there should be at least two test method to check the performances of the system.

6.2 Software Test

In order to produce a good test result for the software component, it is important that the process of writing the programs are planned correctly and carried out smoothly. During the writing of the program, make sure that all the required functions and methods are drafted out clearly. It is also important to frequently compile and run the programs after each and every functions are written correctly. This way, whenever an error is found, at that moment the problem would be a small one and it should be easy to solve the problem.

To test the completed program, at first the client and server program must be executed. It is better to run both client and server program on the same computer initially. After testing the programs successfully, then proceed with running both programs remotely on different computer across a network.

To test the programs is to make sure that all the functions and features on the program works accordingly with the requirements. In this case, it is important that the client program is able to connect to the server program. Next is to ensure that both the

client and server program are able to send text messages across the network, and received by the opposite program.

Before sending the text messages, it is important to test if the single program can be run either in the client mode or the server mode. This is because the same program is written to be able to run as the server or the client. The way to test this is to run up 2 programs and select the different mode on each program, and then connect from the client side.

The most important function on the system is the slider control. In order to test if the slider is working on the parallel port's pins, a third party software is used to test the parallel port's pin status. The software used is the Parallel Port Monitor written by Fred Bulback. This software is a freeware and can be download from the Internet. The Parallel Port Monitor is a utility for viewing and manipulating the state of a parallel port on a Windows operating system. The figure below shows the screen shot of the software:

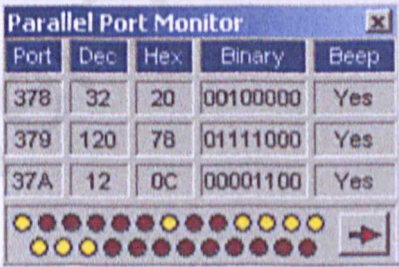


Figure 6.1: Parallel Port Monitor

As seen above, the coloured round dots represents the 25 pins of the parallel port. The dark red colour shows the pins that are not active, while the yellow pins are the active pins.

This software is ideally good for testing the software component of the system.

6.3 Hardware Test

The best way to test the hardware component is by using a multimeter. The multimeter meter used is the Sunwa YX-360TRn.

It is important to always probe any connection points after any process of soldering iron on the printed breadboard. This is a vital step because a slight shorted area will cause the hardware to malfunction.

After the completion of the hardware, it is possible to use a third party software to test on the circuit. The third party software used is the GXSPort by the XSTools. GXSPort is a graphical user interface command line utilities that sends up to 8 logic signals through the computer's parallel port. By using GXSPort, it is possible to test the hardware by enabling the data pins and enabling the relays. Below shows a screen shot of the software:

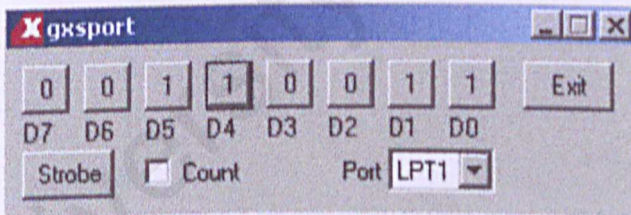


Figure 6.2: GXSPort

To use GXSPort, simply just click on the 8 buttons and the value will change from 0 to 1. This indicates that a high and a low value is assigned to the parallel port. After selecting the button, click Strobe to send the signal to the pins.

6.4 Integration Test

Integration test meaning to test the entire system, combining the software and the hardware. In order to run this test, firstly run a copy of the software program on 2 network computers. One computer will act as the server program, and another will act as the client program. On the server computer, by using a male-to-male parallel cable, connect the cable to the parallel port, and then connect the other end into the hardware interface component. Then connect the power supply for the hardware interface component and connect the hardware component to the audio amplifier. Next play an audio source and try changing the slider control on the client program. If the test work, the output from the speaker should change in audio volume.

2.1 Discussion

This chapter will explain the problems faced during the development of the system. Also will discuss the strength of the system, and not forgetting the constraint and weakness. Finally, this chapter will include suggestions and improvements for this project.

2.1 Previous Project

CHAPTER SEVEN: DISCUSSION

1) To find the right device for a switch

During the first phase of this project, the task was to find a hardware device that could act as a volume control. They came along the line of the digital attenuator. But the problem was to search for the right device to act as a switch whenever there is a small amount of voltage present. This was a difficult task because a lot of expensive electronic components had to be tried. Finally, a relay solved the problem.

2) Software and Hardware in Windows XP

While writing the code for the program, the platform used to compile the source code was the Windows XP operating system. The readily available functions for accessing the parallel port were unable to run correctly. This was due to security issues on the XP operating system. Further research was done on the internet, and a device driver was needed to overcome the security constraints of the Windows XP operating system. The device driver was the `lptutils.dll`.

7.1 Discussion

This chapter will explain the problems faced during the development of the system. Also will discuss the strength of the system, and not forgetting the constraint and weaknesses. Finally this chapter will include suggestions and improvements for this project.

7.2 Problems Faced

Along the development process of this project, there are a few problem faced:

i) To find the right device for a switch

During the first phase of this project, the idea was to build a hardware device that could act as a volume control. Then came along the idea of the stepped attenuator. But the problem was to search for the right device to act as a switch whenever there is a small amount of voltage present. This proved to be a difficult task because a lot of extra readings on electrical components need to be done. Finally, a relay solved the problem.

ii) Software unable to run on Windows XP

While writing the codes for the program, the platform used to compile the source code was the Windows XP operating system. The readily available functions for accessing the parallel port were unable to run correctly. This was due to security reasons on the XP operating system. Further research was done on the Internet, and a device driver was needed to overwrite the security constraint on the Windows XP operating system. The device driver used was the Inpout32.dll.

iii) Finding the right resistors values

This was quite a difficult task too because 8 different values of resistors need to be calculated to produced the right attenuation. A series of try and error process finally came down very close to the right value of resistors.

iv) Insufficient Funding

This system utilizes a great amount of hardware devices. Therefore a large sum of money is used to purchase these electrical devices. Some of the devices are quite costly, and needed to get in a large amount of quantities.

7.3 Strength Of Project

Although they have been some previous projects similar to this one, where it involves the usage of the parallel port. But the previous projects are quite straightforward in the sense of the applications. *This project proves to be unique in the sense that it combines both hardware knowledge and software programming.*

The hardware utilizes a number of different electronic devices, each with different functionalities, combined together to produce a unique solution for different application. For example, the main purpose of this project is to control an audio amplifier. But the same function can be applied on a light bulb.

