# Combiner Box For Multiple Telephone Systems


Final Year project UG 2020

By

Capt Danyal Asad

Capt Ahmad Ali

Capt Arsalan Ali

Capt Usama Mustafa

Caapt Waqar Aslam

Supervisor

Asst. Prof Dr Abdul Wakeel

Submitted to the faculty of Department of Electrical Engineering, Military College of Signals, National University of Sciences and Technology, in partial fulfillment for the requirements of B.E Degree in Electrical Engineering

July 2020

# Combiner Box For Multiple Telephone Systems

By

Capt Danyal Asad

Capt Ahmad Ali

Capt Arsalan Ali

Capt Usama Mustafa

Caapt Waqar Aslam

Supervisor

Asst. Prof Dr Abdul Wakeel

Submitted to the faculty of Department of Electrical Engineering, Military College of Signals, National University of Sciences and Technology, in partial fulfillment for the requirements of B.E Degree in Electrical Engineering

July 2020

## CERTIFICATE OF CORRECTIONS & APPROVAL

Certified that work contained in this thesis titled "Combiner Box for multiple Telephone Systems", carried out by 1) Capt Danyal Asad 2) Capt Ahmad Ali 3) Capt Arsalan Ali 4) Capt Usama Mustafa 5) Capt Waqar Aslam under the supervision of Asst. Prof Dr Abdul Wakeel for partial fulfillment of Degree of Bachelors of Electrical Engineering, in Military College of Signals, National University of Sciences and Technology, Islamabad during the academic year 2019-2020 is correct and approved. The material that has been used from other sources it has been properly acknowledged / referred.

**Approved by**

**Supervisor**

**Asst. Prof Dr Abdul Wakeel**

Date: _____

## DECLARATION OF ORIGINALITY

**We hereby declare that no content and variety of work bestowed during this thesis has been submitted in support of another award of qualification or degree either during this course or anyplace else.**

.

## PLAGIARISM CERTIFICATE (TURNITIN REPORT)

This thesis has been checked for Plagiarism. Turnitin report endorsed by Supervisor is attached.

Capt Danyal Asad
00000199785

Prof Dr Abdul Wakeel
Supervisor

## ACKNOWLEDGEMENTS

We are thankful to our Creator Allah Subhana-Watala to have guided us throughout this work at every step and for every new thought which He setup in our mind to improve it. Indeed we could have done nothing without His priceless help and guidance. Whosoever helped us throughout the course of our thesis, whether my parents or any other individual was His will, so indeed none be worthy of praise but Him.

We are profusely thankful to our beloved parents who raised us when I we were not capable of walking and continued to support us throughout in every department of our life.

We would also like to express special thanks to our supervisor Asst. Prof Dr Abdul Wakeeel for his help throughout our thesis and also for Analog and Digital Communication Systems courses which he has taught us. We can safely say that we haven't learned any other engineering subject in such depth than the ones which he has taught.

We would also like to pay special thanks to the department of EE for its tremendous support and cooperation.

Finally, we would like to express our gratitude to all the individuals who have rendered valuable assistance to our study.

*Dedicated to our parents and loving families whose tremendous support and cooperation led us to this wonderful accomplishment.*

## ABSTRACT

This report presents a new product for use within a multi networked PSTN environment. The telephone combiner box is designed to create an interface and provide a replacement to multiple subscriber end telephone sets within a multi networked environment. The hardware is based on legacy analog telephony standards in a bid to provide security in form of network isolation and independence between different telephone networks. It merely provides an interface to subscribers as opposed to integrating several lines onto one using multiplexing. The project is intended to provide a replacement to multiple telephones in a low or mid tier office and may also provide a cost effective alternative to digital PABX which are installed for the purpose of integrating several lines. The combiner box is able to facilitate an office environment with multi-network or same network subscriber terminals. The design is modular and can be increased subject to restrictions placed by the controller input and output.



**Keywords:-** *PSTN, PASCOM, DEFCOMM, NTC, PATCOMS, NG PATCOMS, Analog Telephony , automated switching, network isolation, python, raspberry pi*

# TABLE OF CONTENTS

## LIST OF FIGURES

# Chapter 1 Introduction

## 1.1 Background

Military communication has long played a pivotal role in security and warfare. The transmission of information between all units plays a crucial part in exercising commands and subsequently having a unanimous front against the enemy. Often in result military communication has to be multi-tiered and multi networked. Pakistan Army is no such exception.

The army has long had multiple telecommunication systems (PSTN) running parallel over several tiers of operation. Namely PASCOM, DEFCOMM, NTC. With the advent of mil Operations in IS environment an integration with civilian infrastructure such as PTCL was also seen. Extension of PASCOMS in the field by use of PATCOMS and NG PATCOMS has increased the complexity of the army's networked environment.

## 1.2 Problem Statement

To design an intelligent combiner with the ability to Interface multiple telephone subscriber lines onto single telephone unit without the use of any external private branch exchange

## 1.3 Project Description

The telephone combiner box aims to reduce the complexity of multi networked environment by interfacing subscriber numbers of the multiple networks onto one subscriber telephone therefore reducing cost, and increasing efficiency of the communication environment.

Keeping in view the sensitivity of the networks aspects of security and network, isolation were also required to be incorporated which required an innovative approach based on legacy electronics whilst still managing to accomplish intelligent tasks such as automated switching. The combiner box is able to facilitate an office environment with multi-network or same network subscriber terminals.

The design is modular and can be increased subject to restrictions placed by the controller input and output. Finally the combiner box makes use of software programming of python and

Raspberry pi as its microcontroller providing more avenues for a software based easily customizable interface.

## 1.4 Prospective Application Areas

Combiner Telephone box for multiple telephone subscribers to include Pakistan Army Strategic Communication (PASCOMM), Defensive communication (DEFCOMM), PTCL, NTC telephones and intercom for use in Pakistan Army. It will eliminate the use of multiple telephones for single user. However, will not allow interconnection of networks by allowing cross network communication

## 1.5 Scope Objectives, Specification and Deliverables

### 1.5.1 Scope and Objectives

For this project it was endeavored to design an intelligent combiner with the ability to Interface multiple telephone subscriber lines onto single telephone unit without the use of any external private branch exchange. The scope is limited to Local PSTN Loop from Central Office to the Subscriber, it will include Call Ring Detection, and switching onto the subscriber

The scope of this project was based on an understanding of different protocols involved in analog and digital telephony. To develop a comprehensive understanding of PSTN networks, subscriber line format and protocols and transmission and switching of voice over the subscriber local loop. It entailed to achieve the following:

1. An alternative method of integrating multiple phone lines from different networks onto one telephone subscriber
2. The use of an intermediary device to switch between telephone lines without integrating them through multiplexing or any other technique
3. Design a device that is modular and expandable
4. Design an automated switching algorithm based on multiple lines
5. Keeping the subscriber phone Independent of switching and interfacing operations by providing a graphical interface for device control
6. Use of Raspberry Pi as a micro controller in the project

7. Following objectives were achieved:
8. Successfully selecting and switching between lines upon input from user
9. Automatic connection to the telephone upon incoming call on any line
10. Call Waiting facility for 30s followed by automatic termination if multiple calls received
11. Ability to switch between calls and place calls on hold for a limited period

### 1.5.2 **Specifications**

1. 4x Analog Subscriber Ports
2. Call Waiting Facility
3. Automatic Call Answering/ Line Switching
4. Fully Automated
5. Python GUI

### 1.5.3 **Deliverables**

1. Combiner Box with 4x Subscriber Input Ports and 1x Output Port
2. LCD touchscreen attached to Module
3. Raspberry Pi 4 as control
4. ISD 1820 audio Modules

# Chapter 2 Literature Review

## 2.1 PSTN Networks

PSTN stands for Public Switched Telephone Network and it is a traditionally setup circuit switched telephone network being used since the 1800's. It uses underground copper wires which provide for a reliable communication means in offices and households alike. They are the bricks to a standard form of communication but have been on a decline this past century due to the phenomenon of the mobile phone.

PSTN comprises of a multitude of networks including switching centres, fibre optic lines, cellular networks, telephone lines, cable systems and satellites all in one. All of these components make communication easier. The general flow of information during a call occurs through these networks by conversion and transmission of sound waves into electrical signals, which is expedited due to fibre optics cables and global switching centres, and converted back into sound waves when the route is complete. [1]

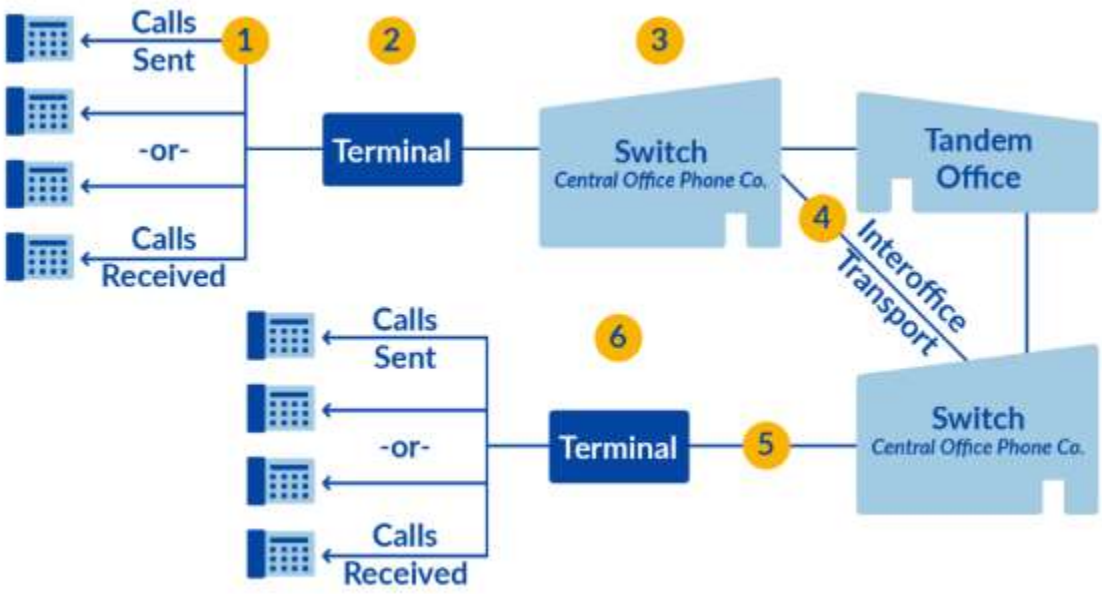



*Figure 1. Plain Old Telephone Service Networking in the form of flowchart. [1]*

### 2.1.1 Switching in PSTN

The backbone of traditional phone networks is mostly about switching. In PSTN's when a call is made, switches create a wire circuit between the two telephones and this connection lasts until the call ends. There are four types of switching taking place at different levels.[2]

### 2.1.2 The Local Exchange

The local exchange, which may consist of more than one exchange, links the user to a PSTN line. This centre may consist of more than a 10,000 lines. The exchange routes the call to its correct destination.[2]

### 2.1.3 The Tandem Office

It is also known as a junction unit, and comprises of multiple local exchanges covering a large geographical area. The switches between local exchanges are managed here.

### 2.1.4 The Toll Office

National long distance switching occurs at a toll office. All the tandem offices are linked by a single toll office.

### 2.1.5 The International Gateway

International call switching is handled by the international gateway. All the domestic calls are routed through here to there particular country destinations.[2]

## 2.2 Local Loop

Local loop is basically the physical wiring that connects the user to the Public Switched Telephone Network or PSTN. It can either be in the form of a data line or a voice line. It comprises of two copper wires twisted together that run from the central office to the users premise and then another pair of copper wires that run back to the central office to the user. The local loop may run a distance of 1 km to 10 km.

These days, the user's phone is not directly linked to the local but most houses now have a grey small box outside called the Network Interface Device or NID. This gray box connects the inner wiring of the house to the local loop wiring which is outside. An analog electrical signal is passed between this duplex of wires all the way to the central office. It is called the local loop because the connection is local to the user's premise.

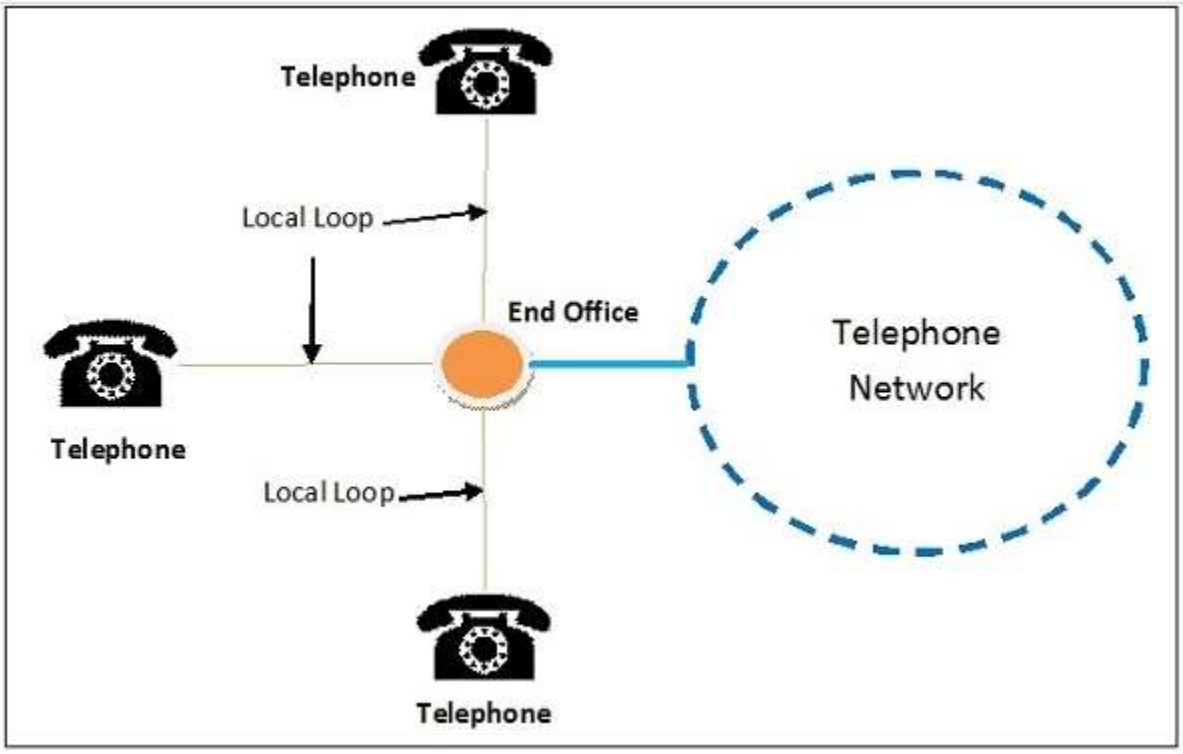



*Figure 2. local loop [3]*

A telephone set has a microphone which converts the sound of our voice into electrical pulses which in combination form an analog signal. This signal then travels from wiring in the house to the NID and consequently to the local loop. The signal is then sent to the central office where it is converted into digital data by devices that sample, quantize and encode information in the analog signal. This digital data is time division multiplexed (It is the process whereby communication time slots are divided into smaller time slots) over trunk lines.

