RADIO SPECTRUM MANAGEMENT SYSTEM

By

Capt Faisal Nasim

Capt Mohsin Bashir

PC Umar Farooq

PC Tahir Noor

Submitted to the Faculty of Computer Science
National University of Sciences and Technology, Rawalpindi in partial fulfillment for the requirements of a B.E Degree in Computer Software Engineering
AUGUST 2009

ABSTRACT

RADIO SPECTRUM MANAGEMENT SYSTEM

R&S ESMB is a monitoring and test receiver for radio signals detection and their monitoring tasks. This receiver is capable of receiving signals ranging from 9.0KHZ to 3.0GHZ. These signals possess valuable information which cannot be extracted due to usage of various types of modulations, source encoding techniques, scrambling and encryption. These all things come in such a variety that any heuristic approach cannot be used and only a careful analysis of signals can reveal the information inside it. For this analysis, we need to receive these signals inside computing machine in a soft format for further processing.

Radio Spectrum Management System is designed to communicate with R&S ESMB radio receiver. This software connects to the R&S ESMB radio receiver via Ethernet 10Base-T interface through TCP/IP and gathers data sent from R&S ESMB radio receiver to computer, stores the received data, implements a control module for automated control of R&S ESMB radio receiver and plays real time audio data received from receiver.

DECLARATION

No portion of the work presented in this dissertation has been submitted in support of any other award or qualification either at this institution or elsewhere.

DEDICATION

In the name of Allah, the Most Merciful, the Most Beneficent

To our parents, without whose unflinching support and unstinting cooperation, a work of this magnitude would not have been possible

ACKNOWLEDGMENTS

We are thankful to Almighty Allah for blessing us with the strength and courage to undertake and complete this project.

We are grateful to **MILITARY COLLEGE OF SIGNALS (MCS)** as it had been our foundation and has provided us opportunity to undertake this project.

We gratefully recognize the continuous supervision and motivation provided to us by our Project Supervisor, Maj Dr NAVEED IQBAL RAO (Image Processing Centre, MCS-NUST). A lot of effort and contribution has been put in by our co-supervisor Lt Col MANSOOR SINDHU (R&D, MCS-NUST). We are really thankful to Maj ZULFIQAR MEHDI (EW Dte) and Lec AHMED MEQEEM SHERI for putting their efforts to facilitate us. We are grateful to the staff of R&D department, MCS for being very helpful in this project. We deeply treasure the support and tolerance that we received from our friends for their useful suggestions that helped us in completion of this project. We are also deeply obliged to our families for their never ending patience and support for our mental peace and to our parents for the strength that they gave us through their prayers.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page Number**

LIST OF FIGURES

Figure **Page Number**

Chapter 1

1 Introduction

1.1 Preface:

R&S ESMB is a monitoring and test receiver for radio signals detection and their monitoring tasks. This receiver can be connected to the computer via 10BASE-T and RS32 interfaces. This receiver is capable of receiving signals ranging from 9.0KHZ to 3.0GHZ and. These signals possess valuable information which cannot be extracted due to usage of various types of modulations, source encoding techniques, Scrambling and encryption. These all things come in such a variety that any heuristic approach cannot be used and only a careful analysis of signals can reveal the information inside it. For this analysis, we need to receive these signals inside computing machine in a soft format for further processing.

1.2 Project Vision:

The basic idea behind this project is to control the R&S ESMB radio receiver remotely. Another objective is to facilitate the operator by providing him easy interface to interact with the receiver. Reception of radio signals inside computer, extraction of valuable information from these radio signals, and storage of the important information retrieved from radio signals is also a main objective .This can be done by implementing software which makes it possible for operator to configure and control the receiver without operating the front panel of the R&S ESMB receiver. This project is hence an effort to contribute towards achieving this goal.

1.3 Proposed Solution:

The proposed solution addresses the following problem:

- Connection of the receiver to the computer via 10BASE-T interface through TCP/IP
- Reception of radio signals inside computer
- Displaying information inside radio signals to the operator
- Providing a facility to control the receiver through the computer
- Making it possible to store the data received from receiver
- Retrieving the stored data

1.4 Aim of the project:

The aim of the project is to effectively deal with the scenario where an operator is operating R&S ESMB radio receiver. In the recommended solution, the operator instead of controlling and configuring the receiver from its front panel, he will be able to do this by sitting on the computer. The operator with the help of proposed solution will be able to receive the radio signals inside computer. The data received from receiver can be viewed by the operator. There is also a facility to store and retrieve the received data. Operator can also listen the audio coming from the receiver.

1.5 Organization of Project Report:

The project report has been drafted carefully deciding the sequence to be followed. After the introduction section, the report incorporates the Literature Review chapter summarizing the text studied before and during the project's execution. Subsequently, the System Analysis chapter comes which includes the major interface, functional and non-functional requirements of the system. It also incorporates the use case diagram and the sequence diagram of the system. Next is the System Design chapter comprising of the architectural diagram, data flow diagram and class diagram. Following this the report includes the implementation chapter identifying and elucidating the classes which are implemented. Then is the testing chapter incorporating the testing process employed to test the system and the results that were obtained. The next chapter then discusses the work that can be done in future to further enhance the system and ultimately this chapter wraps the report.

Chapter 2

2 Literature Review

2.1 Introduction:

This chapter is an effort to summarize the material that has been studied throughout the project. The literature studied can be divided into three main parts, introduction to R&S ESMB, Device Model and Command Processing and Status reporting system. Going through this section will aid in comprehending the later chapters.

2.2 Introduction to R&S ESMB:

R&S ESMB is a monitoring and test receiver for radio signals detection and their monitoring tasks. This receiver is capable of receiving signals ranging from 9.0KHZ to 3.0GHZ. It can be connected to the computer by Ethernet 10Base-T interface through TCP/IP. The front panel of the receiver is shown below.

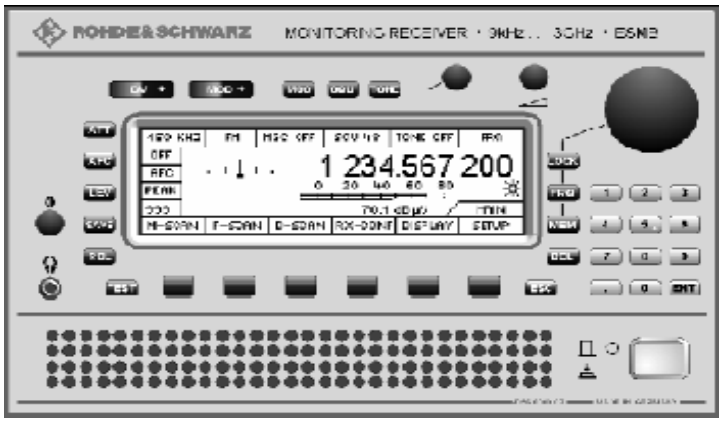


Figure 2-1: R&S ESMB Front Panel [1]

2.3 Device Model and Command Processing:

The following figure shows the basic structure of the unit under firmware aspects.