The software uses the slider control to allow for smooth transition of control on the parallel port. This proves to be creative and different from the traditional button controls.

7.4 System Constraint

A few drawbacks of the system are that a delay in information transfers across the network. The software performance for running on the same computer and running remotely differs in the sense that information needs to be broadcast across the network. Therefore there is a delay when the server program receives the information.

Another drawback of the system is that, the system is actually running in a local area network environment. The algorithm for the software to communicate across the Internet should prove to be more complicated than to communicate across a network computer.

Finally, this project is only a model design to show that a parallel port could actually act as a volume control and to control an audio amplifier.

7.5 Suggestions and Improvements

1. The system can be improved by using more pins to control
2. More pins to allow much more attenuation, thus a smoother volume change.
3. The make the software able to run across the Internet.
4. The Faculty should provide with some funding.
5. Using other medium of control, for instance wireless control.

7.6 Conclusion

This project proves to be a challenging task. This is because the system comprises of both hardware and software. Apart from applying the knowledge in computer programming, a certain degree of knowledge in hardware and electronics are needed to complete this project. This is where extra knowledge is gain and could be useful for future purposes.

Due to the problems encountered during development, it is sad to mention that the developer of the project was almost behind schedule. But with great dedication and beliefs, the developer managed at the end to complete the project.

Along this project, the developer gained many experiences. With these experiences gained, the developer hopes that all these will be put into practice when graduate from university.

BIBLIOGRAPHY

1. Jan Axelson, "Parallel Port Complete. Programming, interfacing & Using PC's Parallel Printer Port", Lakeview Research, 1999.
2. Jan Axelson, "Serial Port Complete. Programming and Circuits for RS-232 and RS-485 Links and Networks", Lakeview Research, 1998.
3. Thomas L Floyd, "Digital Fundamentals", 6 Edition, Prentice Hall, 1997.
4. www.smarthome.com
5. www.x10.com
6. www.google.com
7. www.howstuffworks.com
8. www.ti.com
9. www.head-fi.org

APPENDIX 1: CLIENT SCRIPT

```
// SockDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Sock.h"
#include "SockDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
   //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    }}AFX_DATA

    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CAboutDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    }}AFX_VIRTUAL

// Implementation
protected:
    {{{AFX_MSG(CAboutDlg)
    }}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    {{{AFX_DATA_INIT(CAboutDlg)
    }}}AFX_DATA_INIT
}
```

```
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
```

```
{  
    CDialog::DoDataExchange(pDX);  
   //{{AFX_DATA_MAP(CAboutDlg)  
   //}}AFX_DATA_MAP  
}
```

```
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
```

```
//{{AFX_MSG_MAP(CAboutDlg)
```

```
    // No message handlers
```

```
//}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()
```

```
////////////////////////////////////
```

```
// CSockDlg dialog
```

```
CSockDlg::CSockDlg(CWnd* pParent /*=NULL*/)
```

```
: CDialog(CSockDlg::IDD, pParent)
```

```
{  
    {{{AFX_DATA_INIT(CSockDlg)
```

```
    m_strMessage = _T("");
```

```
    m_strName = _T("");
```

```
    m_iPort = 0;
```

```
    m_iType = -1;
```

```
    }}}AFX_DATA_INIT
```

```
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
```

```
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
```

```
}
```

```
void CSockDlg::DoDataExchange(CDataExchange* pDX)
```

```
{  
    CDialog::DoDataExchange(pDX);  
    {{{AFX_DATA_MAP(CSockDlg)  
    DDX_Control(pDX, IDC_SLIDER, m_ctlSlider);  
    DDX_Control(pDX, IDC_LSENT, m_ctlSent);  
    DDX_Control(pDX, IDC_LRECVD, m_ctlRecv);  
    DDX_Control(pDX, IDC_BCONNECT, m_ctlConnect);  
    DDX_Text(pDX, IDC_EMSG, m_strMessage);  
    DDX_Text(pDX, IDC_ESERVNAME, m_strName);  
    DDX_Text(pDX, IDC_ESERVPORT, m_iPort);  
    DDX_Radio(pDX, IDC_RCLIENT, m_iType);  
    }}}AFX_DATA_MAP  
}
```



```

BEGIN_MESSAGE_MAP(CSockDlg, CDialog)
//{{AFX_MSG_MAP(CSockDlg)
ON_WM_SYSCOMMAND()
ON_WM_PAINT()
ON_WM_QUERYDRAGICON()
ON_BN_CLICKED(IDC_RCLIENT, OnRType)
ON_BN_CLICKED(IDC_BCONNECT, OnBconnect)
ON_BN_CLICKED(IDC_BSEND, OnBsend)
ON_BN_CLICKED(IDC_RSERVER, OnRType)
ON_BN_CLICKED(IDC_BCLOSE, OnBclose)
ON_WM_HSCROLL()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

```

```

////////////////////////////////////
// CSockDlg message handlers

```

```

BOOL CSockDlg::OnInitDialog()
{

```

```

    CDialog::OnInitDialog();

```

```

    // Add "About..." menu item to system menu.

```

```

    // IDM_ABOUTBOX must be in the system command range.

```

```

    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);

```

```

    ASSERT(IDM_ABOUTBOX < 0xF000);

```

```

    CMenu* pSysMenu = GetSystemMenu(FALSE);

```

```

    if (pSysMenu != NULL)
    {

```

```

        CString strAboutMenu;

```

```

        strAboutMenu.LoadString(IDS_ABOUTBOX);

```

```

        if (!strAboutMenu.IsEmpty())
        {

```

```

            pSysMenu->AppendMenu(MF_SEPARATOR);

```

```

            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,

```

```

strAboutMenu);

```

```

        }
    }

```

```

    // Set the icon for this dialog. The framework does this automatically

```

```

    // when the application's main window is not a dialog

```

```

    SetIcon(m_hIcon, TRUE);

```

```

    // Set big icon

```

```

    SetIcon(m_hIcon, FALSE);

```

```

    // Set small icon

```



```

// TODO: Add extra initialization here
// Initialize the control variables
m_iType = 0;
m_strName = "loopback";
m_iPort = 4000;
// Update the controls
UpdateData(FALSE);
//GetDlgItem(IDC_SLIDER)->EnableWindow(TRUE);
// Set the socket dialog pointers
m_sConnectSocket.SetParent(this);
m_sListenSocket.SetParent(this);

return TRUE; // return TRUE unless you set the focus to a control
}