The traditional local loop has several limitations such as crosstalk, narrow bandwidth, distortion of symbols and high attenuation. Currently, the copper wiring is being replaced by the fibre optic cables for a more accurate and faster performance. Installation of these fibre optic cables is also known as FttH (Fibre to the Home). [3]

## 2.3 Signalling and Switching Protocols

Signalling basically refers to the language used by the Network Elements (NE) to exchange information. Complex digital messages are exchanged between the Network Elements in a signalling system.

Signalling systems are the vital building blocks to a standard and efficient worldwide telecommunication. In data communication, signalling systems act to exchange signalling information accurately between subscribers. Introduction of signalling systems brought about a revolution in the efficiency of the PSTN. There are two basic types of signalling; in band signalling and common channel signalling. Voice information and signalling information travel on common paths in in band signalling, whereas they travel on separate paths in common channel signalling. These are further divided into few types depending upon the frequencies and frequency techniques used. [4]

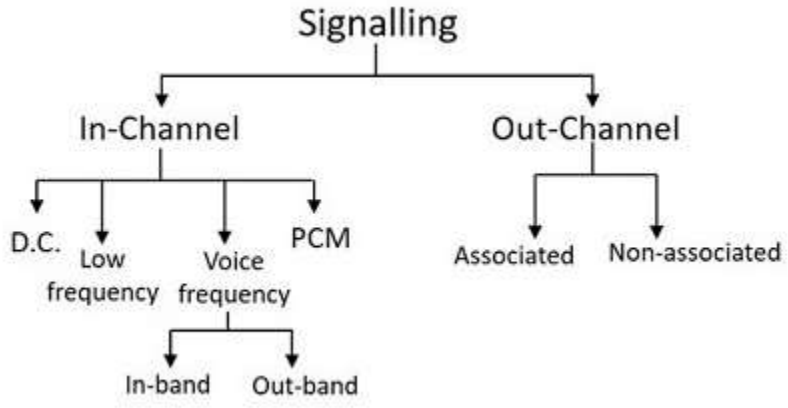



*Figure 3. Types of Signalling [4]*

Signalling systems are differentiated by their principle characteristics into six major categories.

### 2.3.1 In band signalling

Instead of a common voice signal/channel, in band signalling uses tones to relay information. In this form of signalling of PSTN, the call control information uses the same channel that the telephone line itself is using. Dual tone Multi frequency Signalling or DMTF is a type of in band signalling.[5]

### 2.3.2 **Out of band signalling**

Out of band signalling forms the basis of telecommunication, as it is dedicated for the purpose and separate from the channels used for the telephone call. The major advantage of this type of signalling is that continuous supervision is required whether the tone is of or on during the entire time of the phone calls. An example of out-of-band signalling is the ITU-T R-2 System.[5]

### 2.3.3 **Line signalling**

Line signalling comes under the class of various signalling protocols. The network-wide signalling that involves end-to-end signalling between the originating exchange and the terminating exchange is called the Line signalling.

It relates to a two party phone call attempt and if that is positive then the establishment of a phone call. Line signalling used to be referred to electrical pulses generated over two wire or four wire circuit, but later these analogue electrical pulses were replaced by digital signals by a DS0 channel trunk in the 1970's.

In a simple call, the voice transmission occurs by signalling techniques which fall into three categories; supervision, alerting or addressing. Supervision persists of the changes made in the loop when a call is initiated and addressing includes the actual dialling of the numbers and the pulses generated which are sent to a central office. This loop start signalling which is initiated by a telephone call includes five stages which are explained below.[8]

#### 2.3.3.1. On hook

When the telephone is in its handset, meaning it is ready to receive a call, it is in an on hook position.


*Figure 4. on hook configuration [5]*

### 2.3.3.2. Off hook

When the user decides to pick up the phone from the handset to make a phone call, it is called to be in an off hook state.[

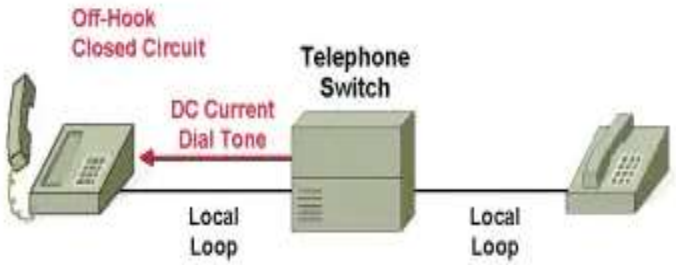



*Figure 5. off hook configuration [5]*

### 2.3.3.3. Dialling

The dialling phase lets the user make a phone call by the dialling of the digits either using a rotary dial that generates pulses or the button dialling system.

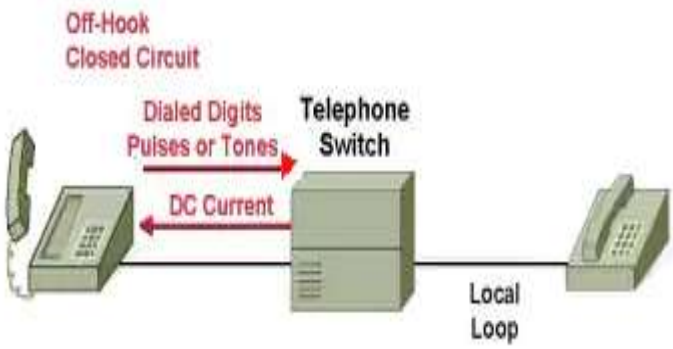



*Figure 6. Dialling configuration [5]*

### 2.3.3.4. Switching

In the switching stage, the central office converts the dialled pulses generated into an address that connects the user to the called person.

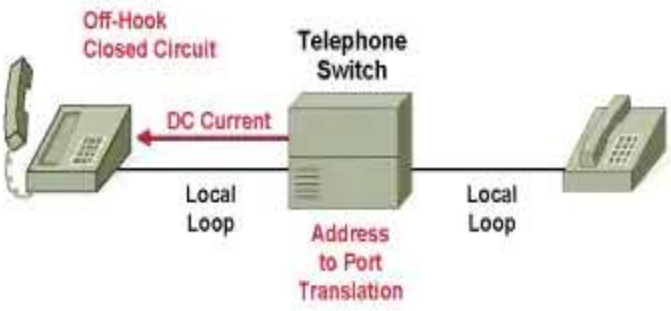



*Figure 7. Switching configuration [5]*

### 2.3.3.5. Ringing

When the central office successfully connects user to the called party, the user can hear a ringing in the telephone line signalling them that the call is being made.

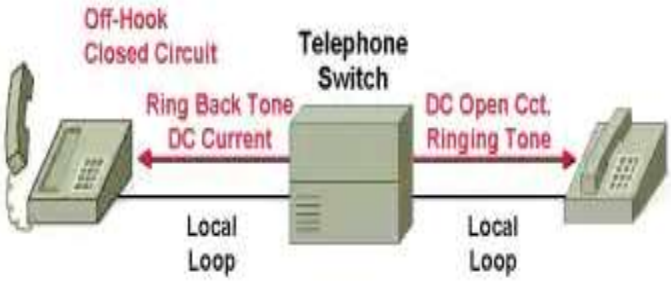



*Figure 8. Ringing configuration[5]*

Line signalling is primarily deals with relaying information on the state of the channel or line, as in hook flash, ringing or off hook and on hook which are collectively called supervision.

| Line State | AC Voltage | DC Voltage | Line Resistance |
|---|---|---|---|
| **Off Hook** | | Ring -7v to -12v<br>Tip Gnd | 600Ω (AC Line Impedance) 200Ω (DC Resistance) |
| **On Hook** | | Ring -48V<br>Tip Gnd | Few MΩ |
| **Ringing** | 120v @ 20Hz 1s on 1s Off | Ring -48V<br>Tip Gnd | |

*Table 1. Summary Line specifications*

### 2.3.4 **Register signalling**

Register signalling is contrasted to Line signalling as it provides addressing information such as the caller ID or their phone number. In the early days, this was the task of the operator who would be vocally told to dial so and so's number. In the 20$^{th}$ century, a rotary dial was introduced which handled this task. It would rapidly break the line current into small pulses, whereby the number of pulses determined the telephone number being dialled. An example of Register signalling is the R2 register. [5]

### 2.3.5 **Channel associated signalling**

Channel associated signalling is also called per trunk signalling, it basically routes the payload of voice or data information to its specific destination. In channel associated signalling the control signals which bound and synchronize frames, are relayed in the same channels as in data signals or voice signals. An example of channel associated signalling is the T carrier system. [5]

### 2.3.6 **Common channel signalling**

In common channel signalling the combined voice and data channels share a unique and separate channel for control signals. This usually controls a multitude of message channels. [5]

## 2.4 **Analog & Digital Telephony**

Analog telephony also known as plain old telephone service or POTS uses a standard narrow RJ-11 plug for one's telephone. It translates analog signals i.e. voice into electronic pulses. These are the most common type of phones used in households. Although this technology has the disadvantage in regards to the amount of data that it can transfer.

Digital telephone has the advantage that it can transfer large amounts of data, therefore it can be used in businesses. Digital telephone systems also have better voice quality over long distances. This technology converts audio signals into digits first and then decodes those signals back into audio at the other end of the phone call. [6]

**Analog and digital comparison**

| | Advantage | Disadvantage |
|---|---|---|
| Analog | Low and easy maintenance | Cannot be scaled up or down easily |
| | Simple to use | Bandwidth is not fully utilized |
| | Minimal setup | Expensive |
| Digital | Increased effectiveness of call routing and transmission | Emergency numbers are not widely available and emergency responders will not be able to pinpoint the location of the call |
| | Lower monthly cost | Security threats because it is connected to the Internet |
| | Scalable | Power outages and problems with Internet connection will bring down connections |
| | Data and video transmission capability | |
| | You can use your PC or laptop as a softphone | |
| | Small businesses can now enjoy the features that only large businesses are able to afford | |

*Table 2. Comparative analysis of analog and digital telephone systems [6]*

## 2.5 **Multiline Telephone Systems**

A multiline telephone system allows a multitude of users to be connected on a phone line at the same time. These lines can be external or internal in nature. In a business setting, the external line links the business to its customers or clients and the internal line connects the business to its workers or employees. Multiline telephone systems use a number of three, four or even five telephone lines at the same time as opposed to the traditional phone system that uses one line to connect two parties.

The Multiline telephone systems offers multiple features like a Caller ID, voicemail, speakerphone, message waiting indicator, mute and volume control, call transferring, hold, intercom, do not disturb, texting. [7]

# Chapter 3 Hardware and Design

## 3.1 Design Parameters

The Combiner Box has been designed as a 4 input, 1 output Line Integration module with a intelligent control and touch interface. It has been designed as a 7 inch by 9 inch rectangular box. The design is modular whereby each line has its independent module. This modularity has been adopted to allow for easy expansion of the module.