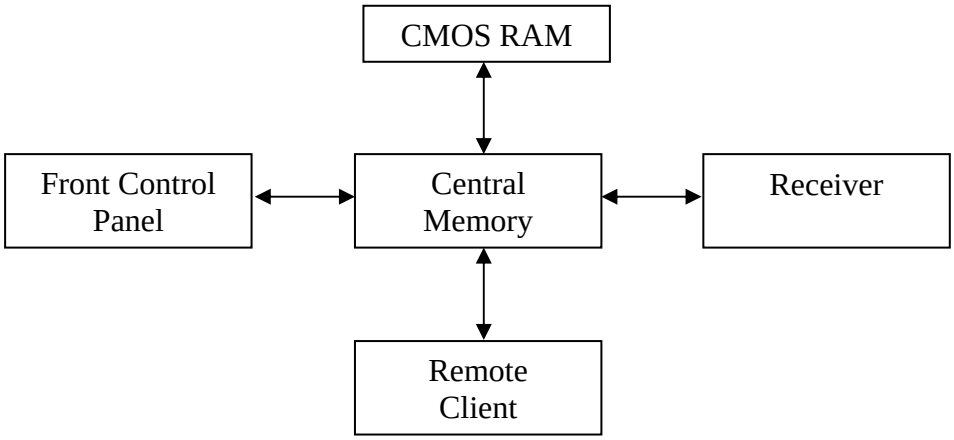


Figure 2-2: Device Model with Remote Control

The actual receiver is isolated from the front control panel and the remote control units by a central data memory. This memory is at the core of the ESMB firmware and deals with the following tasks:

- Administration of connected modules (receiver, front control panel, remote clients)
- Making data available to the receiver (e.g. receive frequency, scan parameters)
- Sequentialization of settings for simultaneous manual and remote control
- Sending messages on parameter changes to all modules
- Storing data in the CMOS RAM for protection against power failure

As mentioned above, the receiver can be controlled from the front panel and one or several remote control units – the remote clients – simultaneously (competitive control). Upon system start, the front control panel and the receiver are logged in to the data memory automatically. These modules are therefore always connected. The remote clients are logged in if a host computer sets up a link to the ESMB [1].

The receiver obtains the required data (receive frequency, bandwidth, etc.) from the memory. It has no data storage facility of its own and therefore has direct access to the central memory. Due to the principle of competitive control, different clients can modify the same parameters. The central memory sequentializes the access

procedures (last client wins) and sends messages to the other users that a parameter has been changed [1].

Remote client 1 modifies the frequency value. The central memory signals to the receiver that a new frequency is to be set. The front control panel is supplied with the new frequency. Remote client 2 (if connected) receives a modification report [1].

If the receive frequency is changed by the receiver due to a scanning procedure or an AFC correction, this is reported to remote clients 1 and 2. The front control panel receives the new receive frequency and displays it [1].

A large number of parameters have to be stored power-failure-proof. This is ensured by the central memory: it stores all modifications of the relevant parameters in the CMOS RAM and assigns them to checksums. Upon switching on the unit, the checksums are checked, and the data are retrieved from the CMOS RAM or default values are used [1].

2.3.1 Remote Client:

Structure of a remote client within firmware is shown in the following figure

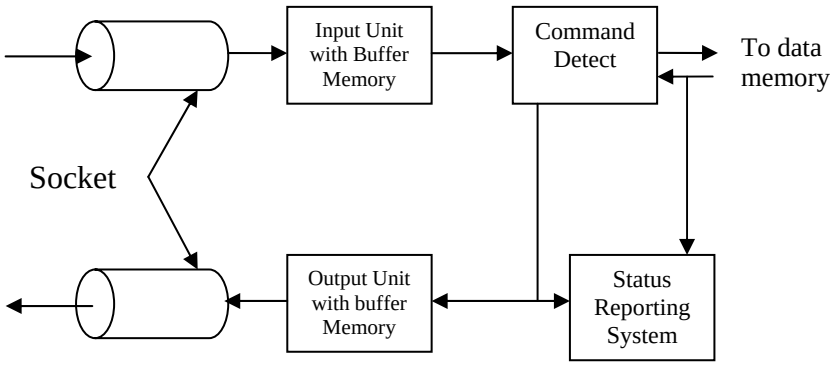


Figure 2-3: Structure of a remote client within the firmware

- **Sockets**

The remote clients are connected to the host computer by so-called sockets. These are logic point-to point links that are independent of the transmission medium used.

Sockets are based on the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP, not used in ESMB). These two protocols are in turn based on the Internet Protocol (IP). Following figure shows the layer model of the sockets [1].

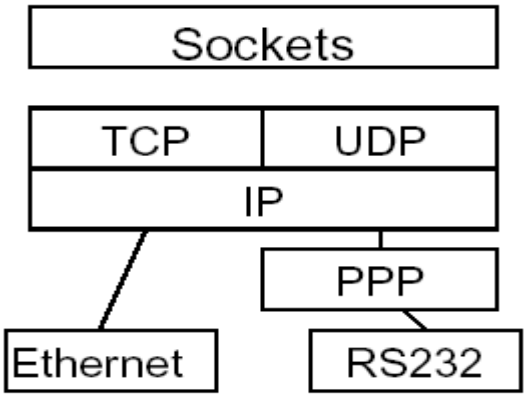


Figure 2-4: Layer Model of sockets

The transmission media are located beneath the IP layer. IF the ESMB is fitted with an RS232- compatible interface (option) it is coupled to the IP layer by means of the Point-to-Point Protocol (PPP). This is necessary because IP is a packet-oriented protocol. This is not necessary if the LAN interface (standard) is used, because the Ethernet protocol of this interface is already packet-oriented [1].

The use of sockets has several advantages:

1. The protocols used (PPP, IP, TCP, UDP) are standardized and implemented on all customary operating systems (WindowsNT, Windows95, Windows 3.1, UNIX, SunOS)
2. TCP links are protected against transmission errors
3. Host software can be generated independent of the transmission medium used (LAN or RS232)
4. Several logic links may use the same transmission medium.
5. IP routing enables access also to remote units also over great distances (e.g. via the Internet)

When the unit is started, a so-called list socket is generated. It functions as the unit’s "receptionist". Each host wishing to remote-control the ESMB has to log in with the list socket first. The list socket then generates a new remote client and allocates the link to a new socket so the list socket remains free to receive further hosts. For login at the list socket, the host needs to have the address and port number of the unit. This can be set in the Setup Remote menu [1].

- **Input Unit**

Data transmission via sockets is packet-oriented. Each packet received is handed over to command recognition [1].

- **Command Detect**

Command detect analyzes the data received from the input unit. Data are processed in the sequence they have been received. The data received consist of strings that have to be in accordance with the SCPI standard. The SCPI standard is based on the IEEE 488 standard. Normally, this standard only applies to IEC/IEEE bus (also referred to as IEC625, HPIB or GPIB). Another IEEE standard, IEEE 1174, is a supplement to IEEE 488, making it applicable also to LAN and serial links (RS232). The ESMB uses this standard as a basis for SCPI commands via sockets.