```

```

void CSockDlg::OnSysCommand(UINT nID, LPARAM lParam)
{

```

```

    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

```

// If you add a minimize button to your dialog, you will need the code below
 // to draw the icon. For MFC applications using the document/view model,
 // this is automatically done for you by the framework.

```

void CSockDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM)
dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
    }
}

```

```
int x = (rect.Width() - cxIcon + 1) / 2;  
int y = (rect.Height() - cyIcon + 1) / 2;
```

```
// Draw the icon
```

```
dc.DrawIcon(x, y, m_hIcon);
```

```
}
```

```
else
```

```
{
```

```
    CDialog::OnPaint();
```

```
}
```

```
// The system calls this to obtain the cursor to display while the user drags  
// the minimized window.
```

```
HCURSOR CSockDlg::OnQueryDragIcon()
```

```
{  
    return (HCURSOR) m_hIcon;  
}
```

```
void CSockDlg::OnRTType()
```

```
{  
    // TODO: Add your control notification handler code here  
    // Sync the controls with the variables  
    UpdateData(TRUE);  
    // Which mode are we in?  
    if (m_iType == 0) { // Set the appropriate text on the button  
        m_ctlConnect.SetWindowText("C&onnect");  
    }  
    else {  
        m_ctlConnect.SetWindowText("&Listen");  
    }  
}
```

```
void CSockDlg::OnBconnect()
```

```
{  
    // TODO: Add your control notification handler code here  
    // Sync the variables with the controls  
    UpdateData(TRUE);  
    // Disable the connection and type controls  
    GetDlgItem(IDC_BCONNECT)->EnableWindow(FALSE);  
    GetDlgItem(IDC_ESERVNAME)->EnableWindow(FALSE);  
    GetDlgItem(IDC_ESERVPORT)->EnableWindow(FALSE);  
    GetDlgItem(IDC_STATICNAME)->EnableWindow(FALSE);  
    GetDlgItem(IDC_STATICPORT)->EnableWindow(FALSE);  
}
```



```

GetDlgItem(IDC_RCLIENT)->EnableWindow(FALSE);
GetDlgItem(IDC_RSERVER)->EnableWindow(FALSE);
GetDlgItem(IDC_STATICTYPE)->EnableWindow(FALSE);

// Are we running as client or server?
if (m_iType == 0){
    GetDlgItem(IDC_SLIDER)->EnableWindow(TRUE);
    GetDlgItem(IDC_LABELMODE)->EnableWindow(TRUE);
    GetDlgItem(IDC_LABELMODE)->SetWindowText("Client Mode");
    // Client, create a default socket
    m_sConnectSocket.Create();
    // Open the connection to the server
    m_sConnectSocket.Connect(m_strName, m_iPort);

    m_ctlSlider.SetRange(0, 10);
}
else{
    GetDlgItem(IDC_SLIDER)->EnableWindow(TRUE);
    GetDlgItem(IDC_LABELMODE)->EnableWindow(TRUE);
    GetDlgItem(IDC_LABELMODE)->SetWindowText("Server Mode");
    // Server, create a socket bound to the port specified
    m_sListenSocket.Create(m_iPort);
    // Listen for connection requests
    m_sListenSocket.Listen();
    m_ctlSlider.SetRange(0, 8);

    /**
    *****

    typedef UINT (CALLBACK* LPFNDLLFUNC1)(INT,INT);
    typedef UINT (CALLBACK* LPFNDLLFUNC2)(INT);
    HINSTANCE hDLL; // Handle to DLL
    LPFNDLLFUNC1 Output; // Function pointer
    LPFNDLLFUNC2 Input; // Function pointer
    INT Addr;
    INT AddrIn;
    INT Value;
    hDLL = LoadLibrary("Inpout32");
    if (hDLL != NULL)
    {
        Output = (LPFNDLLFUNC1)GetProcAddress(hDLL,"Out32");
        Input = (LPFNDLLFUNC2)GetProcAddress(hDLL,"Inp32");
        if (!Output || !Input)
        {
            // handle the error FreeLibrary(hDLL);
        }
    }
    */

```



```

    }
    Addr = 0x378;
    AddrIn = 0x379;
    Value = 0;
    Output(Addr, Value);
    INT somenum = Input(Addr);

```

```

    /*******

```

```

    */

```

```

    }
}

```

```

void CSockDlg::OnAccept()
{

```

```

    // Accept the connection request
    m_sListenSocket.Accept(m_sConnectSocket);
    // Enable the text and message controls
    GetDlgItem(IDC_EMSG)->EnableWindow(TRUE);
    GetDlgItem(IDC_BSEND)->EnableWindow(TRUE);
    GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);

```

```

void CSockDlg::OnConnect()
{

```

```

    // Enable the text and message controls
    GetDlgItem(IDC_EMSG)->EnableWindow(TRUE);
    GetDlgItem(IDC_BSEND)->EnableWindow(TRUE);
    GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
    GetDlgItem(IDC_BCLOSE)->EnableWindow(TRUE);

```

```

void CSockDlg::OnSend()
{

```

```

void CSockDlg::OnReceive()
{

```

```

    char *pBuf = new char[1025];
    int iBufSize = 1024;
    int iRcvd;
    CString strRcvd;

```

```

    // From client site controlling

```

```

typedef UINT (CALLBACK* LPFNDLLFUNC1)(INT,INT);
typedef UINT (CALLBACK* LPFNDLLFUNC2)(INT);
HINSTANCE hDLL; // Handle to DLL
LPFNDLLFUNC1 Output; // Function pointer
LPFNDLLFUNC2 Input; // Function pointer
INT Addr;
INT AddrIn;
INT Value;
hDLL = LoadLibrary("Inpout32");
if (hDLL != NULL)
{
    Output = (LPFNDLLFUNC1)GetProcAddress(hDLL,"Out32");
    Input = (LPFNDLLFUNC2)GetProcAddress(hDLL,"Inp32");
    if (!Output || !Input)
    {
        // handle the error FreeLibrary(hDLL);
    }
}
Addr = 0x378;
AddrIn = 0x379;
Value = 0;
//Output(Addr, Value);
INT somenum = Input(Addr);

// Receive the message
iRcvd = m_sConnectSocket.Receive(pBuf, iBufSize);
// Did we receive anything?
if (iRcvd == SOCKET_ERROR){
}
else{
    pBuf[iRcvd] = NULL;
    //if (pBuf==int){
    //    GetDlgItem(IDC_VALUE)->SetWindowText(pBuf);
    //}
    //else{

    // Copy the message to a CString
    strRecvd = pBuf;
    //i=0;
    if(strRecvd=="000" || strRecvd=="000000"){
        GetDlgItem(IDC_VALUE)->SetWindowText("0");
        //i=1;
        //m_ctlRecvd.AddString(strRecvd);
        m_ctlSlider.SetPos(0);
    }
}