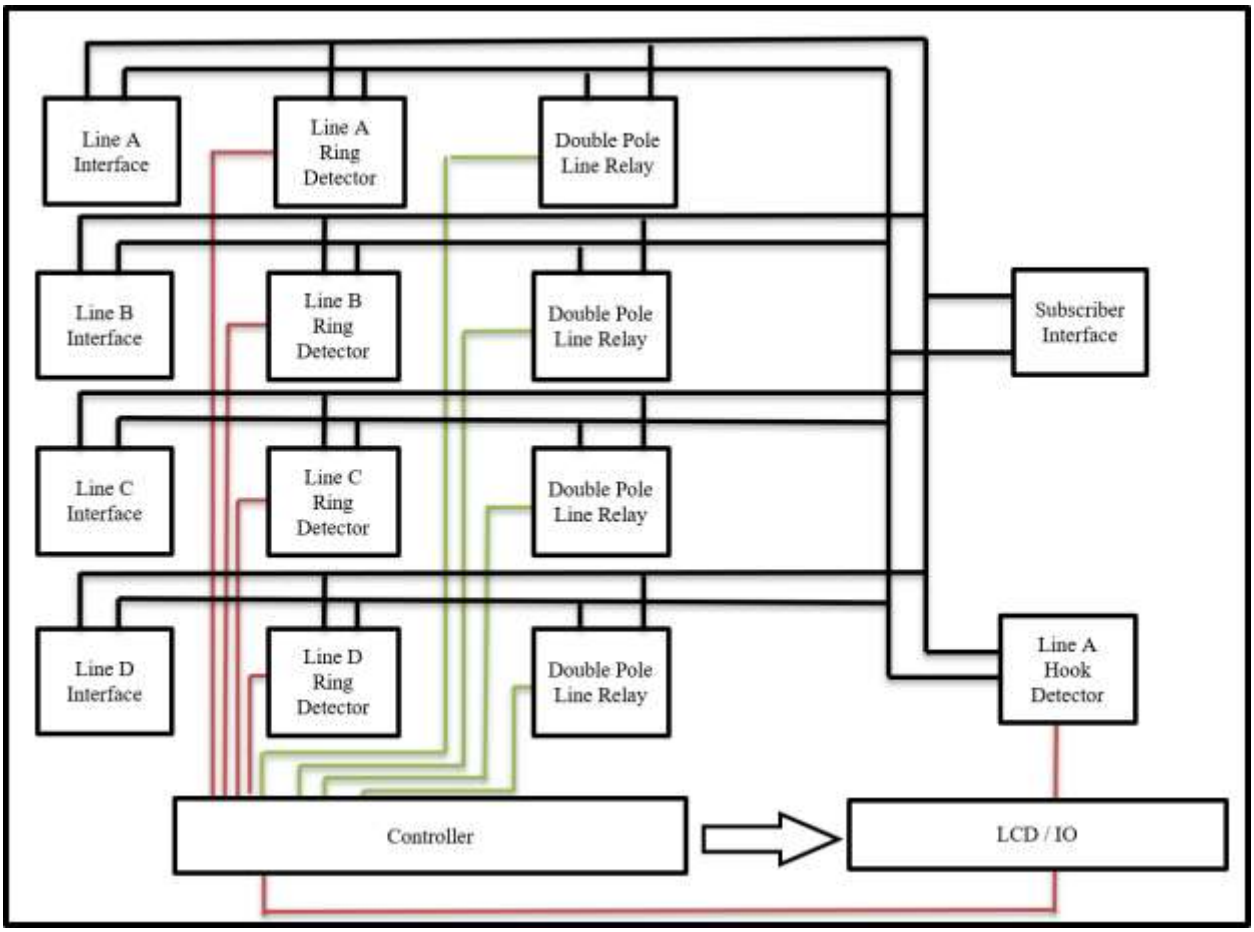
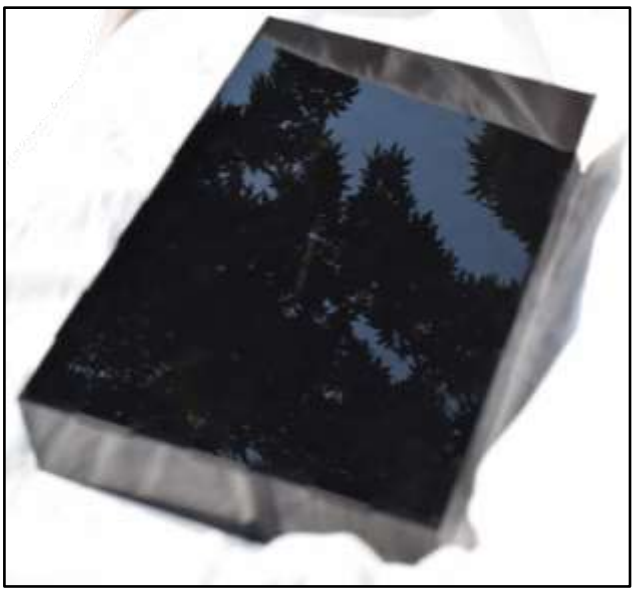



*Figure 9. Block Diagram of Combiner Box*

Apart from separate line modules the product has a separate Control Module, audio Modules and input modules. These have facilitated the design by keeping it simple and expandable at the same time. The PCB designs have been restricted to single layer for simplicity, however this has resulted in the increased size of the Combiner box.



Top View

Side View

Input Panel

Side View

*Figure 10. Combiner Box*

## 3.2 Hardware

### 3.2.1 Detectors

The devices hardware is divided into three main categories as per there function. The first, detectors which detect changes in line state allowing decisions to be made. The second, Control which will provide a link between detectors and display The third display and Interface, which will display results, intimations, prompts and get user feedback The Hardware components are categorized as follows:

1. Detectors.
    a. Ring Detector Module
    b. Hook Detector Module
2. Control - Raspberry Pi
3. Display & Interface
4. Call Answering Module
5. Call Waiting Module
6. Audio Module

### 3.2.2 Ring Detection Module

The Ring Detection Module is designed to detect when a remote user calls a the local phone. As per SS7 protocol to signal a incoming phone call on an analogue line the local exchange sends a 1s pulse of 120v at a frequency of 20Hz. Followed by 0v for another 1s. the module is designed to detect the AC signal on the line and output a single Binary Value (HIGH) to the microcontroller that will allow a decision based on incoming call. Following hardware was used:

1. 0.34uF/ 400v Electrolytic Capacitor
2. 560 Ohm Resistance
3. 47uF/ 50v Polar Capacitor
4. 250V Bridge Rectifier
5. PC817 Optocoupler

6. PN2222 NPN Transistor
7. 10 KOhm Resistance
8. 330 Ohm Resistance

The ring detector is essentially an AC to DC Converter to detect when the phone line rings. The Ring line will be connected to the 560 Ohm and 470nF Capacitor. The 560 Ohm Resistance ensures enough line resistance to allow a small current to pass through the circuit whist not allowing the remote exchange to detect a off hook condition. The capacitor is connected to ensure the -48v DC component on the line is blocked.

This will allow the bridge rectifier to convert the AC Signal of 120V RMS at 20 Hz. The rectifier selected for this purpose is one rated at 250V. This is fed to an optocoupler PC 817.

An optocoupler is a device used to allow electrical isolation between two circuits. This is especially important in our case where the Control unit must be isolated from the high voltages of the Phone line. The Optocoupler contains a IR LED between pins 1 and 2 and a Photo Transistor between pins 3 and 4. When a voltage is applied across pins 1 and 2 the IR LED is on and it forces the Photo diode to saturate, which forms a closed circuit. By connecting a voltage source to pin 4 and grounding pin 3 we can operate a secondary circuit controlled by our primary one having both isolated from each other.

The Optocoupler pin 4 is connected to a 5v power supply through a 10 KOhm resistor and our control input pin in parallel. When the phone does not ring the optocoupler is off therefore providing open circuit. There is no drop across the resistor  Thus the 5v of supply is fed to the input Pin (giving HIGH Output). When the line rings the optocoupler turns on and off repeatedly according to the duty cycle of the ringing signal from the remote exchange. When this happens the optocoupler in turned on creating a short between the ground at pin 3 and pin 4. There is a voltage drop across the resistor now giving the input pin a LOW state. This LOW state is only momentary in nature due to the AC ringing duty cycle and a HIGH LOW Pulse is generated. The pulse has a ½ duty cycle and 20 HZ frequency. (Further explained in Figure 4.2 – 4.4 ).

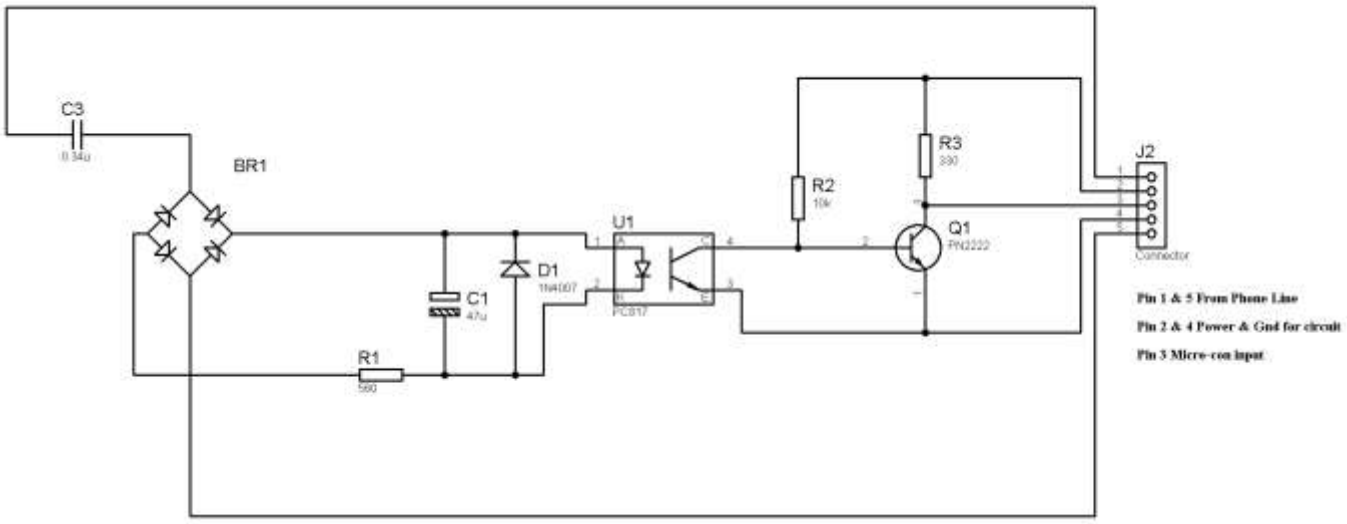



*Figure 11. Ring Detector Circuit*

**0v From Phone Line**
**Indicates No Ring**

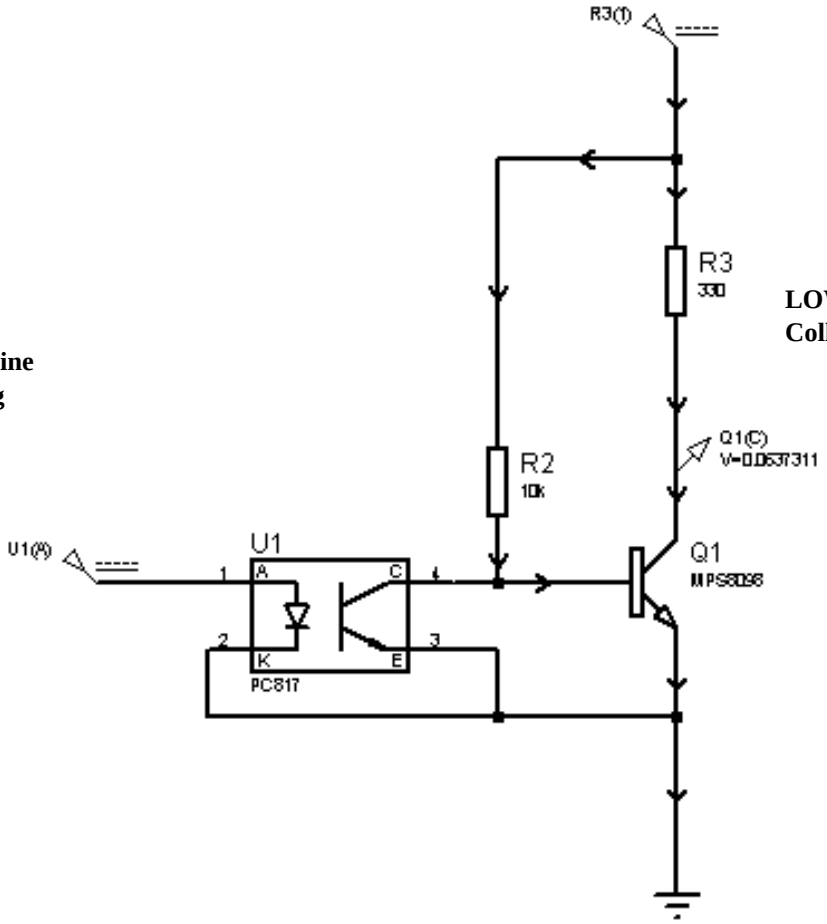
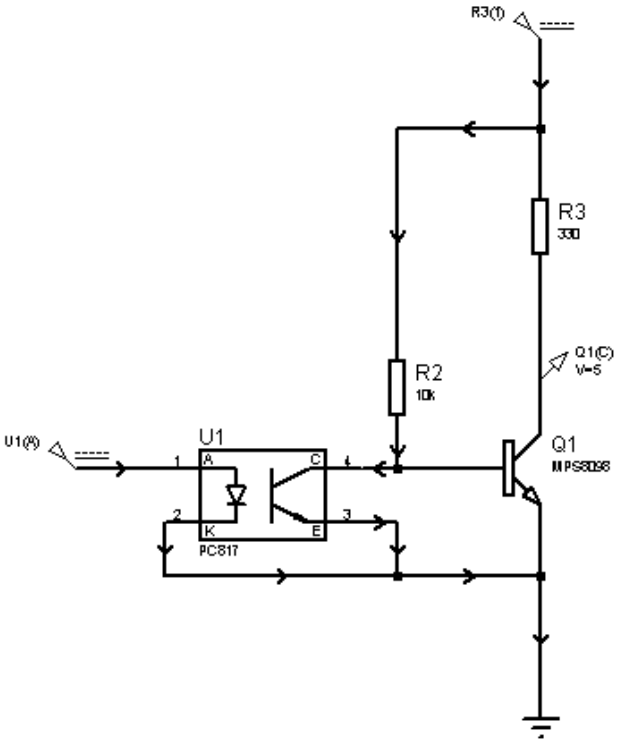



*Figure 12. No Ring State*

**1.5v pulse of 2s (1/2 Duty Cycle) From Phone Line Indicates Ring**




**HIGH voltage at Collector**

*Figure 13. Ring State*

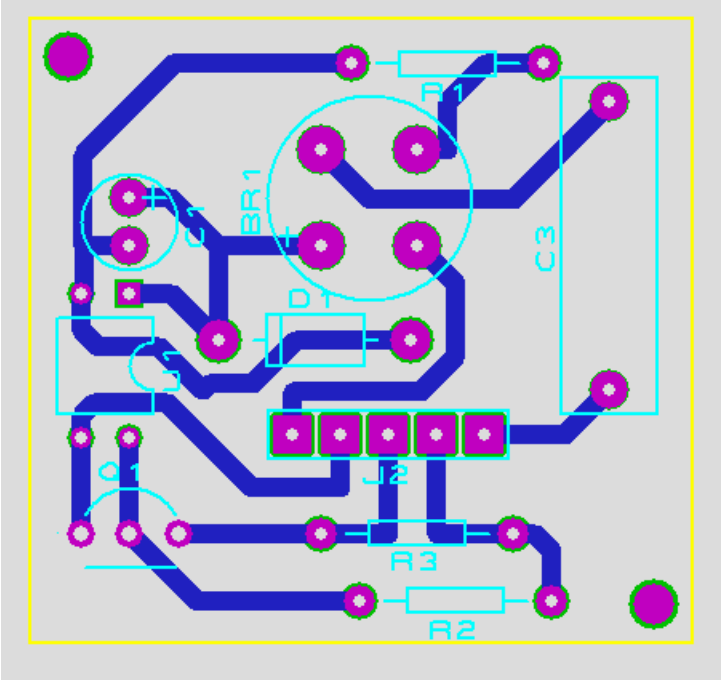



*Figure 14. PCB Design of Ring Detector*

### 3.2.3 **Hook Detection Module**

The hook detector is a module that detects when the user phone is in use. Here Phone in use is termed as whether the receiver is on or off hook. It does this by detecting a voltage change between the on and off hook states of the phone which are discussed below.