Each identified setting command contained in an SCPI string is first stored in a buffer memory. Only a <Program Message Terminator> (line feed) or a query command will cause the setting commands to be sent to the data memory, where they are checked for consistency. If the commands are consistent, they will be executed at once, and the other modules will be informed. Query commands generate a request to the memory. The memory sends back the data, which will then be processed according to the SCPI standard by the command detect. Finally, the SCPI response strings are sent to the output unit [1].

- **Output Unit**

The output unit collects all data in the output buffer that were generated in response to query commands. If the command detect identifies the end of an SCPI command (by the <Program Message Terminators>), it causes the output unit to send the data in the output buffer to the host computer via the socket [1].

- **Status Reporting System**

The Status Reporting System gathers information on the device status and makes it available to the output unit on request. The Status Reporting System may be used for messaging asynchronous events (e.g. error statuses, availability of results, data modifications by other users, etc.) to the host computer [1].

2.3.2 Data Memory:

This figure shows the classification of data into data groups. These groups are also reflected by the Status Reporting System of the remote clients in the extension register status [1].

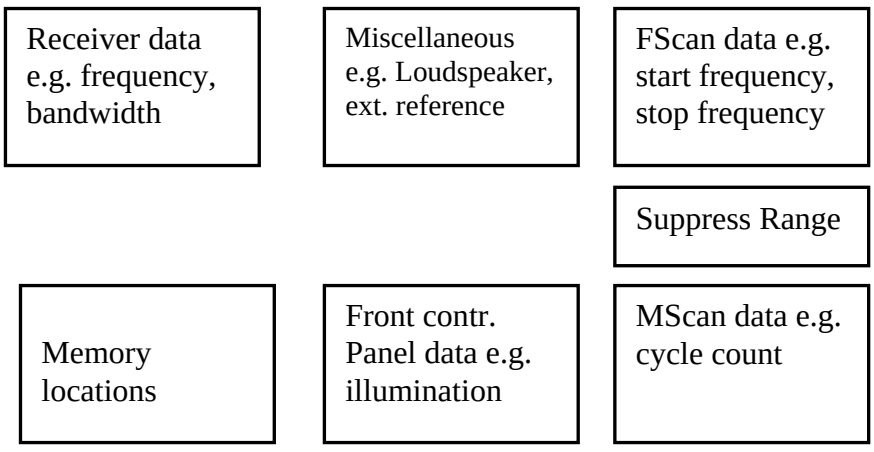


Figure 2-5: Data Classification into Data Groups

2.4 Conclusion:

This chapter incorporated the explanation of all the significant technical terms which are used in coming chapters. This chapter provides comprehensive information necessary for the understanding of this project.

Chapter 3

3 System Analysis

3.1 Introduction:

This chapter covers the system analysis phase of the project. In this phase, first of all scope of the project is presented as it's clear definition and understanding is needed for the absolute comprehension of the system's requirements' specification phase, including major functional and non-functional requirements, is described. The requirement specification phase is then followed by use case diagram for the better understanding of the system analysis phase of the project.

3.2 Project Scope:

Radio Spectrum Management System Using R&S ESMB is designed to communicate with R&S ESMB device. This system will establish a connection to the receiver via 10BASE-T interface through TCP/IP protocol. The system gets UDP datagram from the receiver and extracts meaningful information out of this datagram. Mechanism of data storage is devised so that important data can be stored on hard disk for record and future data analysis. Automated control of the receiver is implemented so that receiver can be configured through computer without operating the front panel of the receiver and in order to achieve this; command set of the receiver is implemented. A sound module is also implemented so that operator can listen and store the audio data.

3.3 Requirements Specification:

The requirements specifications of the system include its external interface requirements including user and software interface requirements. It also involves the

analysis of the major functional requirements including the main functionalities that the system is expected to deliver. Analysis of main non-functional requirements is also important as it tends to give an idea about the characteristics of the system such as accuracy, scalability etc.

3.3.1 External Interface Requirements:

Requirements which include the interaction of the system with the external requirement e.g. user interface through which the user interacts with the system and software interfaces means the software tools which are employed in the system.

3.3.1.1 User Interfaces:

Several GUIs are required to enable the operator to interact with the system for example; a GUI to connect the receiver to the computer, a GUI to control the receiver and another GUI to save the data in receiver's memory locations.

3.3.1.2 Software Interfaces:

The Project employs opensuse11.0 as the operating system and software tools such as GTKMM and KDEVELOP are used. GTKMM is a library for designing the GUI and KDEVELOP is an IDE for C++.

3.3.2 Major Functional Requirements:

Functional requirements means the core functionalities that the system is meant to deliver, e.g. separation of UDP datagram into meaningful information, controlling the receiver from the computer rather than receiver's front panel, playing the audio data coming from the receiver and storing the data into system's hard disk and then retrieving this data. These functionalities are illustrated in more details under the following headings.

3.3.2.1 Controlling the receiver remotely:

The ESMB receiver has the capability to be remotely controlled from a computer through the TCP/IP connection. The receiver has a standard command set through which it is controlled e.g. if we want to set the frequency of the receiver to 106.20 through remote control client, we will use the following command, encapsulate it in a TCP packet and send it to the receiver.

FREQuency 106.20 MHz

The frequency of ESMB receiver will be set to 106.20 MHz. Furthermore, to report any changes in the receiver to the remote client, it has a complete status reporting system through which it can send interrupts to client.

By using this command set and status reporting system, a GUI based control module for the receiver will be designed. The command set of the receiver has a lot of commands. The GUI of the control module will not implement all the commands rather only those commands will be implemented that forms the basic structure for controlling the receiver e.g. setting of the frequency remotely is necessary for remote client and its commands will be implemented but command for adjusting the brightness of the receiver's display will not be the part of the control software and which are not necessary at initial stage.

3.3.2.2 Data Parsing:

The receiver has the capability to send the monitored data to the client computer through UDP. The remote client will implement the UDP server and listens for the UDP data sent from the ESMB receiver. This UDP connection will be in addition to the TCP connection for controlling the ESMB receiver.

The ESMB receiver monitors following types of data

- FSCAN
- DSCAN
- MSCAN
- Audio
- GSM
- CW
- IFPAN
- FASTLAVCW
- LIST

One UDP datagram encloses only one type of data. The device monitors one type of data at a time enclosed in a UDP datagram and sends to the nominated PC. When the ESMB receiver is switched from one mode to another mode of data during send process e.g. from FSCAN to Audio mode then UDP datagram contains data for that mode. The data is enclosed by the device in UDP datagram as per standard set in SCPI.

The functioning of this module is to check whether the UDP datagram received is of RSMB receiver. After confirmation about the data, it will parse data i.e. it will check which type of data the UDP datagram contains.