```



```

        Output(0x378,0);
    }
    else if(strRecv == "001" || strRecv == "001001"){
        //m_ctlRecv.AddString(strRecv);
        GetDlgItem(IDC_VALUE)->SetWindowText("1");
        //i=1;
        m_ctlSlider.SetPos(1);
        Output(0x378,1);
    }
    else if(strRecv == "002" || strRecv == "002002"){
        //m_ctlRecv.AddString(strRecv);
        GetDlgItem(IDC_VALUE)->SetWindowText("2");
        //i=1;
        m_ctlSlider.SetPos(2);
        Output(0x378,2);
    }
    else if(strRecv == "003" || strRecv == "003003"){
        //m_ctlRecv.AddString(strRecv);
        GetDlgItem(IDC_VALUE)->SetWindowText("3");
        //i=1;
        m_ctlSlider.SetPos(3);
        Output(0x378,4);
    }
    else if(strRecv == "004" || strRecv == "004004"){
        //m_ctlRecv.AddString(strRecv);
        GetDlgItem(IDC_VALUE)->SetWindowText("4");
        //i=1;
        m_ctlSlider.SetPos(4);
        Output(0x378,8);
    }
    else if(strRecv == "005" || strRecv == "005005"){
        //m_ctlRecv.AddString(strRecv);
        GetDlgItem(IDC_VALUE)->SetWindowText("5");
        //i=1;
        m_ctlSlider.SetPos(5);
        Output(0x378,16);
    }
    else if(strRecv == "006" || strRecv == "006006"){
        //m_ctlRecv.AddString(strRecv);
        GetDlgItem(IDC_VALUE)->SetWindowText("6");
        //i=1;
        m_ctlSlider.SetPos(6);
        Output(0x378,32);
    }
    else if(strRecv == "007" || strRecv == "007007"){
        //m_ctlRecv.AddString(strRecv);

```



```

        GetDlgItem(IDC_VALUE)->SetWindowText("7");
        //i=1;
        m_ctlSlider.SetPos(7);
        Output(0x378,64);
    }
    else if(strRecvd=="008" || strRecvd=="008008"){
        //m_ctlRecv.AddString(strRecvd);
        GetDlgItem(IDC_VALUE)->SetWindowText("8");
        //i=1;
        m_ctlSlider.SetPos(8);
        Output(0x378,128);
    }
    else if(strRecvd=="009" || strRecvd=="009009"){
        //m_ctlRecv.AddString(strRecvd);
        GetDlgItem(IDC_VALUE)->SetWindowText("9");
        //i=1;
        m_ctlSlider.SetPos(9);
        Output(0x378,128);
    }
    else if(strRecvd=="010" || strRecvd=="010010"){
        //m_ctlRecv.AddString(strRecvd);
        GetDlgItem(IDC_VALUE)->SetWindowText("10");
        //i=1;
        m_ctlSlider.SetPos(10);
        Output(0x378,128);
    }
    else if(strRecvd=="6623"){
        //MessageBox("Client Disconnected");
        m_ctlRecv.AddString("**** Client Disconnected ****");
    }
    else{
        // Add the message to the received list box
        m_ctlRecv.AddString(strRecvd);
        //GetDlgItem(IDC_VALUE)->SetWindowText(strRecvd);
        // Sync the variables with the controls
    }
    //}
    //}
    UpdateData(TRUE);
}
}

```

```

void CSockDlg::OnClose()
{

```

```

    // Close the connected socket
    m_sConnectSocket.Close();

```

```
//Disable the message sending controls
GetDlgItem(IDC_EMSG)->EnableWindow(FALSE);
GetDlgItem(IDC_BSEND)->EnableWindow(FALSE);
GetDlgItem(IDC_STATICMSG)->EnableWindow(FALSE);
GetDlgItem(IDC_BCLOSE)->EnableWindow(FALSE);
```

```
// Are we running in Client mode?
```

```
if (m_iType == 0){
    // Yes, so enable the connection configuration controls
    GetDlgItem(IDC_BCONNECT)->EnableWindow(TRUE);
    GetDlgItem(IDC_ESERVNAME)->EnableWindow(TRUE);
    GetDlgItem(IDC_ESERVPORT)->EnableWindow(TRUE);
    GetDlgItem(IDC_STATICNAME)->EnableWindow(TRUE);
    GetDlgItem(IDC_STATICPORT)->EnableWindow(TRUE);
    GetDlgItem(IDC_RCLIENT)->EnableWindow(TRUE);
    GetDlgItem(IDC_RSERVER)->EnableWindow(TRUE);
    GetDlgItem(IDC_STATICTYPE)->EnableWindow(TRUE);
    GetDlgItem(IDC_LABELMODE)->EnableWindow(FALSE);
    GetDlgItem(IDC_LABELMODE)->SetWindowText("");
}
```

```
void CSockDlg::OnBsend()
```

```
// TODO: Add your control notification handler code here
```

```
int iLen;
```

```
int iSent;
```

```
// Sync the controls with the variables
```

```
UpdateData(TRUE);
```

```
// Is there a message to be sent?
```

```
if(m_strMessage != ""){
```

```
    // Get the length of the message
```

```
    iLen = m_strMessage.GetLength();
```

```
    // Send the message
```

```
    iSent = m_sConnectSocket.Send(LPCTSTR(m_strMessage), iLen);
```

```
    // Were we able to send it?
```

```
    if(iSent == SOCKET_ERROR){
```

```
    }
```

```
    else{
```

```
        // Add the message to the list box.
```

```
        m_ctlSent.AddString(m_strMessage);
```

```
        GetDlgItem(IDC_EMSG)->SetWindowText("");
```

```
        // Sync the variables with the controls
```

```
        UpdateData(TRUE);
```



```

    }
}

void CSockDlg::OnBclose()
{
    // TODO: Add your control notification handler code here
    m_sConnectSocket.Send(LPCTSTR("6623"), 4);

    // Call the OnClose function
    OnClose();
}

void CSockDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    CString pos;
    pos.Format("%d", m_ctlSlider.GetPos());
    GetDlgItem(IDC_VALUE)->SetWindowText(pos);

    //if (m_iType == 0){
    if(pos == "0")
        m_sConnectSocket.Send(LPCTSTR("000"),3);
    if(pos == "1")
        m_sConnectSocket.Send(LPCTSTR("001"),3);
    if(pos == "2")
        m_sConnectSocket.Send(LPCTSTR("002"),3);
    if(pos == "3")
        m_sConnectSocket.Send(LPCTSTR("003"),3);
    if(pos == "4")
        m_sConnectSocket.Send(LPCTSTR("004"),3);
    if(pos == "5")
        m_sConnectSocket.Send(LPCTSTR("005"),3);
    if(pos == "6")
        m_sConnectSocket.Send(LPCTSTR("006"),3);
    if(pos == "7")
        m_sConnectSocket.Send(LPCTSTR("007"),3);
    if(pos == "8")
        m_sConnectSocket.Send(LPCTSTR("008"),3);
    if(pos == "9")
        m_sConnectSocket.Send(LPCTSTR("009"),3);
    if(pos == "10")
        m_sConnectSocket.Send(LPCTSTR("010"),3);
    //}
    if (m_iType == 1){
        // For server site controlling
    }
}