As per Line Signalling based on SS7 protocol. The local exchange provides the analogue phone with -48v on the Ring wire and a 0v level on the Tip wire. The local loop between exchange and subscriber has a resistance of approximately 600 Ohm. While on Hook (not in Use) the user's loop acts as an open circuit having a resistance in the order of a few Mega ohms. Once off Hook (in Use) the phone closes the local loop dropping its resistance to 600 ohms and thus dropping the voltage level from -48v to -7v. Following hardware was used:

1. Diode DN4007
2. 10 KOhm Resistor
3. 330 Ohm Reistor
4. 15v Zener Diode
5. PC817 Optocoupler
6. PN2222 NPN Transistor

The hook detector (Figure 4.6) has been designed to detect a change in the voltage level of the phone line from -7v to -48v or vice versa. It does so by use of a PC817 optocoupler that drives a PN2222 transistor giving the desired output on its collector junction.

From the phone line connectors the circuit is fed to a polarity reversal circuit. (Diodes D3, D4, D7, D8) this circuit ensures that the positive (Tip) is fed to the anode and negative (Ring) is fed to the cathode of the PC817. A 10K Ohm resistor is attached to provide a voltage drop and allow a current to pass through the Optocoupler.

2 15v Zener Diodes are placed at the Cathode in series. These have a cumulative breakdown voltage of -30v. When on Hook (Not in Use) the voltage between tip and ring wires is -48v. In this case the Zener diodes break down and conduct closing the circuit. Thus the collector and emitter ends of the PC817 are shorted and there is 0v at base. Due to this the collector voltage is at 5v or HIGH. (Figure 4.7). When off hook (in Use) the Anode is at 0v and the cathode is at -7v. This is not enough to breakdown the Zener diode and thus the Anode and

Cathode of PC817 remain open. So does the collector and emitter. Causing a base Voltage to appear from the 10kOhm resistance and shorting the collector emitter junction. Therefore a 0v or LOW State occurs at collector (Figure 4.5)

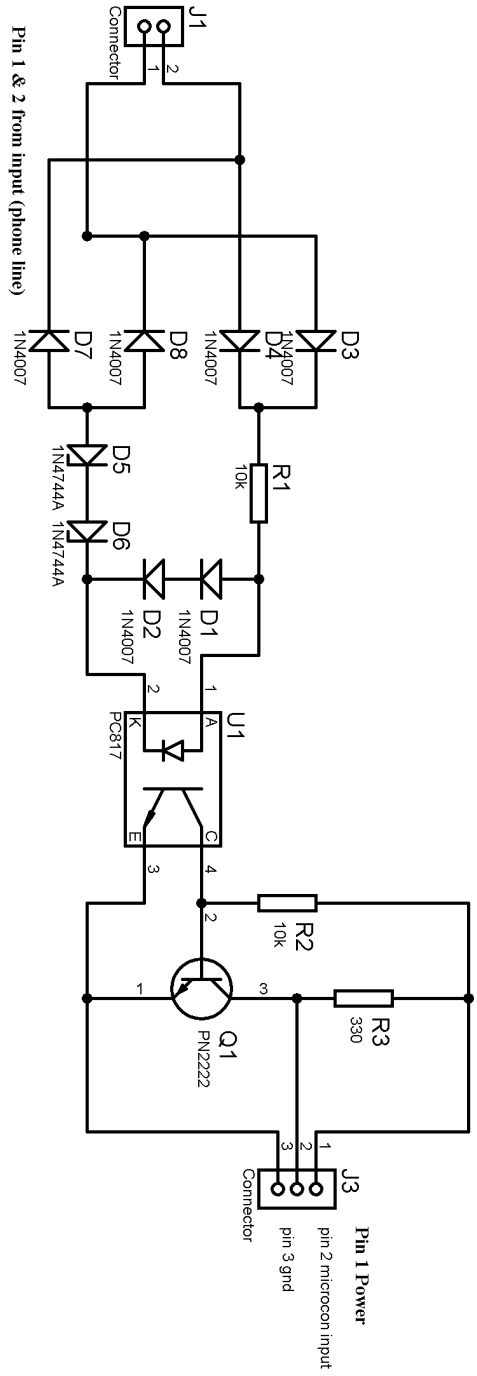



*Figure 15. Hook Detector Circuit*

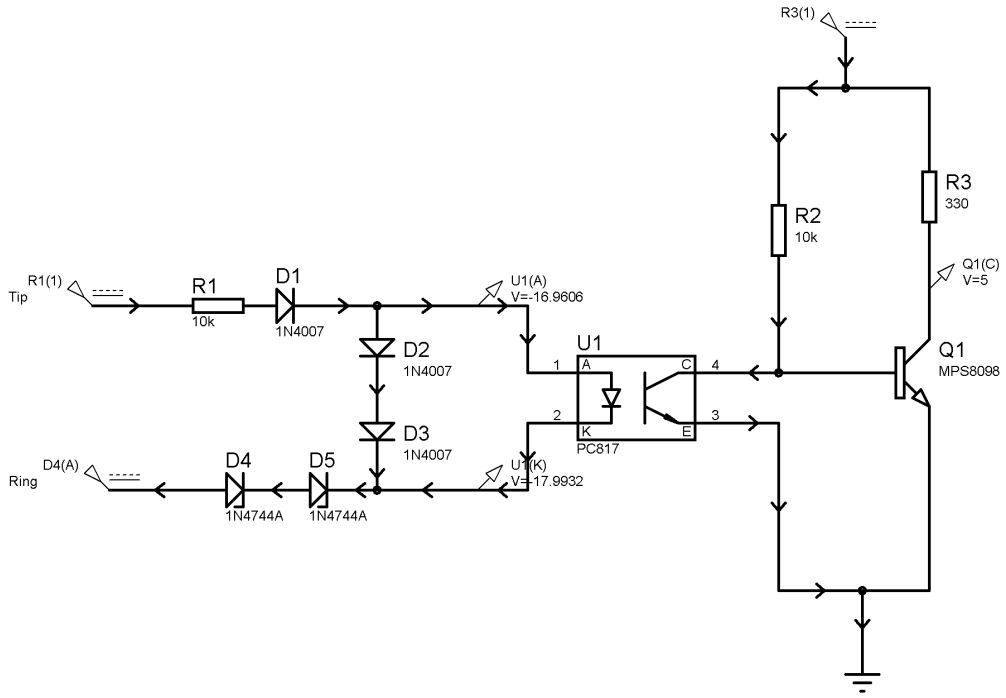



*Figure 16. ON Hook state*

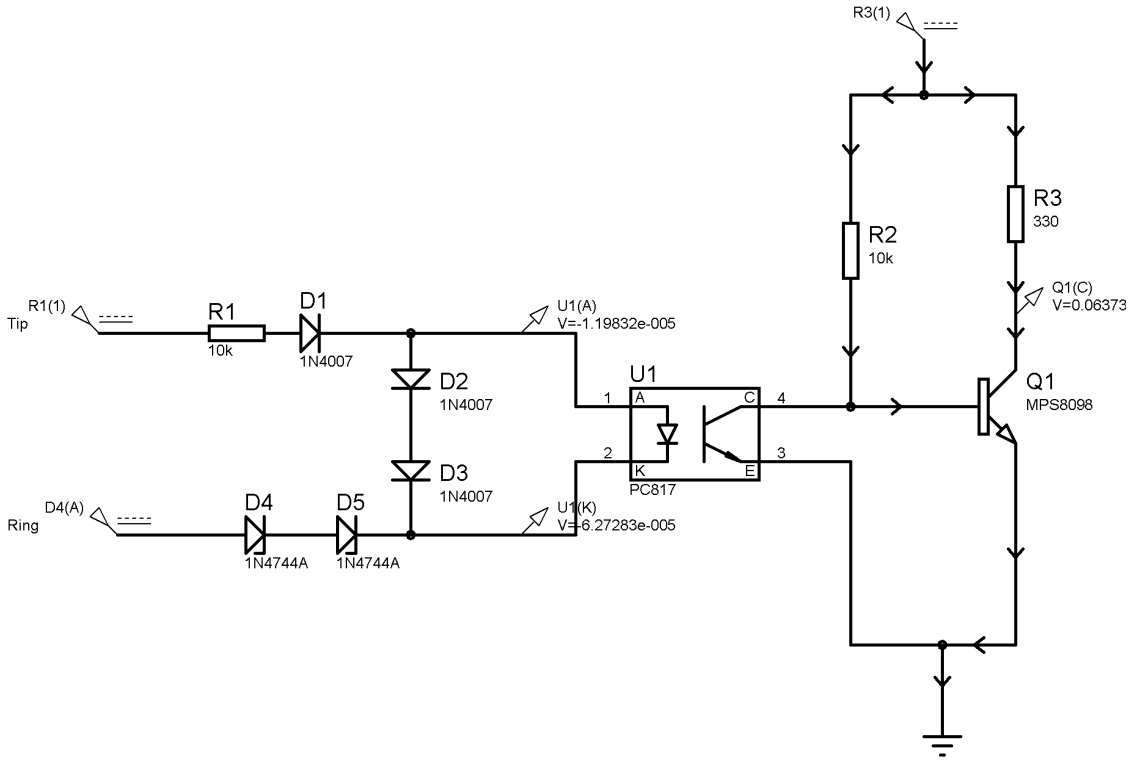



*Figure 17. OFF Hook state*

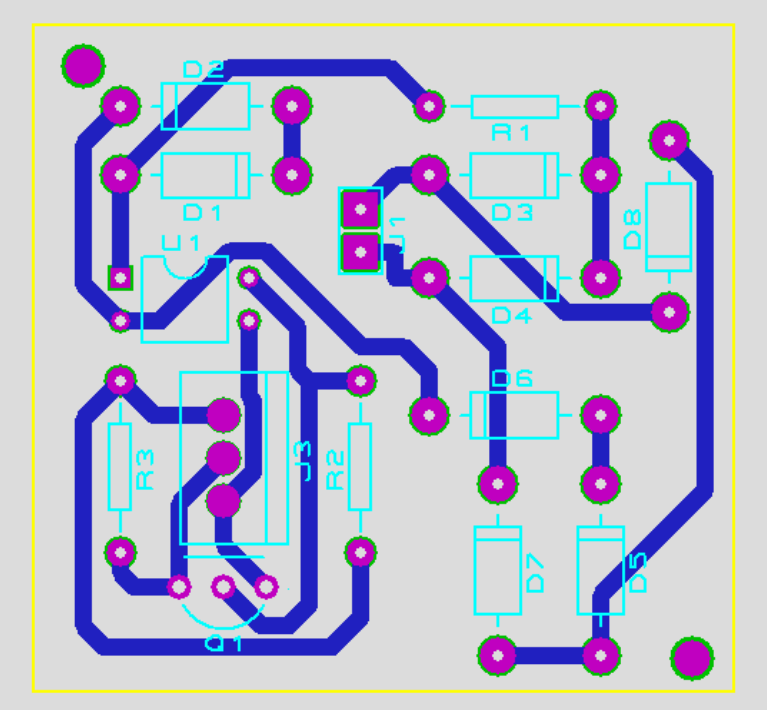



*Figure 18. Hook detector PCB Diagram*

### 3.2.4 **Call Answering Module**

The call answering and waiting is based on two separate modules. Each having 1x 5v OMRON G2V-5 Relay. The relay is a DPDT relay capable of operating on both AC and DC. However in this case DC has been used to energise the relay. Following hardware was used:

1. Resistor 330 Ohm
2. Resistor 1.1k Ohm
3. Resistor 3.3k Ohm
4. 2N2222 NPN Transistor
5. OMRON G5V-2 VDC DPDT Relay
6. 1N4007 Diode

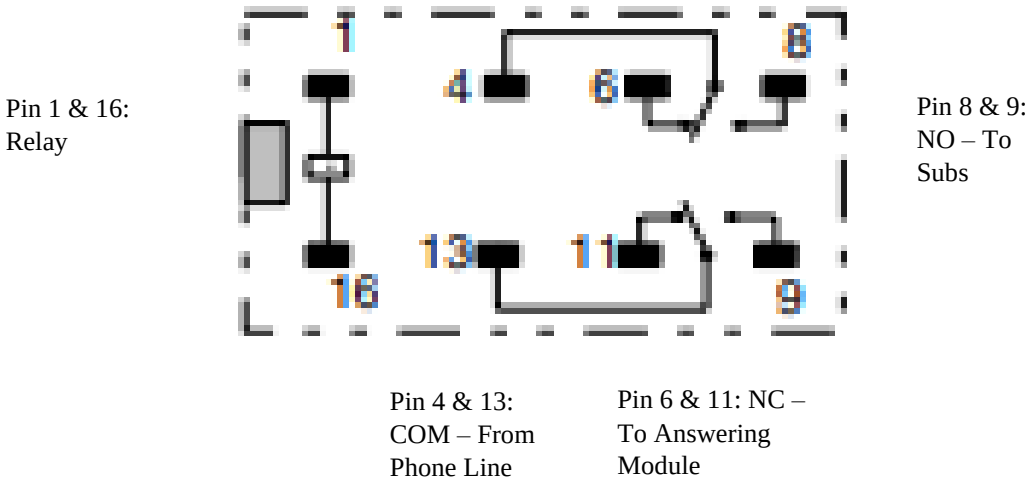



*Figure 19. G5V-2  pin Configuration for call Answering*

The Call answering module connects the subscriber telephone to the FXO port (local loop line). It does this through the output from the micro controller. As the micro controller input has a low current and may not be able to energize the relay  therefore, we use a transistor switch to drive the Relay. A flywheel Diode is also added between the inputs of the relay to block reverse DC voltage from the relay inductor.