3.3.2.3 Playing real time audio data:

The ESMB receiver uses 12 Audio modes to send Audio data as shown in the following table

Mode	Bits per sample	Channels	Data rate [kbyte/s]
0	-	-	0
1	16	2	128
2	16	1	64
3	8	2	64
4	8	1	32
5	16	2	64
6	16	1	32
7	8	2	32
8	8	1	16
9	16	2	32
10	16	1	16
11	8	2	16
12	8	1	8

Table 3.1: Audio Modes

It should be able to pick up the mode of the ESMB receiver automatically and configure its computer's sound card accordingly for the play back of raw Audio data.

3.3.2.4 Data Storage:

The system should have the mechanism to handle different types of data and save individual types of data separately in different files for future analysis. This requirement should be able to handle following issues

- When ESMB receiver is delivering one type of UDP data to the hosting computer and is switched from one mode to another mode, the software system should be able to handle new type of data and must be able to save every type of data by tagging it.
- There should be a mechanism so that the stored data can be recognized i.e. if user requires the data as per date/time or type of data he/she should be able to find target files.

3.3.3 Major Non-Functional Requirements:

3.3.3.1 Security

The system should be secure in a sense that the information should be received by the intended user only.

3.3.3.2 Reliability

The system should be reliable in a sense that the system should provide the users with the required functionality round the clock.

3.3.3.3 Maintainability

The system will be made maintainable so that incase of error or the user's complaints the system might be changed to satisfy the new needs or to correct the errors.

3.3.3.4 Reusability

The system will be made reusable by making the application open source.

3.3.3.5 User friendly GUI

The user interface for the application should be friendly and interactive.

3.4 Use case Diagram:

Following is the use case diagram of the system

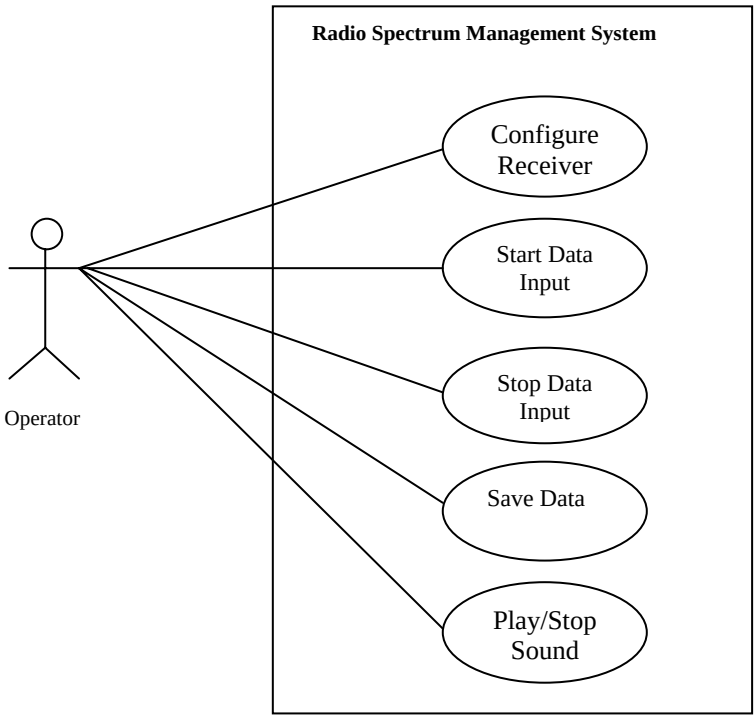


Figure 3.1: Use Case Diagram

Brief Description:

- **Configure Receiver:**

The operator will be able to configure the receiver through the control module. The configuration will include such steps as setting of frequency changing bandwidth and increasing or decreasing volume etc.

- **Start Data Input:**

Data will be collected at computer which is sent from receiver. This is real time data, which can be of any type.

- **Stop Data Input:**

The data coming from the receiver and collected by the computer will longer be coming from the receiver.

- **Play Sound:**

Operator sitting on the computer listen real time audio data coming from the receiver.

- **Stop Sound:**

Operator can stop the audio which is being played and listened on the computer connected to the receiver.

- **Save Data:**

The data coming from the receiver can be stored on the computer's hard disk, so that it can be available for future analysis.

3.5 Conclusion:

The system analysis of the project has been covered in this chapter. The scope of the project has been revised for the clear understanding of the requirements, key functional and non-functional requirements have been enumerated, the use case diagram showing the major actors and their actions have been included. This chapter has been written comprehensively so that the fore coming design chapter becomes easy to comprehend.

Chapter 4

4 System Design

4.1 Introduction:

System design is a very important phase in the software development process. The succeeding implementation phase is performed taking into consideration the design constraints. This chapter begins by presenting the high level design of the project showing the main modules of the system without including much detail. Next the low level design is incorporated elucidating the modules identified in the high level design. It is then followed by the data flow diagram of the project. Class diagram is also included focusing on the implemented classes, their attributes and their relationships with each other.

4.2 Architectural Diagram:

The architectural diagram of the system is specified in Figure 4.1. It illustrates the basic diagram and design of the project.

As depicted in Figure 4.1, the communication between the receiver and the parser is unidirectional as the parser can only receive data from receiver. Parser sends the parsed data to the GUI and also to the Data Base. Data from the Data Base can be viewed in GUI. Similarly the data can be stored in the data base through GUI. Communication between GUI and control module is unidirectional and is from GUI to control. Operator can send commands to receiver through GUI and these commands go to control module which then send them to receiver.