```



```

typedef UINT (CALLBACK* LPFNDLLFUNC1)(INT,INT);
typedef UINT (CALLBACK* LPFNDLLFUNC2)(INT);
HINSTANCE hDLL; // Handle to DLL
LPFNDLLFUNC1 Output; // Function pointer
LPFNDLLFUNC2 Input; // Function pointer
INT Addr;
INT AddrIn;
INT Value;
hDLL = LoadLibrary("Inpout32");
if (hDLL != NULL)
{
    Output = (LPFNDLLFUNC1)GetProcAddress(hDLL,"Out32");
    Input = (LPFNDLLFUNC2)GetProcAddress(hDLL,"Inp32");
    if (!Output || !Input)
    {
        // handle the error FreeLibrary(hDLL);
    }
}
Addr = 0x378;
AddrIn = 0x379;
Value = 0;
//Output(Addr, Value);
INT somenum = Input(Addr);

if(m_ctlSlider.GetPos()==0){
    Output(0x378,0);
}

//***** PIN 1
*****//
if(m_ctlSlider.GetPos()>0 && m_ctlSlider.GetPos()<=1){
    Output(0x378,1);
}

//***** PIN 2
*****//
if(m_ctlSlider.GetPos()>1 && m_ctlSlider.GetPos()<=2){
    Output(0x378,2);
}

//***** PIN 3
*****//
if(m_ctlSlider.GetPos()>2 && m_ctlSlider.GetPos()<=3){
    Output(0x378,4);
}

```

```
}
```

```
//***** PIN 4
```

```
*****//
```

```
if(m_ctlSlider.GetPos()>3 && m_ctlSlider.GetPos()<=4){  
    Output(0x378,8);
```

```
}
```

```
//***** PIN 5
```

```
*****//
```

```
if(m_ctlSlider.GetPos()>4 && m_ctlSlider.GetPos()<=5){  
    Output(0x378,16);
```

```
}
```

```
//***** PIN 6
```

```
*****//
```

```
if(m_ctlSlider.GetPos()>5 && m_ctlSlider.GetPos()<=6){  
    Output(0x378,32);
```

```
}
```

```
//***** PIN 7
```

```
*****//
```

```
if(m_ctlSlider.GetPos()>6 && m_ctlSlider.GetPos()<=7){  
    Output(0x378,64);
```

```
}
```

```
//***** PIN 8
```

```
*****//
```

```
if(m_ctlSlider.GetPos()>7){  
    Output(0x378,128);
```

```
}
```

```
}
```

```
}
```

APPENDIX 2: SERVER SCRIPT

```
// SockDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Sock.h"
#include "SockDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
   //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    }}AFX_DATA

    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CAboutDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    }}AFX_VIRTUAL

// Implementation
protected:
   //{{AFX_MSG(CAboutDlg)
    }}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
   //{{AFX_DATA_INIT(CAboutDlg)
    }}AFX_DATA_INIT
}
```



```

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

```

```

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
    // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

```

```

////////////////////////////////////
// CSockDlg dialog

```

```

CSockDlg::CSockDlg(CWnd* pParent /*=NULL*/)
: CDialog(CSockDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CSockDlg)
    m_strMessage = _T("");
    m_strName = _T("");
    m_iPort = 0;
    m_iType = -1;
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

```

```

void CSockDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CSockDlg)
    DDX_Control(pDX, IDC_SLIDER, m_ctlSlider);
    DDX_Control(pDX, IDC_LSENT, m_ctlSent);
    DDX_Control(pDX, IDC_LRECV, m_ctlRecv);
    DDX_Control(pDX, IDC_BCONNECT, m_ctlConnect);
    DDX_Text(pDX, IDC_EMSG, m_strMessage);
    DDX_Text(pDX, IDC_ESERVNAME, m_strName);
    DDX_Text(pDX, IDC_ESERVPORT, m_iPort);
    DDX_Radio(pDX, IDC_RCLIENT, m_iType);
    //}}AFX_DATA_MAP
}

```

```

BEGIN_MESSAGE_MAP(CSockDlg, CDialog)
//{{AFX_MSG_MAP(CSockDlg)
ON_WM_SYSCOMMAND()
ON_WM_PAINT()
ON_WM_QUERYDRAGICON()
ON_BN_CLICKED(IDC_RCLIENT, OnRType)
ON_BN_CLICKED(IDC_BCONNECT, OnBconnect)
ON_BN_CLICKED(IDC_BSEND, OnBsend)
ON_BN_CLICKED(IDC_RSERVER, OnRType)
ON_BN_CLICKED(IDC_BCLOSE, OnBclose)
ON_WM_HSCROLL()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

```

```

////////////////////////////////////
// CSockDlg message handlers

```

```

BOOL CSockDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
        }
    }

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