As indicated above when the transistor energizes pin 1 and 16, the COM ports 4 and 13 connect to the NO ports 8 and 9. This connects the phone line circuit till the phone allowing it to be used. In normal circumstances when the relay is not energized the phone is connected to the answering module through the NC ports 6 and 11. Which is discussed further

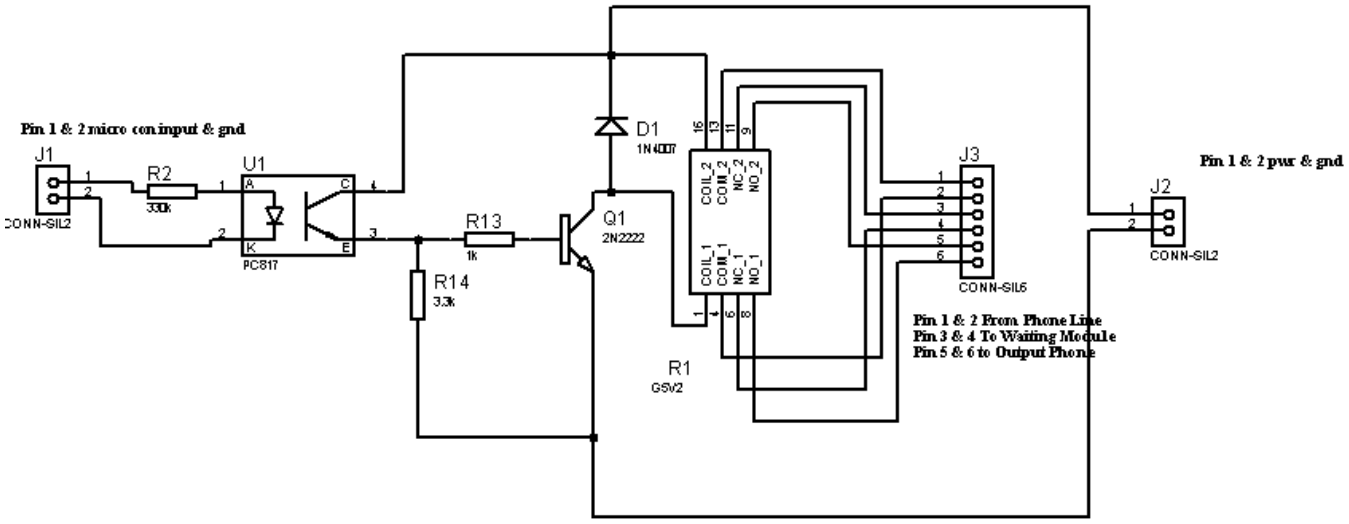



*Figure 20. Call Answering Module Circuit*

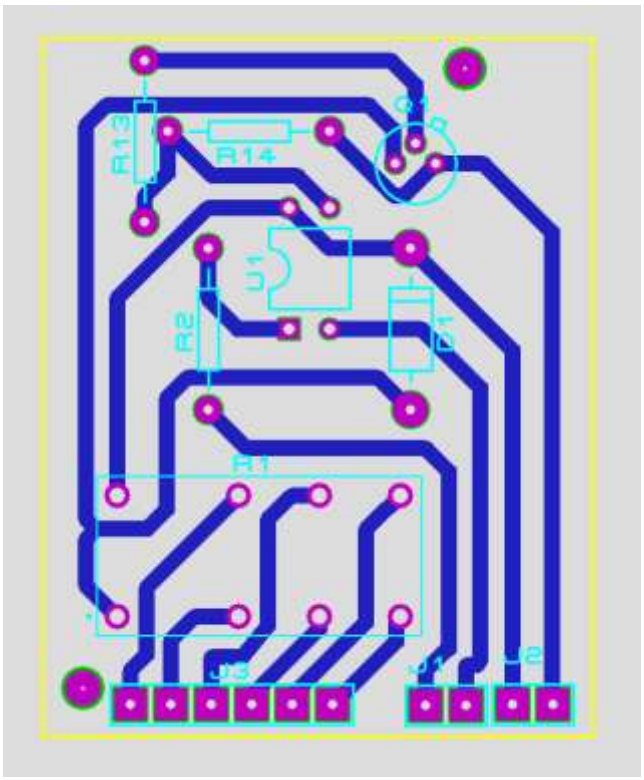



*Figure 21. PCB layout of Call Answering Module*

### 3.2.5 **Call Waiting Module**

The call answering and waiting is based on two separate modules. Each having 1x 5v OMRON G2V-5 Relay. The relay is a DPDT relay capable of operating on both AC and DC. However in this case DC has been used to energise the relay. Following hardware was used:

1. Resistor 330 Ohm
2. Resistor 1.1k Ohm
3. Resistor 3.3k Ohm
4. 2N2222 NPN Transistor
5. OMRON G5V-2 VDC DPDT Relay
6. 0.38uF Capacitor
7. 1N4007 Diode

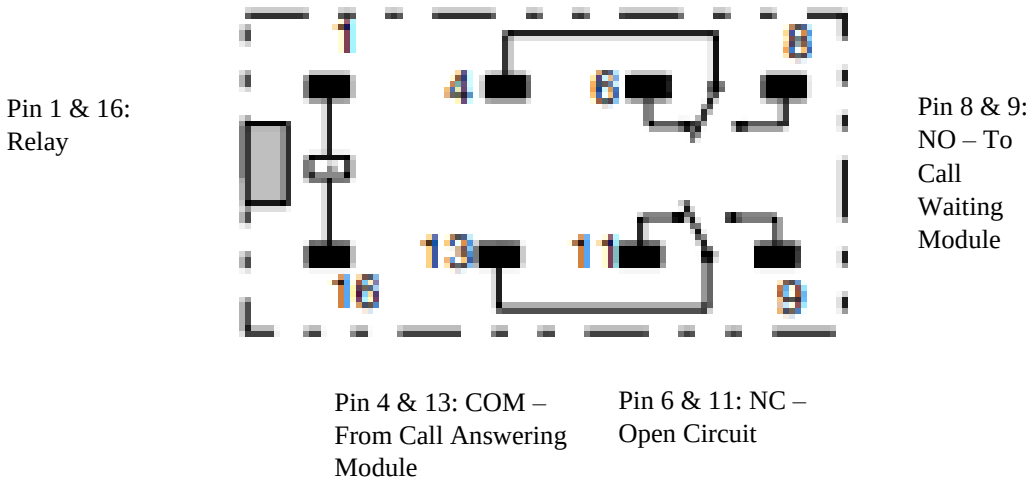



*Figure 22. Pin configuration of G5V-2 for Call waiting*

The call Waiting Module is based on the same circuit as the call answering with a few exceptions. Instead of the COM ports being fed from the phone local loop they are fed by the call answering module. Enabling the call waiting to receive phone signals while the call answering is not activated. The NC ports of the call waiting module is left open. This is done to ensure that unless we activate the call waiting module the line is not engaged or picked up. On the NO ports however, a resistance of 330 Ohm is placed. This small resistance provides enough load to the

line prompting the exchange to put the line off hook and thereby engaging the line. In case a phone was ringing the line will be answered. The 330 ohm resistor also acts as a voltage divider bring down the peak voltage of the audio signal and DC on the phone line so that it can be interfaced with the audio playback Module.

Two 0.38uF capacitors are placed between the 330 ohm resistor and audio playback module in parallel to block any DC component from the line that may damage the audio module. Therefore only audio signals from the audio module are put onto the line. This may be any preprogramed audio message or tone as programmed in the audio module.[12]

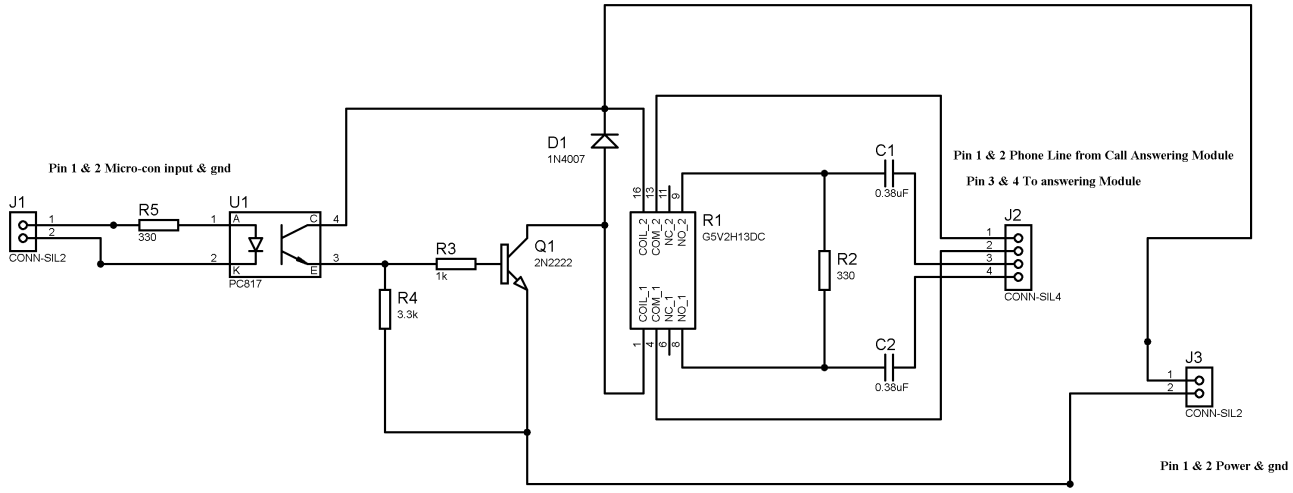



*Figure 23. Call Waiting Module Circuit*

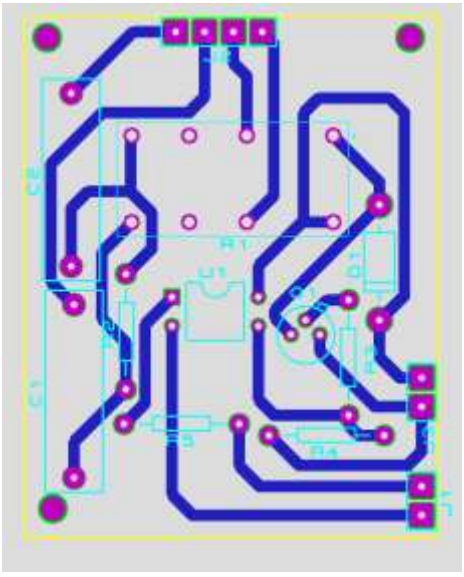


*Figure 24. PCB layout of Call Waiting Module*

### 3.2.6 **Audio Module**

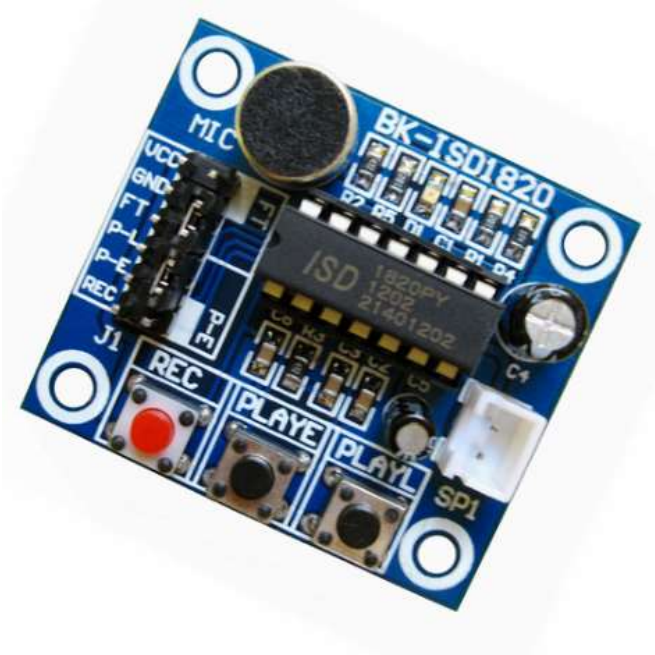



*Figure 25. ISD-1820 Audio Playback Module*

The purpose of the audio module is to allow an automated call waiting message to be forwarded for the period the phone line is put on hold by the user. This may be done by any audio playback module. In our case the ISD 1820 audio module is used.

The ISD 1820 module is a audio voice recording and playback module based on the ISD1820 micro-chip. It supports playback for 8 to 20s at a sampling rate of 3.2K. The module can be controlled by push buttons or by micro controllers easily.

The module features an 8 ohm speaker driver which is connected to the call waiting module to send audio signals to the phone line. The module allows recording upto 20s and allows on loop play back which is suitable for call waiting messages. For further references datasheet of the module has been attached.

### 3.2.7 **Line Module**

The Line Module is a single layer PCB which is a combined form of the ring detector, Call Answering and Call waiting relays and control. The module supports power and ground ports, in addition to 3x Microcontroller ports and ports for phone input, output and audio module. Each

line module is independent of the other and is supplied with power and microcontroller input/ output independently. Not only does this bring modularity to the combiner box but also allows easy troubleshooting of problems, fault identification and does not render the whole product unusable in case of a problem in a particular line/ module.