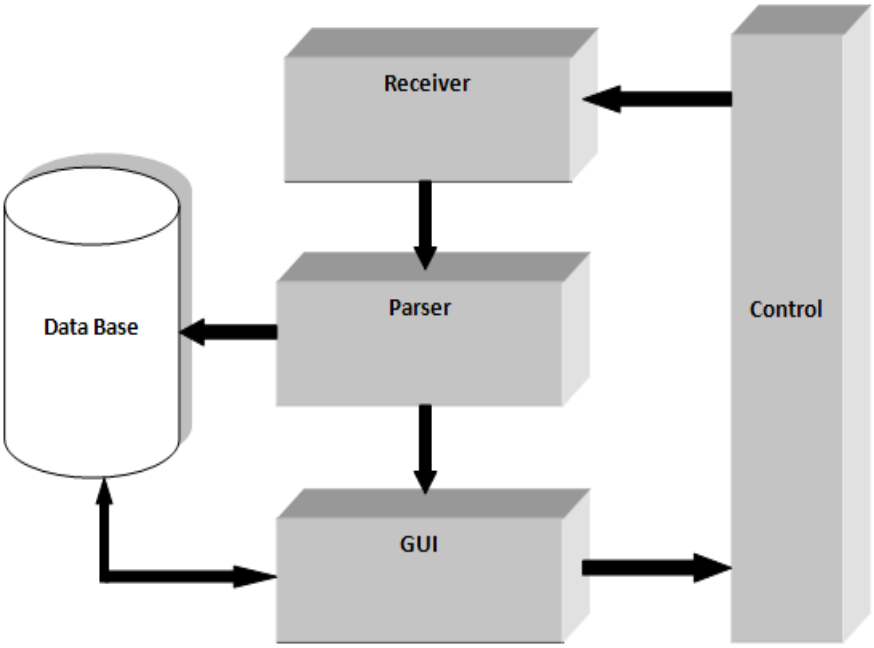


Figure 4-1: Architectural Diagram

4.3 High Level Design:

The high level design of the project is shown in the Figure 4.2

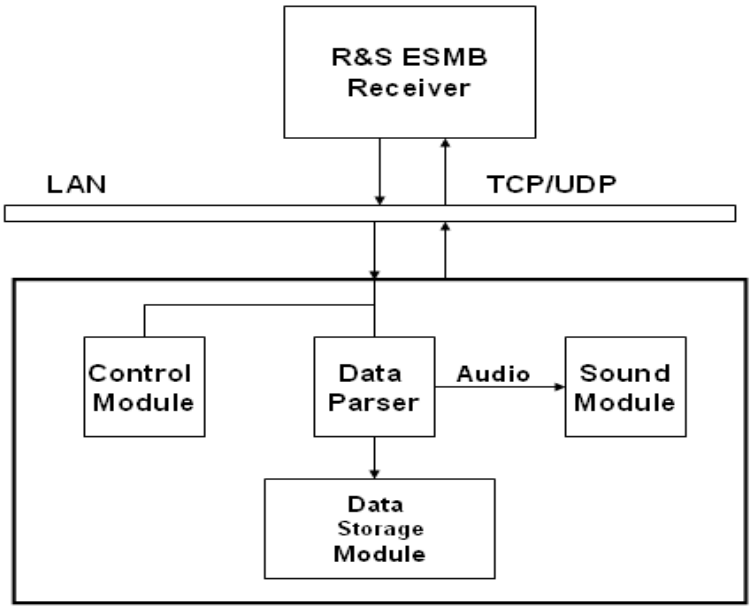


Figure 4-2: High Level Design

4.3.1 Control Module:

The ESMB receiver has the capability to be remotely controlled from a computer through the TCP/IP connection. The receiver has a standard command set through which it is controlled e.g. if we want to set the frequency of the receiver to 106.20 through remote control client, we will use the following command, encapsulate it in a TCP packet and send it to the receiver.

FREQuency 106.20 MHz

The frequency of ESMB receiver will be set to 106.20 MHz. Furthermore, to report any changes in the receiver to the remote client, it has a complete status reporting system through which it can send interrupts to client.

By using this command set and status reporting system, a GUI based control module for the receiver will be designed through which receiver can be controlled remotely. The command set of the receiver has a lot of commands. The GUI of the control module will not implement all the commands rather only those commands will be implemented that forms the basic structure for controlling the receiver e.g. setting of the frequency remotely is necessary for remote client and its commands will be implemented but command for adjusting the brightness of the receiver's display will not be the part of the control software and which are not necessary at initial stage.

4.3.2 Data Parser Module:

The receiver has the capability to send the monitored data to the client computer through UDP. The remote client will implement the UDP server and listens for the UDP data sent from the ESMB receiver. This UDP connection will be in addition to the TCP connection for controlling the ESMB receiver.

The ESMB receiver monitors following types of data

- FSCAN
- DSCAN
- MSCAN
- Audio
- CW
- IFPAN
- FASTLAVCW
- LIST

One UDP datagram encloses only one type of data. The device monitors one type of data at a time enclosed in a UDP datagram and sends to the nominated PC. When the ESMB receiver is switched from one mode to another mode of data during send process e.g. from FSCAN to Audio mode then UDP datagram contains data for that mode. The data is enclosed by the device in UDP datagram as per standard set in SCPI.

The functioning of this module is to check whether the UDP datagram received is of RSMB receiver. After confirmation about the data, it will parse data i.e. it will check which type of data the UDP datagram contains.

4.3.3 Data Storage Module:

The system handles different types of data and saves individual types of data separately in different files for future analysis.

4.3.4 Sound Module:

The ESMB receiver uses 12 Audio modes to send Audio data as shown in the following table

Mode	Bits per sample	Channels	Data rate [kbyte/s]
0	-	-	0
1	16	2	128
2	16	1	64
3	8	2	64
4	8	1	32
5	16	2	64
6	16	1	32
7	8	2	32
8	8	1	16
9	16	2	32
10	16	1	16
11	8	2	16
12	8	1	8

Table 4.1: Audio Modes

It should be able to pick up the mode of the ESMB receiver automatically and configure its computer's sound card accordingly for the play back of raw Audio data.

4.4 Low Level Design:

Following diagram shows the low level diagram of the system:

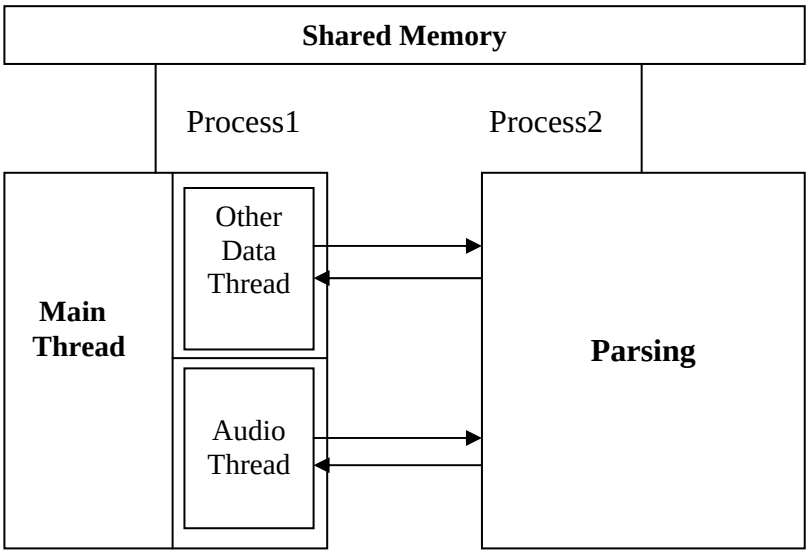


Figure 4-3: Low Level Design

This software is mainly divided in two processes. The functionality of receiving and parsing the incoming data is delegated to one process. The other process has the responsibility of gathering that parsed data, viewing the stored data and playing the audio data. These two processes communicate through shared memory i.e. same address space is accessible to both the processes. One process writes the parsed data to the shared memory while other process reads that data from the same memory. Shared memory is the fastest mean of IPC available in Linux. The process that receives and parses the data is implemented as a single thread, while the other process is implemented as four threads i.e. one main thread and three sub-threads.

Main thread has the responsibility to display the main window and other GUI elements of the application. One thread runs continuously to receive any audio data written to the shared memory. While other thread runs continuously to receive any data other than audio data written to the shared memory. Fourth thread has the responsibility to play the audio data.