```



```

// TODO: Add extra initialization here
// Initialize the control variables
m_iType = 0;
m_strName = "loopback";
m_iPort = 4000;
// Update the controls
UpdateData(FALSE);
//GetDlgItem(IDC_SLIDER)->EnableWindow(TRUE);
// Set the socket dialog pointers
m_sConnectSocket.SetParent(this);
m_sListenSocket.SetParent(this);

return TRUE; // return TRUE unless you set the focus to a control
}

```

```

void CSockDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

```

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon. For MFC applications using the document/view model,
// this is automatically done for you by the framework.

```

void CSockDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (LPARAM)
dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
    }
}

```



```
int x = (rect.Width() - cxIcon + 1) / 2;  
int y = (rect.Height() - cyIcon + 1) / 2;
```

```
// Draw the icon  
dc.DrawIcon(x, y, m_hIcon);
```

```
}  
else
```

```
{  
    CDialog::OnPaint();  
}
```

```
// The system calls this to obtain the cursor to display while the user drags  
// the minimized window.
```

```
HCURSOR CSockDlg::OnQueryDragIcon()
```

```
{  
    return (HCURSOR) m_hIcon;  
}
```

```
void CSockDlg::OnRType()
```

```
{  
    // TODO: Add your control notification handler code here  
    // Sync the controls with the variables  
    UpdateData(TRUE);  
    // Which mode are we in?  
    if (m_iType == 0){ // Set the appropriate text on the button  
        m_ctlConnect.SetWindowText("C&onnect");  
    }  
    else{  
        m_ctlConnect.SetWindowText("&Listen");  
    }  
}
```

```
void CSockDlg::OnBconnect()
```

```
{  
    // TODO: Add your control notification handler code here  
    // Sync the variables with the controls  
    UpdateData(TRUE);  
    // Disable the connection and type controls  
    GetDlgItem(IDC_BCONNECT)->EnableWindow(FALSE);  
    GetDlgItem(IDC_ESERVNAME)->EnableWindow(FALSE);  
    GetDlgItem(IDC_ESERVPORT)->EnableWindow(FALSE);  
    GetDlgItem(IDC_STATICNAME)->EnableWindow(FALSE);  
    GetDlgItem(IDC_STATICPORT)->EnableWindow(FALSE);  
}
```

```

GetDlgItem(IDC_RCLIENT)->EnableWindow(FALSE);
GetDlgItem(IDC_RSERVER)->EnableWindow(FALSE);
GetDlgItem(IDC_STATICTYPE)->EnableWindow(FALSE);

```

```

// Are we running as client or server?

```

```

if (m_iType == 0){
    GetDlgItem(IDC_SLIDER)->EnableWindow(TRUE);
    GetDlgItem(IDC_LABELMODE)->EnableWindow(TRUE);
    GetDlgItem(IDC_LABELMODE)->SetWindowText("Client Mode");
    // Client, create a default socket
    m_sConnectSocket.Create();
    // Open the connection to the server
    m_sConnectSocket.Connect(m_strName, m_iPort);

    m_ctlSlider.SetRange(0, 10);
}
else{
    GetDlgItem(IDC_SLIDER)->EnableWindow(TRUE);
    GetDlgItem(IDC_LABELMODE)->EnableWindow(TRUE);
    GetDlgItem(IDC_LABELMODE)->SetWindowText("Server Mode");
    // Server, create a socket bound to the port specified
    m_sListenSocket.Create(m_iPort);
    // Listen for connection requests
    m_sListenSocket.Listen();
    m_ctlSlider.SetRange(0, 8);
}

```

```

//*****

```

```

*//

```

```

typedef UINT (CALLBACK* LPFNDLLFUNC1)(INT,INT);
typedef UINT (CALLBACK* LPFNDLLFUNC2)(INT);
HINSTANCE hDLL; // Handle to DLL
LPFNDLLFUNC1 Output; // Function pointer
LPFNDLLFUNC2 Input; // Function pointer
INT Addr;
INT AddrIn;
INT Value;
hDLL = LoadLibrary("Inpout32");
if (hDLL != NULL)
{
    Output = (LPFNDLLFUNC1)GetProcAddress(hDLL,"Out32");
    Input = (LPFNDLLFUNC2)GetProcAddress(hDLL,"Inp32");
    if (!Output || !Input)
    {
        // handle the error FreeLibrary(hDLL);
    }
}

```



```

    }
    Addr = 0x378;
    AddrIn = 0x379;
    Value = 0;
    Output(Addr, Value);
    INT somenum = Input(Addr);

```

```

//*****

```

```

*/

```

```

}

```

```

}

```

```

void CSockDlg::OnAccept()

```

```

{

```

```

    // Accept the connection request
    m_sListenSocket.Accept(m_sConnectSocket);
    // Enable the text and message controls
    GetDlgItem(IDC_EMSG)->EnableWindow(TRUE);
    GetDlgItem(IDC_BSEND)->EnableWindow(TRUE);
    GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);

```

```

}

```

```

void CSockDlg::OnConnect()

```

```

{

```

```

    // Enable the text and message controls
    GetDlgItem(IDC_EMSG)->EnableWindow(TRUE);
    GetDlgItem(IDC_BSEND)->EnableWindow(TRUE);
    GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
    GetDlgItem(IDC_BCLOSE)->EnableWindow(TRUE);

```

```

}

```

```

void CSockDlg::OnSend()

```

```

{

```

```

}

```

```

void CSockDlg::OnReceive()

```

```

{

```

```

    char *pBuf = new char[1025];
    int iBufSize = 1024;
    int iRcvd;
    CString strRcvd;