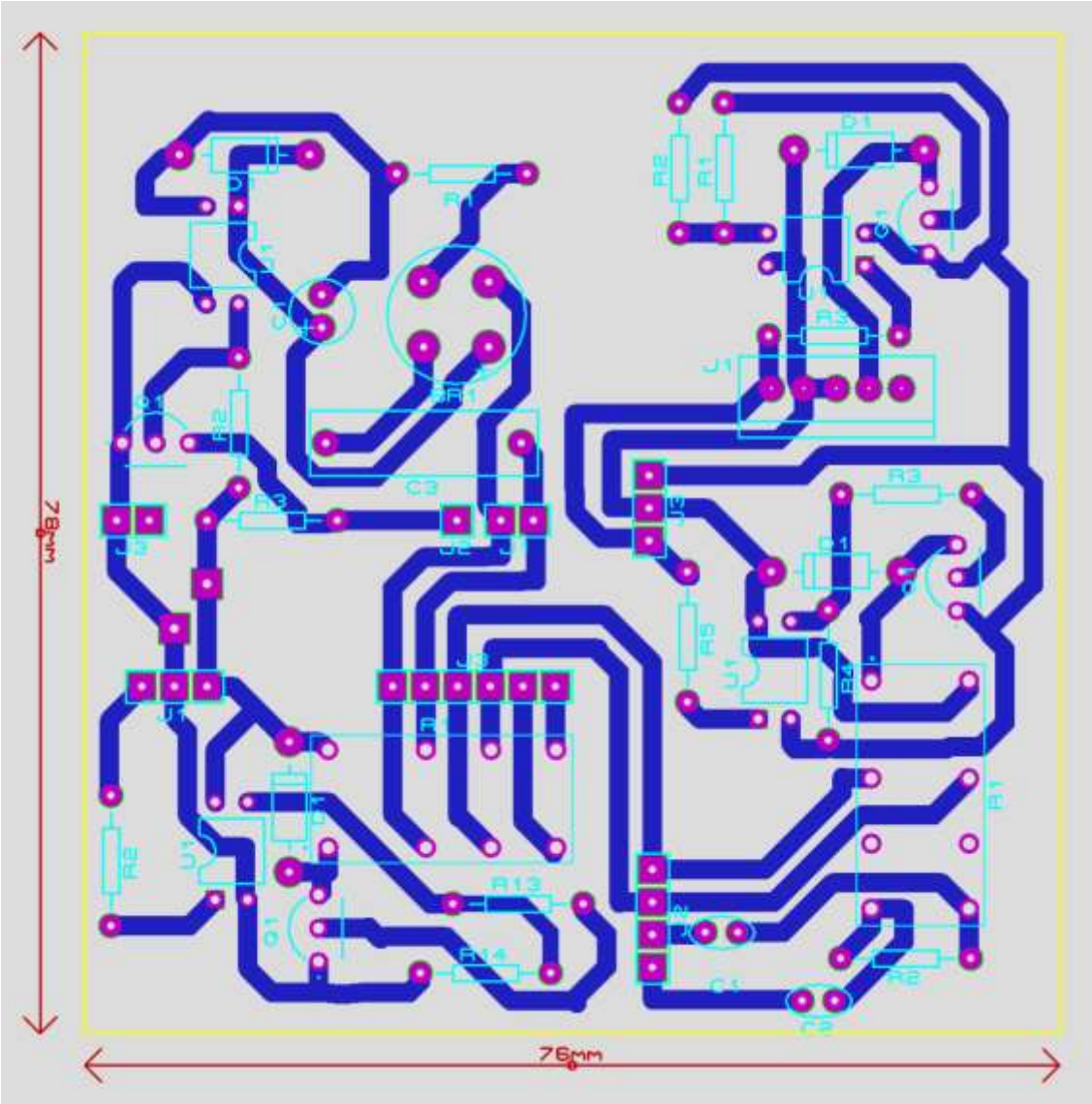



*Figure 26. PCB Design of Line Module*

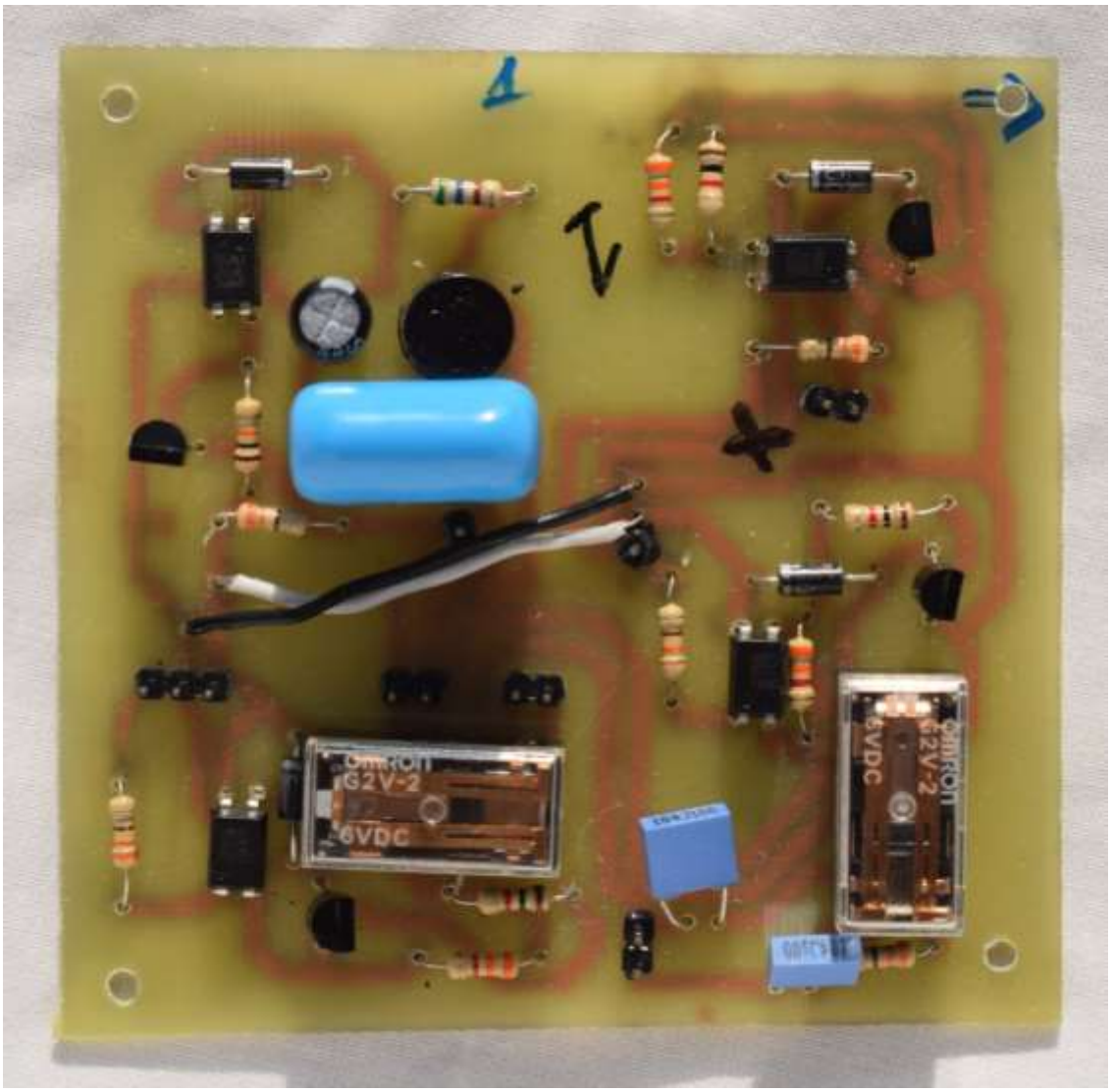



*Figure 27. Completed Line Module*

## 3.3 **Control**

Control of the hardware can be done through any micro-controller depending on the interface features and speed of switching that is to be provided to the device. In case the Combiner box is to be provided with regular interface composed of physical buttons an 8086 microcontroller or equivalent may be used. For this particular project Raspberry pi has been used to allow shifting the entire interface onto an LCD touchscreen for educational purposes. This has allowed us to explore the Pi as a smart microcontroller improving our knowledge of its programming in python, its capabilities and limitations for such products. It must be noted

however, that the addition of Pi has withered its cost effectiveness and is not recommended for industrial application of the project.

In case of using any other microcontroller, the same principles and logic states as discussed above may apply to the device thus allowing it to be modular in nature.

### 3.3.1 **Raspberry Pi**

The Raspberry Pi for model B is the latest of the new generation Raspberry Pi computers that supports and enhanced GPU, CPU, RAM and I/O performance. The pipe comes with a quad core 64 bit ARM cortex processor, 1 gigabyte RAM which was seen as sufficient for this particular project and one 8GB SD card. The pi supports multiple display outputs in form of 2 micro HDMI ports and one DPI port. Despite being a computer the fact that the pie has 40 pin GPIO header Has allowed us to use it as a microcontroller. Pi's GPIO pins provide input and output on the same pins with either pull up or pull down Resistances. [9]

However it was noticed that after installing and running the Raspbian operating system the Raspberry Pi was somewhat slow in terms of clock rate which does not cause problems on a system with few lines (four) however may result in errors if the module is based on a greater number of lines (more than four) therefore making it unsuitable for this operation. Each of the Raspberry Pi GPIO pins have a input output voltage of 3.3 volts and have an operating current from -2ma to 2ma. This operating current range was seem to be low to operate the several relays incorporated in the module and therefore required an addition of a driving circuit as well. The advantages of using Raspberry Pi can be summarized below :[9]

1. Pi is small, low powered and easily programmable
2. With a linux OS the pi can provide an interactive interface and added hardware functionality that can match modern PABX
3. The pi can be programmed using the RPI.GPIO module in python, wiringpi in C and node.js in JavaScript, allowing multi language support for the device software. In this case programming has been done in python.

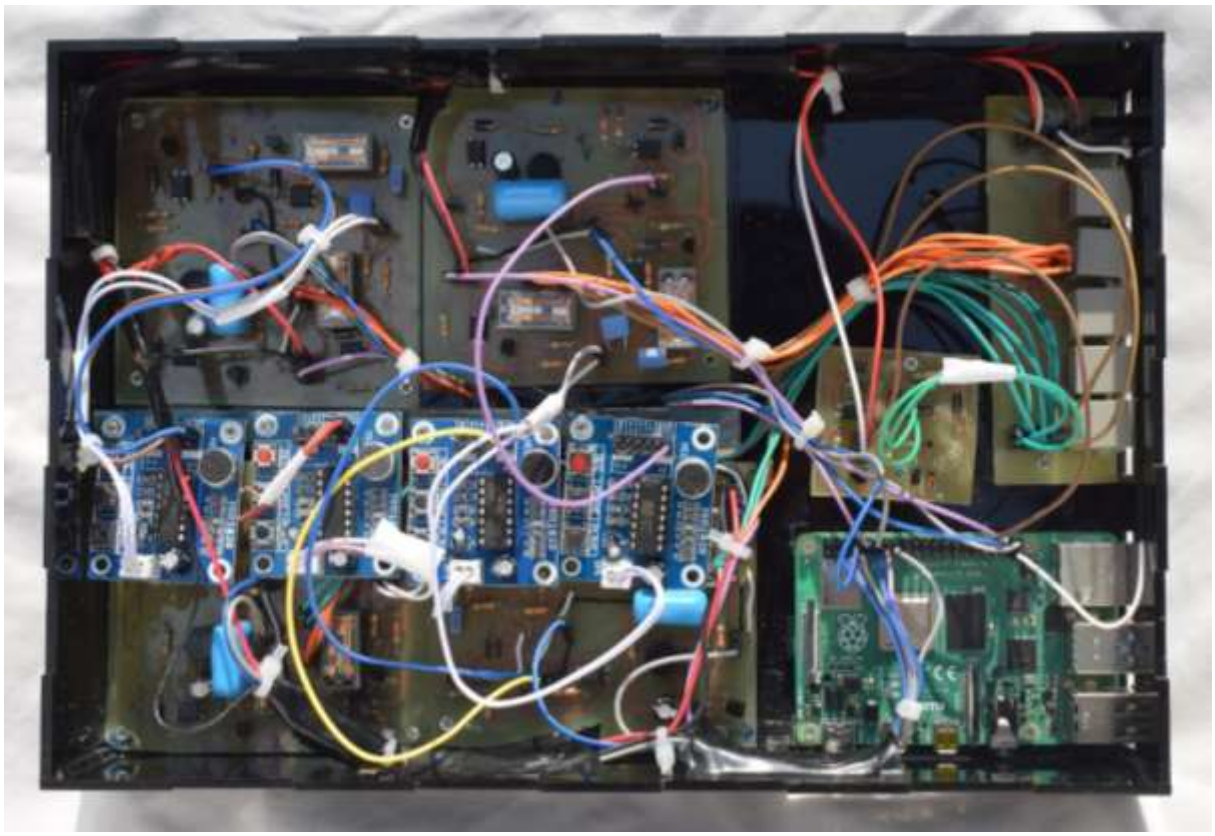


*Figure 28. Top view of Components of Combiner Box*

## 3.4 **Software**

For the combiner Box software programming was done in Python. The version used was python 3.8. Notable features within the python script included RPi.GPIO module and multithreading. A simple yet interactive GUI was made using tkinter module. Further details of the software have been provided below

### 3.4.1 **RPi.GPIO**

RPi.GPIO module defines in itself a class that provides control to GPIO pins on the raspberry Pi. Through this it allows user to control 26 pins on the raspberry pi, however the module is not suited for fast changing I/O or for timing in critical applications. This is because it is unpredictable when python would be garbage collecting during execution of the module and other processes may also be taking priority of tasks in the CPU. RPi.GPIO does not support SPI, I2C, hardware PWM or serial functionality, however software PWM is a functionality in it.

Whilst using RPi.GPIO first pin modes of the used GPIO pins was set, followed by setting up pins as output or input pins.

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(hook, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(ring1, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(ring2, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(ring3, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(ring4, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(relay1[0], GPIO.OUT)
GPIO.setup(relay1[1], GPIO.OUT)
GPIO.setup(relay1[2], GPIO.OUT)
GPIO.setup(relay1[3], GPIO.OUT)
GPIO.setup(relay2[0], GPIO.OUT)
GPIO.setup(relay2[1], GPIO.OUT)
GPIO.setup(relay2[2], GPIO.OUT)
GPIO.setup(relay2[3], GPIO.OUT)
```

*Figure 29. Python code for setup of GPIO Pins*

One feature of the RPi.GPIO module that has been used in our code is the *`add_event_detect()`* method. This method creates a thread in the python code whenever it detects a particular event in relation to a particular GPIO pin. It may detect rising, falling or both states and calls a method to be executed every time the *`add_event_detect()`* method runs. The method has been used to detect incoming ring on each of the four lines and hook state of the subscriber phone. This method is beneficial as it allows us to interrupt the code in the background that is running and execute a ringing or phone hook related method.