4.5 Class Diagram:

Class diagram of the system is illustrated in Figure 4.4. In this diagram the classes that have been implemented are shown along with the inherited classes and the implemented interfaces, along with identification of their relationships with each other.

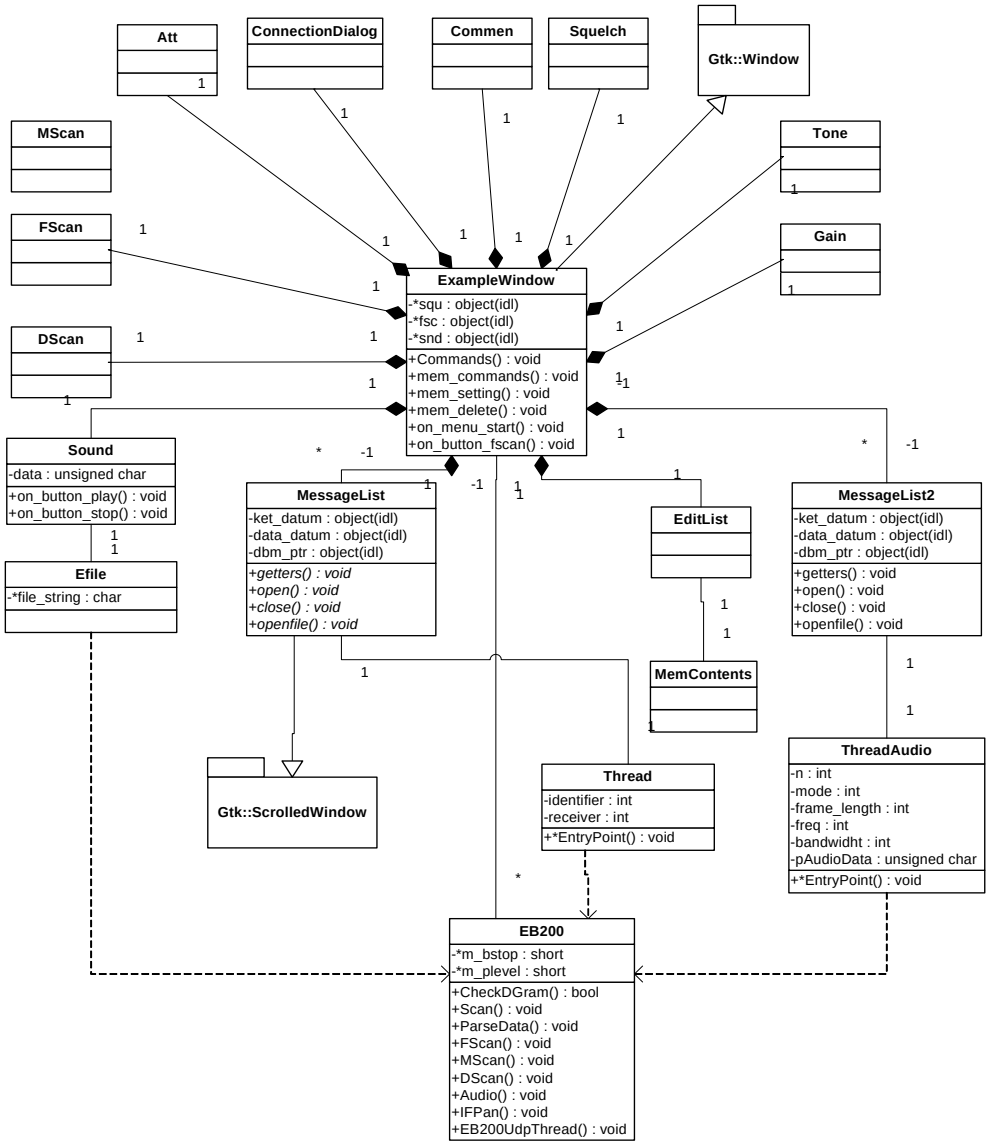


Figure 4-4: Class Diagram

4.6 Data Flow Diagram:

The data flow diagram of the system is given in Figure 4.5. First of all the receiver sends the data which is intended to be received by the data parser. Parser receives the real time data being sent by the receiver.

After reception of data by parser from receiver, parser separates data into two portions based on its data type. These portions are audio data and other data types. Parser modules sends audio data to the sound playing module and other data to the function which is supposed to provide the functionality of data view and data storage. Data send to the sound playing module is played. And data received by the function responsible for data viewing and storage are individually send to the view in GUI and data storage functions which treats data accordingly.

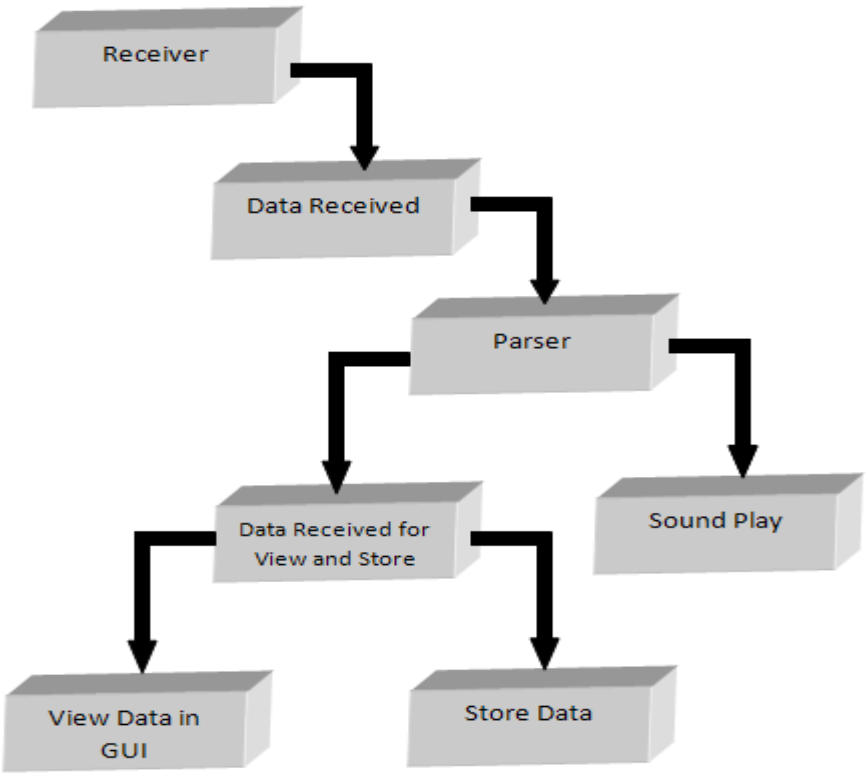


Figure 4-5: Data Flow Diagram

4.7 Conclusion:

This chapter presented the architecture for Radio Spectrum Management System. It has incorporated the high level design, low level design, data flow diagram and class diagram of the system. Four basic modules have been identified which are Data parser, Control, Sound Playing and Data storage modules. The system design chapter included details that are very important in comprehending well the upcoming implementation chapter.