```

```

    // From client site controlling

```



```

typedef UINT (CALLBACK* LPFNDLLFUNC1)(INT,INT);
typedef UINT (CALLBACK* LPFNDLLFUNC2)(INT);
HINSTANCE hDLL; // Handle to DLL
LPFNDLLFUNC1 Output; // Function pointer
LPFNDLLFUNC2 Input; // Function pointer
INT Addr;
INT AddrIn;
INT Value;
hDLL = LoadLibrary("Inpout32");
if (hDLL != NULL)
{
    Output = (LPFNDLLFUNC1)GetProcAddress(hDLL,"Out32");
    Input = (LPFNDLLFUNC2)GetProcAddress(hDLL,"Inp32");
    if (!Output || !Input)
    {
        // handle the error FreeLibrary(hDLL);
    }
}
Addr = 0x378;
AddrIn = 0x379;
Value = 0;
//Output(Addr, Value);
INT somenum = Input(Addr);

// Receive the message
iRcvd = m_sConnectSocket.Receive(pBuf, iBufSize);
// Did we receive anything?
if (iRcvd == SOCKET_ERROR){
}
else{
    pBuf[iRcvd] = NULL;
    //if (pBuf==int){
    //    GetDlgItem(IDC_VALUE)->SetWindowText(pBuf);
    //}
    //else{

    // Copy the message to a CString
    strRcvd = pBuf;
    //i=0;
    if(strRcvd=="000" || strRcvd=="000000"){
        GetDlgItem(IDC_VALUE)->SetWindowText("0");
        //i=1;
        //m_ctlRcvd.AddString(strRcvd);
        m_ctlSlider.SetPos(0);
    }
}

```

```

        Output(0x378,0);
    }
    else if(strRecvd=="001" || strRecvd=="001001"){
        //m_ctlRecvd.AddString(strRecvd);
        GetDlgItem(IDC_VALUE)->SetWindowText("1");
        //i=1;
        m_ctlSlider.SetPos(1);
        Output(0x378,1);
    }
    else if(strRecvd=="002" || strRecvd=="002002"){
        //m_ctlRecvd.AddString(strRecvd);
        GetDlgItem(IDC_VALUE)->SetWindowText("2");
        //i=1;
        m_ctlSlider.SetPos(2);
        Output(0x378,2);
    }
    else if(strRecvd=="003" || strRecvd=="003003"){
        //m_ctlRecvd.AddString(strRecvd);
        GetDlgItem(IDC_VALUE)->SetWindowText("3");
        //i=1;
        m_ctlSlider.SetPos(3);
        Output(0x378,4);
    }
    else if(strRecvd=="004" || strRecvd=="004004"){
        //m_ctlRecvd.AddString(strRecvd);
        GetDlgItem(IDC_VALUE)->SetWindowText("4");
        //i=1;
        m_ctlSlider.SetPos(4);
        Output(0x378,8);
    }
    else if(strRecvd=="005" || strRecvd=="005005"){
        //m_ctlRecvd.AddString(strRecvd);
        GetDlgItem(IDC_VALUE)->SetWindowText("5");
        //i=1;
        m_ctlSlider.SetPos(5);
        Output(0x378,16);
    }
    else if(strRecvd=="006" || strRecvd=="006006"){
        //m_ctlRecvd.AddString(strRecvd);
        GetDlgItem(IDC_VALUE)->SetWindowText("6");
        //i=1;
        m_ctlSlider.SetPos(6);
        Output(0x378,32);
    }
    else if(strRecvd=="007" || strRecvd=="007007"){
        //m_ctlRecvd.AddString(strRecvd);

```



```

        GetDlgItem(IDC_VALUE)->SetWindowText("7");
        //i=1;
        m_ctlSlider.SetPos(7);
        Output(0x378,64);
    }
    else if(strRecvd=="008" || strRecvd=="008008"){
        //m_ctlRecv.AddString(strRecvd);
        GetDlgItem(IDC_VALUE)->SetWindowText("8");
        //i=1;
        m_ctlSlider.SetPos(8);
        Output(0x378,128);
    }
    else if(strRecvd=="009" || strRecvd=="009009"){
        //m_ctlRecv.AddString(strRecvd);
        GetDlgItem(IDC_VALUE)->SetWindowText("9");
        //i=1;
        m_ctlSlider.SetPos(9);
        Output(0x378,128);
    }
    else if(strRecvd=="010" || strRecvd=="010010"){
        //m_ctlRecv.AddString(strRecvd);
        GetDlgItem(IDC_VALUE)->SetWindowText("10");
        //i=1;
        m_ctlSlider.SetPos(10);
        Output(0x378,128);
    }
    else if(strRecvd=="6623"){
        //MessageBox("Client Disconnected");
        m_ctlRecv.AddString("**** Client Disconnected ****");
    }
    else{
        // Add the message to the received list box
        m_ctlRecv.AddString(strRecvd);
        //GetDlgItem(IDC_VALUE)->SetWindowText(strRecvd);
        // Sync the variables with the controls
    }
    UpdateData(TRUE);
}
}

```

```

void CSockDlg::OnClose()
{
    // Close the connected socket
    m_sConnectSocket.Close();
}

```



```

//Disable the message sending controls
GetDlgItem(IDC_EMSG)->EnableWindow(FALSE);
GetDlgItem(IDC_BSEND)->EnableWindow(FALSE);
GetDlgItem(IDC_STATICMSG)->EnableWindow(FALSE);
GetDlgItem(IDC_BCLOSE)->EnableWindow(FALSE);

// Are we running in Client mode?
if (m_iType == 0){
    // Yes, so enable the connection configuration controls
    GetDlgItem(IDC_BCONNECT)->EnableWindow(TRUE);
    GetDlgItem(IDC_ESERVNAME)->EnableWindow(TRUE);
    GetDlgItem(IDC_ESERVPORT)->EnableWindow(TRUE);
    GetDlgItem(IDC_STATICNAME)->EnableWindow(TRUE);
    GetDlgItem(IDC_STATICPORT)->EnableWindow(TRUE);
    GetDlgItem(IDC_RCLIENT)->EnableWindow(TRUE);
    GetDlgItem(IDC_RSERVER)->EnableWindow(TRUE);
    GetDlgItem(IDC_STATICTYPE)->EnableWindow(TRUE);
    GetDlgItem(IDC_LABELMODE)->EnableWindow(FALSE);
    GetDlgItem(IDC_LABELMODE)->SetWindowText("");
}
}

```

```

void CSockDlg::OnBsend()
{
    // TODO: Add your control notification handler code here
    int iLen;
    int iSent;

    // Sync the controls with the variables
    UpdateData(TRUE);
    // Is there a message to be sent?
    if(m_strMessage != ""){
        // Get the length of the message
        iLen = m_strMessage.GetLength();

        // Send the message
        iSent = m_sConnectSocket.Send(LPCTSTR(m_strMessage), iLen);
        // Were we able to send it?
        if(iSent == SOCKET_ERROR){
        }
        else{
            // Add the message to the list box.
            m_ctlSent.AddString(m_strMessage);
            GetDlgItem(IDC_EMSG)->SetWindowText("");
            // Sync the variables with the controls
            UpdateData(TRUE);
        }
    }
}