```
def hookup(hook):
    global hookstate
    global time_hd
    if GPIO.event_detected(hook):
        if GPIO.input(hook) == False:
            time_hd = time.time()
        else:
            hookstate = 1

GPIO.add_event_detect(hook, GPIO.BOTH, callback = hookup)
```

*Figure 30. Python code for add_event_detect()*

### 3.4.2 **Multithreading**

Threading is defined as the execution of small sequence of code independently by a scheduler. Python like other languages like ruby, C# and java also supports multithreading. Python threading allows us to have different blocks of code run in parallel. Multithreading is available from python 3.6 and later. Python threads however, merely appear to run independently and result in multiprocessing causing extra overhead, depending upon processing speed. This, therefore is a limiting factor in our particular application where it was noticed that multiple threads was causing the application to slow down. The `_thread` module provides several functions and methods to create and manipulate threads. The `_thread.start_new_thread(function,args[, kwargs])` method is used to define a thread. The thread calls a function that is to be executed. Once its execution is completed the thread is destroyed. In the combiner box application threads are created and made to run infinitely for the duration of operation. [11]

```
def output_allot():
    global relay1_state
    global relay2_state
    while True:
        for i in range(4):
            GPIO.output(relay1[i], relay1_state[i])
            GPIO.output(relay2[i], relay2_state[i])

start_new_thread(output_allot, ())
```

*Figure 31. Python code for a thread*

### 3.4.3 **Tkinter GUI**

Tkinter provides a binding to python with the Tk GUI toolkit. It is the standard Python interface. Tkinter is included in Linux, Mac and Windows Operating Systems. Tkinter is free under the Python license. A Tkinter software operates as a single root window that contains several widgets. These widgets may be labels, buttons Pages, menu buttons etc. that form part of the core widgets. Apart from this tkinter provides a separate Tkmessagebx module to display message dialog boxes. Tkinter supports inheritance of widgets as each widget is created as a class. [10]

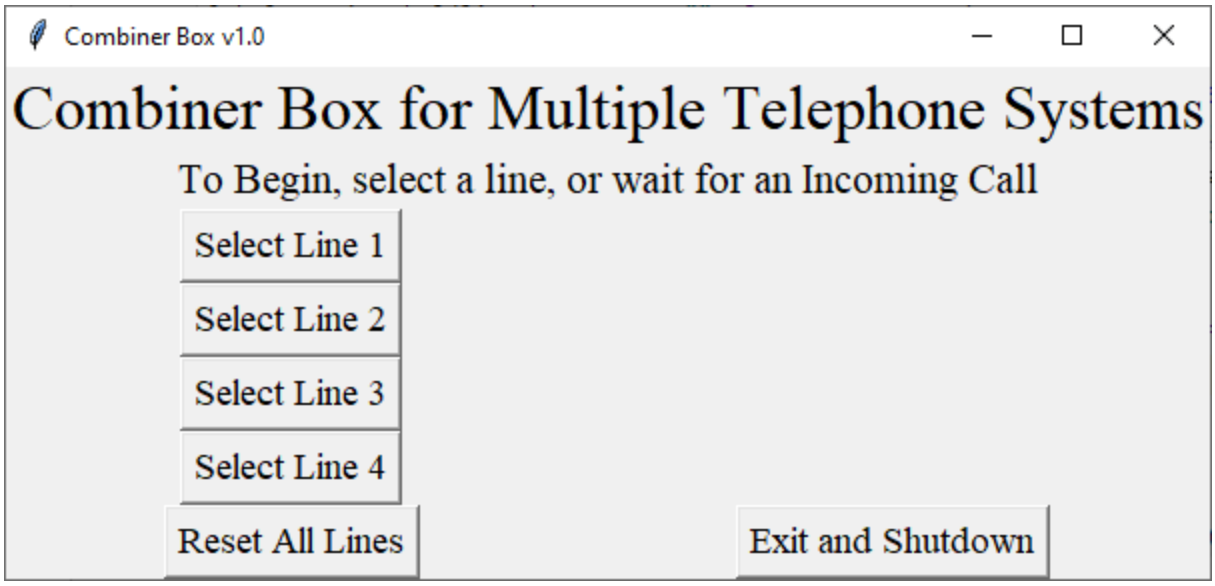



*Figure 32. Main Page of GUI*

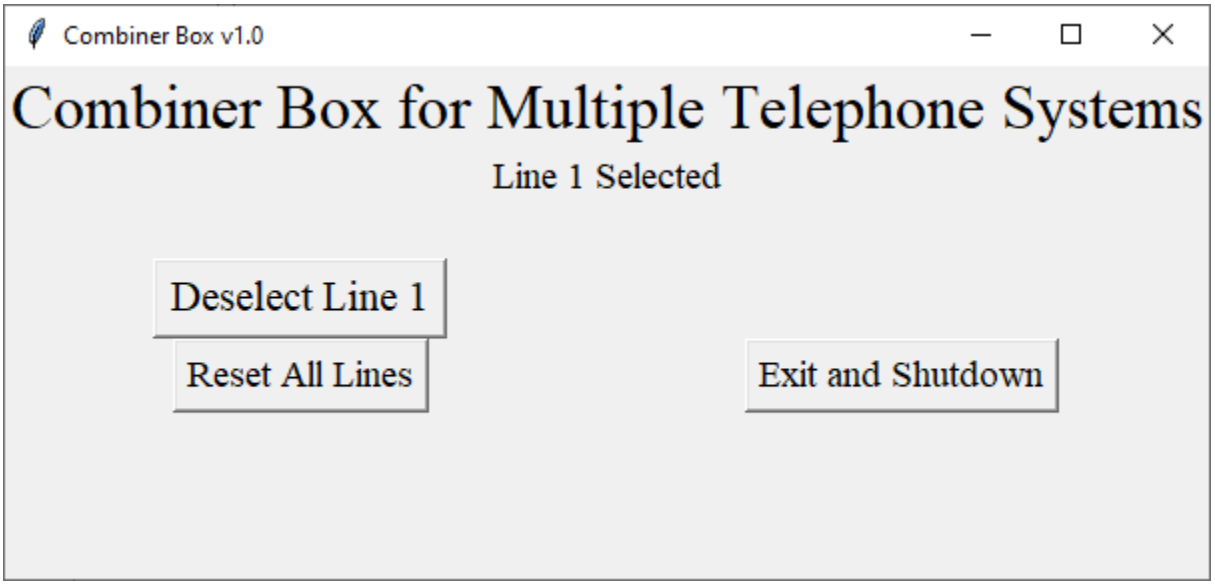



*Figure 33. line 1 as selected*

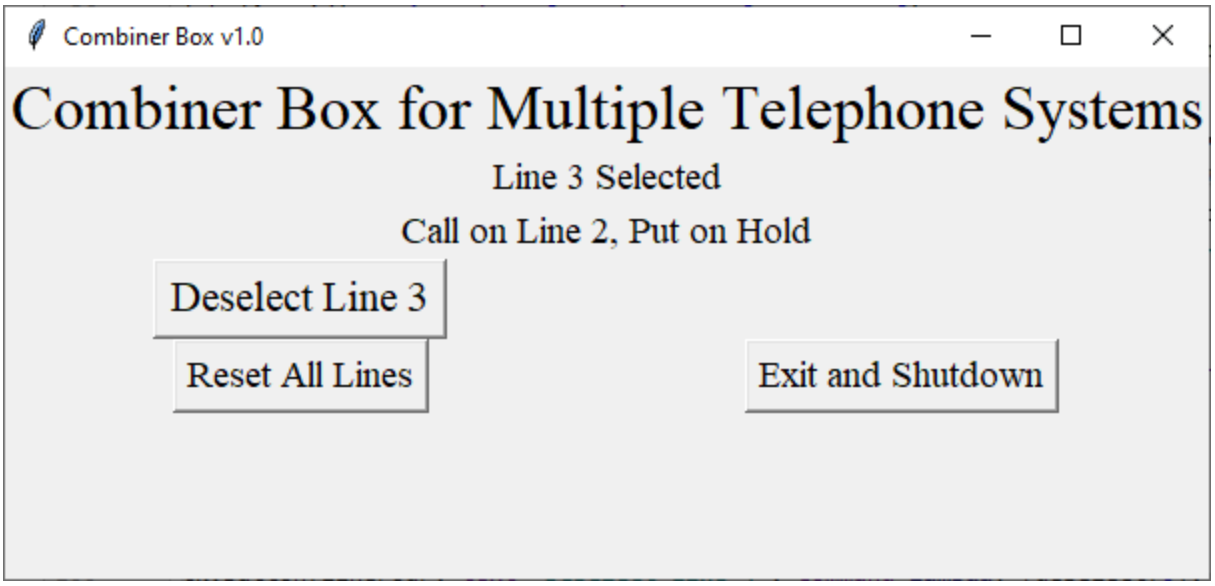



*Figure 34. Intimation of Phone Call on line other than selected*

# Chapter 4 Recommendations and Conclusion

## 4.1 Recommendations and Improvements

1. Use of an alternative micro controller that is dedicated unlike the pi where python script runs as overhead in the operating system
2. The LCD can be replaced with buttons to make a more affordable version of the product.
3. Introducing a DTMF detector to allow changing lines from the telephone and not the combiner box.
4. Integration of an IP Phone by use of an ATA adaptor.
5. Conference calling facility can be added to allow multiple lines to interconnect.
6. Security and encryption modules may be added in each combiner box which will add a layer of security.
7. A master slave steno phone system can also be introduced.
8. Voice mail facility with custom greeting messages can also be featured in the device.

## 4.2 Conclusion

1. The Combiner box for multiple telephone systems will provide its users with easy facilitation in a multi networked environment.
2. Its use may not only be restricted to Pakistan Army's multi networked environment but also small, medium and large office environments.
3. It will provide with cheaper alternatives to PABX installations, and digital line requirements.
4. The combiner box will save manpower in terms of operators and may also replace multiline master slave steno systems.
5. The system is passive, and will not interfere by any other signaling protocols or ISDN / DSL signaling that may be on the line.

## Appendix A

## Project Timeline

| **Progress** | **May 19** | **Jun 19** | **Jul 19** | **Aug 19** | **Sep 19** | **Oct 19** | **Nov 19** | **Dec 19** | **Jan 20** | **Feb 20** | **Mar 20** | **Apr 20** | **May 20** | **Jun 20** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Literature review | Completed | | | | | | | | | | | | | |
| Design of Ring Detector | | | Completed | | | | | | | | | | | |
| Design of Hook Detector | | | | Completed | | | | | | | | | | |
| Implementation of 1x Interface Module with Arduino | | | | | | Completed | | | | | | | | |
| Implementation of 2x Interface Module with Arduino | | | | | | | | Completed | | | | | | |
| Implementation of 4x Interface Module with Arduino | | | | | | | | | Completed | | | | | |
| GUI Design, Implementation with Raspberry Pi | | | | | | | | | Completed | | | | | |
| PCB Design | | | | | | | | | | Completed | | | | |
| Proc of Enclosure, LCD and Test and Trial | | | | | | | | | | | | | | * |

*Completed

## Appendix B

## Cost Breakdown

| Components | Qty | Unit Price | Amount | Remarks |
|---|---|---|---|---|
| Single Layer PCB design & manufacture | 6 | 1500 | 9000 | 4x Line Modules, 1x Hook detector, 1x Input PCB |
| Audio Module ISD-1820 | 4 | 450 | 1800 | |
| Raspberry Pi 4 | 1 | 8000 | 8000 | |
| Raspberry Pi LCD Touchscreen | 1 | 7500 | 7500 | Not Available in market due to COVID-19 |
| Enclosure | 1 | 4000 | 4000 | 7in by 6in Acrylic component Box |
| Miscellaneous | | | 4000 | Wire, active components e.g resistors, transistors, couplers, relays etc. |
| Total | | | Rs 34300/- | |

## Appendix C

## Project Code

```
import tkinter as tk
import time
import tkinter.messagebox as MsgBox
import array as arr
import RPi.GPIO as GPIO
from _thread import start_new_thread

relay1 = [20, 15, 27, 3]
relay2 = [16, 14, 17, 2]
relay1_state = [False, False, False, False]
relay2_state = [False, False, False, False]
hook = 12
hookstate = True
time_hd = 0
busy_time = [0, 0, 0, 0]
connect_time = 0
isconn = True
ring1 = 21
ring2 = 18
ring3 = 22
ring4 = 4
ringstate = [False, False, False, False]
time_rd = [0, 0, 0, 0]
busy_time= [0, 0, 0, 0]

def connect_line(i):
    global relay1_state
    relay1_state[i] = True

def deselect(i):
    global relay1_state
    relay1_state[i] = False

def reset():
    global relay1_state
    global relay2_state
    relay1_state = [False, False, False, False]
    relay2_state = [False, False, False, False]


GPIO.setmode(GPIO.BCM)
GPIO.setup(hook, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(ring1, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(ring2, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(ring3, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(ring4, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(relay1[0], GPIO.OUT)
GPIO.setup(relay1[1], GPIO.OUT)
GPIO.setup(relay1[2], GPIO.OUT)
GPIO.setup(relay1[3], GPIO.OUT)
```

```
GPIO.setup(relay2[0], GPIO.OUT)
GPIO.setup(relay2[1], GPIO.OUT)
GPIO.setup(relay2[2], GPIO.OUT)
GPIO.setup(relay2[3], GPIO.OUT)

LARGE_FONT = ("Times New Roman", 50)
MED_FONT = ("Times New Roman", 46)
SMALL_FONT = ("Times New Roman", 38)

app = tk.Tk()

def show_frame(frame):
    frame.tkraise()