Chapter 5

5 Implementation

5.1 Introduction:

This chapter presents the implementation details of the project. The coding is done in the C++ and mainly object-oriented approach has been adopted. There are logically four major modules of this project.

- Data Parsing Module
- Control Module
- Sound Module
- Data Storage Module

The data sent by the receiver will be received by the data parsing module. The function of the data parsing module is to parse the incoming data i.e. data sent by the receiver is composed of **9** types of data and is encapsulated in a UDP datagram. Data parser separates each type of data and sends that data to its respective handling functions. Control Module controls the receiver remotely. Sounds module receives the audio data and plays it. Data storage module stores the data sent by the receiver.

5.2 Implementation language:

The implementation language which has been used to develop this project is C++, which is an object oriented language. The system is developed in Linux (open SUSE 11.0) operating system. For the implementation and design of GUI, Gtkmm is used. Basically Gtkmm is a C++ wrapper of Gtk+ library which is implemented in C. Gtk+ is used for GUI development in GNOME desktop. Threads being used in this software are implemented using **pthread** library. This **pthread** library is supplied with every major Linux distribution. The database system employed in the software is

gdbm. This is file system for database system. It is also supplied with every major Linux distribution.

5.3 Software Architecture:

Following diagram shows the software architecture of the system:

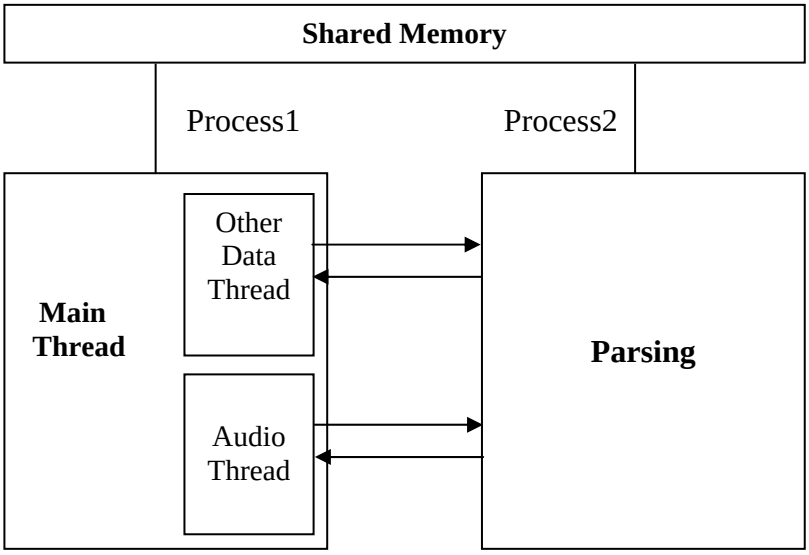


Figure 5-1: Software Architecture

This software is mainly divided in two processes. The functionality of receiving and parsing the incoming data is delegated to one process. The other process has the responsibility of gathering that parsed data, viewing the stored data and playing the audio data. These two processes communicate through shared memory i.e. same address space is accessible to both the processes. One process writes the parsed data to the shared memory while other process reads that data from the same memory. Shared memory is the fastest mean of IPC available in Linux. The process that receives and parses the data is implemented as a single thread, while the other process is implemented as four threads i.e. one main thread and three sub-threads.

Main thread has the responsibility to display the main window and other GUI elements of the application. One thread runs continuously to receive any audio data written to the shared memory. While other thread runs continuously to receive any

data other than audio data written to the shared memory. Fourth thread has the responsibility to play the audio data.

Refer to the class diagram of the project in section 4.5.

5.4 Distribution of classes with respect to Modules:

Distribution of classes with respect to modules for this project is demonstrated below.

5.4.1 Data Parsing Module:

This module consists of following class.

- CEB200

This class implements a member named **EB200UdpThread ()** which implements UDP server and is continuously listening for the receiver packets.

After receiving the packet, the **CheckDGram ()** function is called which checks whether this packet is EB200 packet or some other packet. If it is EB200 packet then it means packet is sent from receiver. If packet is from receiver then **ParseData ()** function is called. This function is to parse data which means to separate out packets which belong to different data.

After separating different packets, corresponding data functions are called e.g. **FScan()** is called for packets that are FScan packets and **MScan()** is called for MScan packets. When the corresponding data packets are received by corresponding functions they further dissect the packets getting different values which are sent by receiver.

Another functionality of these corresponding functions i.e. **FScan()** and **MScan()** etc is to populate shared memory with these new values which are received after

dissection of packets. After all this the control is returned to the **EB200UdpThread()** which waits for the next packet to come.

Following figure will clear the concept:

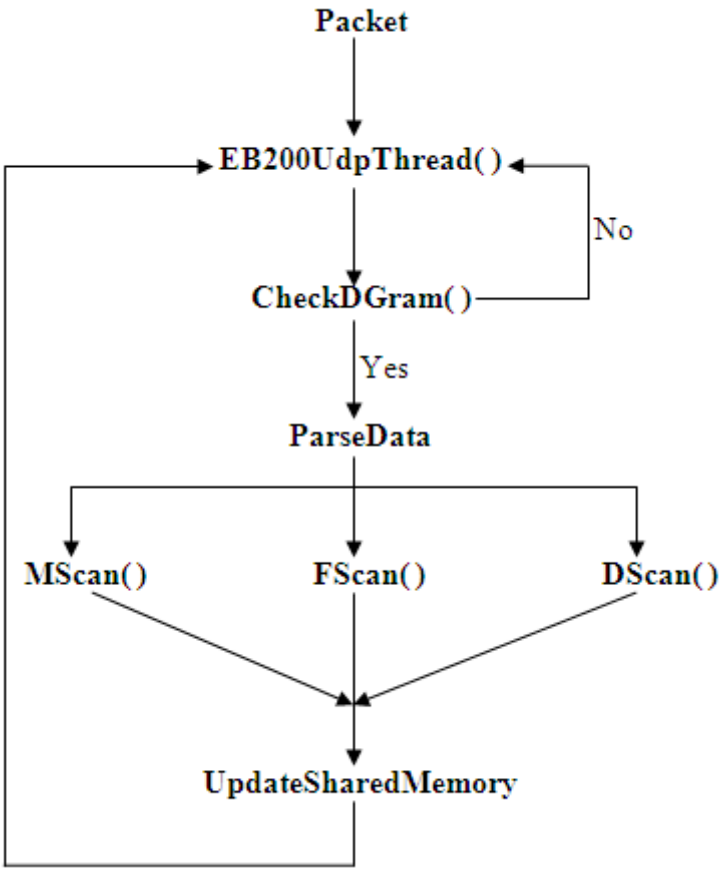


Figure 5-2: Data Parsing

5.4.2 Control Module:

Control Module consists of following classes.

a) MScan

MScan is for scanning memory from one memory location to the other memory locations which are set on the receiver.