```

```
}
```

```
}
```

```
void CSockDlg::OnBclose()
```

```
{
```

```
// TODO: Add your control notification handler code here
```

```
m_sConnectSocket.Send(LPCTSTR("6623"), 4);
```

```
// Call the OnClose function
```

```
OnClose();
```

```
}
```

```
void CSockDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
```

```
{
```

```
CString pos;
```

```
pos.Format("%d", m_ctlSlider.GetPos());
```

```
GetDlgItem(IDC_VALUE)->SetWindowText(pos);
```

```
//if (m_iType == 0){
```

```
if(pos == "0")
```

```
    m_sConnectSocket.Send(LPCTSTR("000"),3);
```

```
if(pos == "1")
```

```
    m_sConnectSocket.Send(LPCTSTR("001"),3);
```

```
if(pos == "2")
```

```
    m_sConnectSocket.Send(LPCTSTR("002"),3);
```

```
if(pos == "3")
```

```
    m_sConnectSocket.Send(LPCTSTR("003"),3);
```

```
if(pos == "4")
```

```
    m_sConnectSocket.Send(LPCTSTR("004"),3);
```

```
if(pos == "5")
```

```
    m_sConnectSocket.Send(LPCTSTR("005"),3);
```

```
if(pos == "6")
```

```
    m_sConnectSocket.Send(LPCTSTR("006"),3);
```

```
if(pos == "7")
```

```
    m_sConnectSocket.Send(LPCTSTR("007"),3);
```

```
if(pos == "8")
```

```
    m_sConnectSocket.Send(LPCTSTR("008"),3);
```

```
if(pos == "9")
```

```
    m_sConnectSocket.Send(LPCTSTR("009"),3);
```

```
if(pos == "10")
```

```
    m_sConnectSocket.Send(LPCTSTR("010"),3);
```

```
//}
```

```
if (m_iType == 1){
```

```
// For server site controlling
```



```

typedef UINT (CALLBACK* LPFNDLLFUNC1)(INT,INT);
typedef UINT (CALLBACK* LPFNDLLFUNC2)(INT);
HINSTANCE hDLL; // Handle to DLL
LPFNDLLFUNC1 Output; // Function pointer
LPFNDLLFUNC2 Input; // Function pointer
INT Addr;
INT AddrIn;
INT Value;
hDLL = LoadLibrary("Inpout32");
if (hDLL != NULL)
{
    Output = (LPFNDLLFUNC1)GetProcAddress(hDLL,"Out32");
    Input = (LPFNDLLFUNC2)GetProcAddress(hDLL,"Inp32");
    if (!Output || !Input)
    {
        // handle the error FreeLibrary(hDLL);
    }
}
Addr = 0x378;
AddrIn = 0x379;
Value = 0;
//Output(Addr, Value);
INT somenum = Input(Addr);

if(m_ctlSlider.GetPos()==0){
    Output(0x378,0);
}

//***** PIN 1
*****//
if(m_ctlSlider.GetPos()>0 && m_ctlSlider.GetPos()<=1){
    Output(0x378,1);
}

//***** PIN 2
*****//
if(m_ctlSlider.GetPos()>1 && m_ctlSlider.GetPos()<=2){
    Output(0x378,2);
}

//***** PIN 3
*****//
if(m_ctlSlider.GetPos()>2 && m_ctlSlider.GetPos()<=3){
    Output(0x378,4);
}

```



```

}

//***** PIN 4
*****//
if(m_ctlSlider.GetPos()>3 && m_ctlSlider.GetPos()<=4){
    Output(0x378,8);
}

//***** PIN 5
*****//
if(m_ctlSlider.GetPos()>4 && m_ctlSlider.GetPos()<=5){
    Output(0x378,16);
}

//***** PIN 6
*****//
if(m_ctlSlider.GetPos()>5 && m_ctlSlider.GetPos()<=6){
    Output(0x378,32);
}

//***** PIN 7
*****//
if(m_ctlSlider.GetPos()>6 && m_ctlSlider.GetPos()<=7){
    Output(0x378,64);
}

//***** PIN 8
*****//
if(m_ctlSlider.GetPos()>7){
    Output(0x378,128);
}

}

```

APPENDIX 3: USER MANUAL

This user manual will explain on how to set up the Smart Home Prototype System. Before going into the installation details, let's look at the checklist for the component of the system. This system contains:

- CD containing the software program
- Hardware Interface Package
- Audio Amplifier

Setting Up The Software

The following steps show how to install and run the software programs:

1. Insert the CD into the CD-Rom. Open the folder named DLL and copy all files into the Windows System32 directory.
2. Open folder SmartHome Prototype and copy the SmartHome.exe file into your desktop. If you wish to run the programs remotely, repeat step 1 and 2 on another computer.
3. Double click the SmartHome.exe and the program should run properly.
4. If running on Server Mode, enter the port number. Default is set to 4000. Click Listen to run the program.
5. If running on Client Mode, make sure to enter either the destination computer's ID or the IP address. Next, enter the port address. Default is set to 4000. Click Connect to run the program.

6. Now you are ready to communicate between the Server program and the Client program. Enter any text messages on the textbox and click Send to send to the remote program. Text messages received will be displayed on the receive listbox.

Setting Up The Hardware

The following steps explain how to set up the hardware:

1. Connect the hardware interface port to the server computer's parallel port by using a male-to male parallel port cable.
2. Plug in a DC power supply into the DC jack. No setting on polarity is needed.
 - The recommended DC power source is 18V. Switch on the DC power supply.
3. Connect the volume socket from the hardware interface package to the audio amplifier volume socket.
4. Next connect from an audio source to the amplifier by connecting into the RCA socket.
5. Finally, connect a headphone or a multimedia speaker into the headphone socket on the amplifier.
6. Turn on the switch on the audio amplifier.
7. Now the hardware interface and the audio amplifier are ready to run.

Controlling The Volume Using the Client Program

From the client program, control the volume on the audio amplifier by sliding the slider on the client program. The level of the volume can be seen on the value changes on the program. The server program also has the slider build in for locally controlling the volume of the audio amplifier.

Closing the Program

To close the Client Program, simply click Close to disconnect from the Server Program. Then click OK to close down the Client Program.

To close the Server Program, simply click OK to close down the program.