StartPage = tk.Frame(app)
tk.Label(StartPage, text="Combiner Box for Multiple Telephone Systems",
font=LARGE_FONT).grid(row=0, column=0, columnspan=2, rowspan = 2, sticky='nsew', padx
= 5, pady = 10)
tk.Label(StartPage, text="To Begin, select a line, or wait for an Incoming Call",
font=MED_FONT).grid(row=2, column=0, columnspan=2, rowspan=2, sticky='nsew', padx =
5, pady = 10)
tk.Button(StartPage, text="Select Line 1", command=lambda: [connect_line(0),
show_frame(lineOne)], font=SMALL_FONT, height = 2).grid(row=4, column=0, rowspan=4,
sticky='nsew', padx = 5, pady = 10)
tk.Button(StartPage, text="Select Line 2", command=lambda: [connect_line(1),
show_frame(lineTwo)], font=SMALL_FONT, height = 2).grid(row=8, column=0, rowspan=4,
sticky='nsew', padx = 5, pady = 10)
tk.Button(StartPage, text="Select Line 3", command=lambda: [connect_line(2),
show_frame(lineThree)], font=SMALL_FONT, height = 2).grid(row=12, column=0,
rowspan=4, sticky='nsew', padx = 5, pady = 10)
tk.Button(StartPage, text="Select Line 4",  command=lambda: [connect_line(3),
show_frame(lineFour)], font=SMALL_FONT, height = 2).grid(row=16, column=0, rowspan=4,
sticky='nsew', padx = 5, pady = 10)
tk.Button(StartPage, text="Reset All Lines", command=reset(), font=SMALL_FONT, height
= 2).grid(row=20, column=0, padx = 5, pady = 5)
tk.Button(StartPage, command=lambda: on_closing(), text="Exit and Shutdown",
font=SMALL_FONT, height = 2).grid(row=20, column=1, padx = 5, pady = 5)


lineOne = tk.Frame(app)
tk.Label(lineOne, text="Combiner Box for Multiple Telephone Systems",
font=LARGE_FONT).grid(row=0, column=0, columnspan=2,  sticky='nsew')
tk.Label(lineOne, text="Line 1 Connected", font=SMALL_FONT).grid(row=4, column=0,
columnspan=2, rowspan=2,  sticky='nsew')
label2 = tk.Label(lineOne, text="", font =SMALL_FONT)
label2.grid(row=6, column=0, columnspan=1, rowspan=2,  sticky='nsew')
tk.Button(lineOne, text="Deselect Line", command=lambda: [deselect(0),
show_frame(StartPage)], font=MED_FONT, height = 4).grid(row=8, column=0,
sticky='nsew')
tk.Button(lineOne, text="Reset All Lines", command=lambda: [reset(),
show_frame(StartPage)], font=SMALL_FONT, height = 4).grid(row=9, column=0)
tk.Button(lineOne, command=lambda: on_closing(), text="Exit and Shutdown",
font=SMALL_FONT, height = 4).grid(row=9, column=1)

lineTwo = tk.Frame(app)
```

```
tk.Label(lineTwo, text="Combiner Box for Multiple Telephone Systems",
font=LARGE_FONT).grid(row=0, column=0, columnspan=2, sticky='nsew')
tk.Label(lineTwo, text="Line 2 Connected", font=SMALL_FONT).grid(row=4, column=0,
columnspan=2, rowspan=2, sticky='nsew')
label3 = tk.Label(lineTwo, text="", font =SMALL_FONT)
label3.grid(row=6, column=0, columnspan=2, rowspan=2, sticky='nsew')
tk.Button(lineTwo, text="Deselect Line", command=lambda: [deselect(1),
show_frame(StartPage)], font=MED_FONT, height = 4).grid(row=8, column=0,
sticky='nsew')
tk.Button(lineTwo, text="Reset All Lines", command=lambda: [reset(),
show_frame(StartPage)], font=SMALL_FONT, height = 4).grid(row=9, column=0)
tk.Button(lineTwo, command=lambda: on_closing(), text="Exit and Shutdown",
font=SMALL_FONT, height = 4).grid(row=9, column=1)

lineThree = tk.Frame(app)
tk.Label(lineThree, text="Combiner Box for Multiple Telephone Systems",
font=LARGE_FONT).grid(row=0, column=0, columnspan=2, sticky='nsew')
tk.Label(lineThree, text="Line 3 Connected", font=SMALL_FONT).grid(row=4, column=0,
columnspan=2, rowspan=2, sticky='nsew')
label4 = tk.Label(lineThree, text="", font =SMALL_FONT)
label4.grid(row=6, column=0, columnspan=2, rowspan=2, sticky='nsew')
tk.Button(lineThree, text="Deselect Line", command=lambda: [deselect(2),
show_frame(StartPage)], font=MED_FONT, height = 4).grid(row=8, column=0,
sticky='nsew')
tk.Button(lineThree, text="Reset All Lines", command=lambda: [reset(),
show_frame(StartPage)], font=SMALL_FONT, height = 4).grid(row=9, column=0)
tk.Button(lineThree, command=lambda: on_closing(), text="Exit and Shutdown",
font=SMALL_FONT, height = 4).grid(row=9, column=1)

lineFour = tk.Frame(app)
tk.Label(lineFour, text="Combiner Box for Multiple Telephone Systems",
font=LARGE_FONT).grid(row=0, column=0, columnspan=2, sticky='nsew')
tk.Label(lineFour, text="Line 4 Connected", font=SMALL_FONT).grid(row=4, column=0,
columnspan=2, rowspan=2, sticky='nsew')
label5 = tk.Label(lineFour, text="", font =SMALL_FONT)
label5.grid(row=6, column=0, columnspan=2, rowspan=2, sticky='nsew')
tk.Button(lineFour, text="Deselect Line", command=lambda: [deselect(3),
show_frame(StartPage)], font=MED_FONT, height = 4).grid(row=8, column=0,
sticky='nsew')
tk.Button(lineFour, text="Reset All Lines", command=lambda: [reset(),
show_frame(StartPage)], font=SMALL_FONT, height = 4).grid(row=9, column=0)
tk.Button(lineFour, command=lambda: on_closing(), text="Exit and Shutdown",
font=SMALL_FONT, height = 4).grid(row=9, column=1)


for frame in (StartPage, lineOne, lineTwo, lineThree, lineFour):
    frame.grid(row=0, column=0, sticky='news')

def ringup1(ring1):
    global ringstate
    global time_rd
    if GPIO.event_detected(ring1):
        if GPIO.input(ring1) == True:
            ringstate[0] += 1
        else:
```

```
            time_rd[0] = time.time()

GPIO.add_event_detect(ring1, GPIO.BOTH, callback=ringup1)

def ringup2(ring2):
    global ringstate
    global time_rd
    if GPIO.event_detected(ring2):
        if GPIO.input(ring2) == True:
            ringstate[1] += 1
        else:
            time_rd[1] = time.time()

GPIO.add_event_detect(ring2, GPIO.BOTH, callback=ringup2)

def ringup3(ring3):
    global ringstate
    global time_rd
    if GPIO.event_detected(ring3):
        if GPIO.input(ring3) == True:
            ringstate[2] += 1
        else:
            time_rd[2] = time.time()

GPIO.add_event_detect(ring3, GPIO.BOTH, callback=ringup3)

def ringup4(ring4):
    global ringstate
    global time_rd
    if GPIO.event_detected(ring4):
        if GPIO.input(ring4) == True:
            ringstate[3] += 1
        else:
            time_rd[3] = time.time()

GPIO.add_event_detect(ring4, GPIO.BOTH, callback=ringup4)

def hookup(hook):
    global hookstate
    global time_hd
    if GPIO.event_detected(hook):
        if GPIO.input(hook) == False:
            time_hd = time.time()
        else:
            hookstate = 1

GPIO.add_event_detect(hook, GPIO.BOTH, callback=hookup)

def hookdn():
    global hook
    global hookstate
    global relay1
    global relay1_state
    global time_hd
    global connect_time
```

```
    global ringstate
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(hook, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    time_now = time.time()
    if (time_now - time_hd) > 1:
        if GPIO.input(hook) == False:
            hookstate = False
    time_now2 = time.time()
    if hookstate == True and (time_now2 - connect_time) > 10:
        for i in range(4):
            if relay1_state[i] == True and ringstate[i] == False:
                show_frame(StartPage)
                relay1_state[i] = False
                break

def ringdn(i):
    global ringstate
    global time_rd
    if i == 21:
        j = 0
    elif i == 18:
        j = 1
    elif i == 22:
        j = 2
    elif i == 4:
        j = 3
    now_time = time.time()
    if abs(now_time - time_rd[j]) >= 6:
        ringstate[j] = 0

def config_relay():
    global ringstate
    global relay1_state
    global relay2_state
    global isconn
    if ringstate[0] > 2:
        if relay1_state[1] == True or relay1_state[2] == True or relay1_state[3] ==
True:
            label3.config(text="Call Detected on Line 1, put on Hold")
            label3.grid(row=6, column=0, columnspan=1, rowspan=2)
            label4.config(text="Call Detected on Line 1, put on Hold")
            label4.grid(row=6, column=0, columnspan=1, rowspan=2)
            label5.config(text="Call Detected on Line 1, put on Hold")
            label5.grid(row=6, column=0, columnspan=1, rowspan=2)
            relay2_state[0] = True
            relay1_state[0] = False
            busy_time[0] = time.time()
        else:
            show_frame(lineOne)
            connect_time = time.time()
            relay1_state[0] = True
            relay2_state[0] = False
    elif ringstate[1] > 2:
        if relay1_state[0] == True or relay1_state[2] == True or relay1_state[3] ==
True:
```

```
            label2.config(text="Call Detected on Line 2, put on Hold")
            label2.grid(row=6, column=0, columnspan=1, rowspan=2)
            label4.config(text="Call Detected on Line 2, put on Hold")
            label4.grid(row=6, column=0, columnspan=1, rowspan=2)
            label5.config(text="Call Detected on Line 2, put on Hold")
            label5.grid(row=6, column=0, columnspan=1, rowspan=2)
            relay2_state[1] = True
            relay1_state[1] = False
            busy_time[1] = time.time()
        else:
            show_frame(lineTwo)
            connect_time = time.time()
            relay1_state[1] = True
            relay2_state[1] = False
    elif ringstate[2] > 2:
        if relay1_state[1] == True or relay1_state[0] == True or relay1_state[3] ==
True:
            label2.config(text="Call Detected on Line 3, put on Hold")
            label2.grid(row=6, column=0, columnspan=1, rowspan=2)
            label3.config(text="Call Detected on Line 3, put on Hold")
            label3.grid(row=6, column=0, columnspan=1, rowspan=2)
            label5.config(text="Call Detected on Line 3, put on Hold")
            label5.grid(row=6, column=0, columnspan=1, rowspan=2)
            relay2_state[2] = True
            relay1_state[2] = False
            busy_time[2] = time.time()
        else:
            show_frame(lineThree)
            connect_time = time.time()
            relay1_state[2] = True
            relay2_state[2] = False
    elif ringstate[3] > 2:
        if relay1_state[1] == True or relay1_state[2] == True or relay1_state[0] ==
True:
            relay2_state[3] = True
            relay1_state[3] = False
            busy_time[3] = time.time()
            label2.config(text="Call Detected on Line 4, put on Hold")
            label2.grid(row=6, column=0, columnspan=1, rowspan=2)
            label3.config(text="Call Detected on Line 4, put on Hold")
            label3.grid(row=6, column=0, columnspan=1, rowspan=2)
            label4.config(text="Call Detected on Line 4, put on Hold")
            label4.grid(row=6, column=0, columnspan=1, rowspan=2)
        else:
            show_frame(lineFour)
            connect_time = time.time()
            relay1_state[3] = True
            relay2_state[3] = False

def discon_relay2(line = 4):
    global busy_time
    if line != 4:
        relay1_state[line] = True
        time.sleep(0.2)
        relay2_state[line] = False
```

```
    else:
        for i in range(4):
            if time.time() - busy_time[i] > 30:
                relay2_state[i] = False

def output_allot():
    global relay1_state
    global relay2_state
    for i in range(4):
        GPIO.output(relay1[i], relay1_state[i])
        GPIO.output(relay2[i], relay2_state[i])

def global_runn():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(hook, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(ring1, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(ring2, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(ring3, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(ring4, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(relay1[0], GPIO.OUT)
    GPIO.setup(relay1[1], GPIO.OUT)
    GPIO.setup(relay1[2], GPIO.OUT)
    GPIO.setup(relay1[3], GPIO.OUT)
    GPIO.setup(relay2[0], GPIO.OUT)
    GPIO.setup(relay2[1], GPIO.OUT)
    GPIO.setup(relay2[2], GPIO.OUT)
    GPIO.setup(relay2[3], GPIO.OUT)
    while True:
        hookdn()
        ringdn(ring1)
        ringdn(ring2)
        ringdn(ring3)
        ringdn(ring4)
        config_relay()
        discon_relay2()
        output_allot()

start_new_thread(global_runn, ())

def on_closing():
    GPIO.cleanup()
    app.destroy()

show_frame(StartPage)
app.attributes('-fullscreen', True)
app.protocol("WM_DELETE_WINDOW", on_closing)
app.mainloop()
```

## References


[1][1] Polyzois, C.A. & Purdy, Kermit & Yang, Ping-Fai & Shrader, D. & Sinnreich, H. & Menard, F. & Schulzrinne, Henning. (1999). From POTS to PANS: a commentary on the evolution to Internet telephony. Internet Computing, IEEE. 3. 83 - 91. 10.1109/4236.769426.

[2][2] N. Amitay, "Distributed switching and control with fast resource assignment/handoff for personal communications systems," in *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 6, pp. 842-849, Aug. 1993.

[3] Cole, Marion. Telecommunications. Prentice Hall, 1999.

[4] "Packet Switching Network Access Protocols for Multi-Media Packet Communications Microprocessors and Microsystems, vol. 10, no. 7, 1986, p. 398., doi:10.1016/0141-9331(86)90366-2.

[5] V. B. J. G., Signaling in todays telecommunication networks. New York: Wiley, 1997.

[6] M. Eick and H. Graeb, "A versatile structural analysis method for analog, digital and mixed-signal circuits," 2012 International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), 2012.

[7] "Specification for simple telephones for connection to public switched telephone networks run by certain public telecommunication operators."

[8] Hiraguri, Shigeto. "Recent Research and Development of Signaling and Telecommunications Technologies." Quarterly Report of RTRI, vol. 56, no. 3, 2015, pp. 160–163., doi:10.2219/rtriqr.56.160.

[9] B. Balon and M. Simić, "Using Raspberry Pi Computers in Education," 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 2019, pp. 671-676, doi: 10.23919/MIPRO.2019.8756967.

[10] “Tkinter GUI.” Python by Example, 2019, pp. 110–123., doi:10.1017/9781108591942.019.

[11] Tran, Nhat-Phuong, et al. “High Throughput Parallel Implementation of Aho-Corasick Algorithm on a GPU.” 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, 2013, doi:10.1109/ipdpsw.2013.116.

[12] Perry, M., and A. Nilsson. “Performance Analysis of Automatic Call Distributors with Call Waiting Lamps.” Proceedings of GLOBECOM 95, doi:10.1109/glocom.1995.502694.