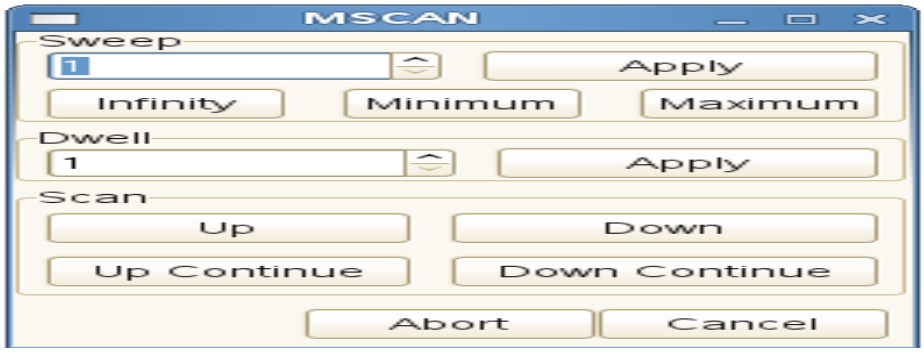


Figure 5-3: GUI-MScan

b) DScan

DScan is for digital scan, it performs the digital scan from a specified to a specified frequency.

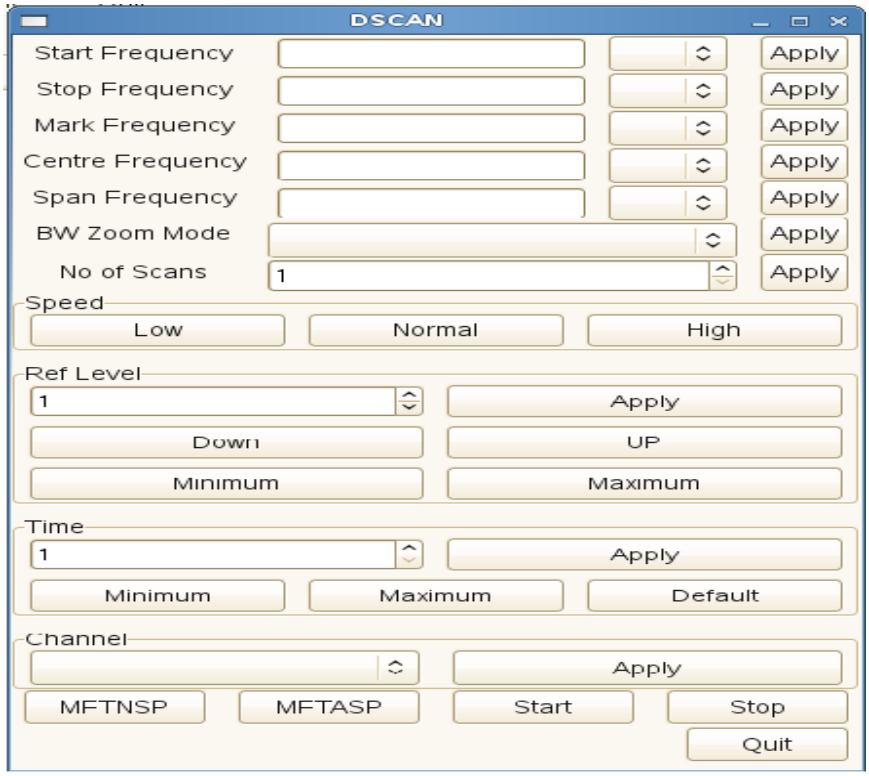


Figure 5-4: GUI-DScan

c) Att

Attenuation can be set, auto mode of attenuation can be set and also the state of the attenuation can be set.

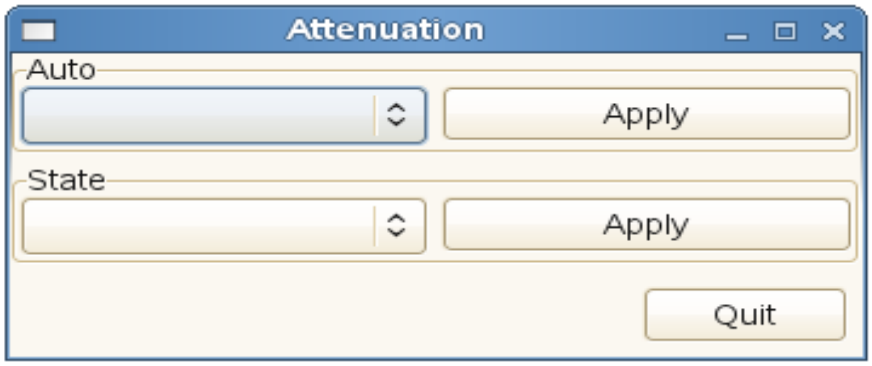


Figure 5-5: GUI-Att

d) FScan

Frequency scan be started and stopped. Start, stop and step frequencies are mentioned and frequency scan is run. Frequency scan can be run to the up direction or down direction.

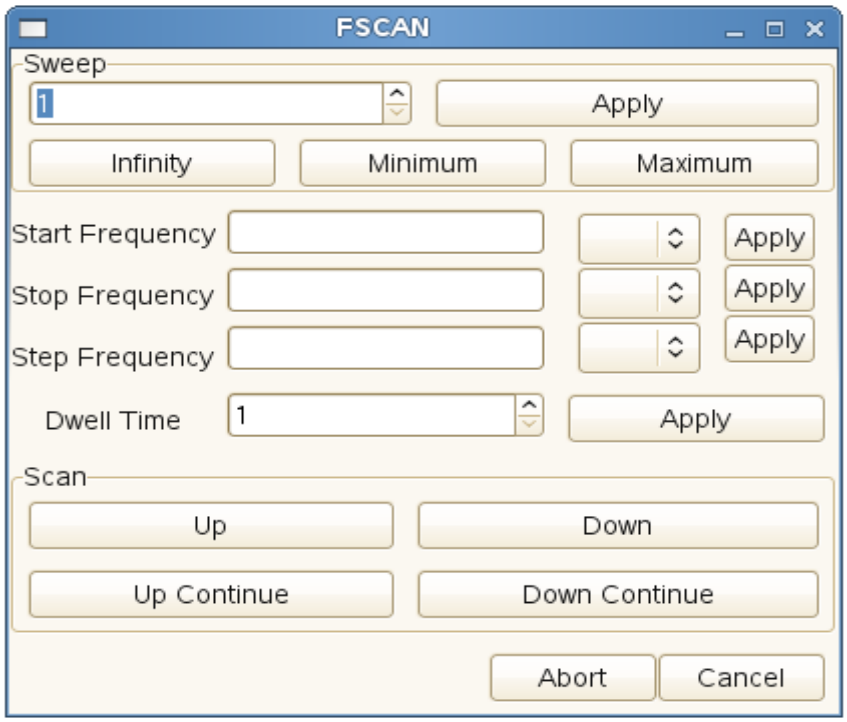


Figure 5-6: GUI-FScan

e) EditList

EditList can be used to view the memory contents. Basically receiver provides 1000 memory locations to be saved in it. Using EditList memory locations can be added or removed.

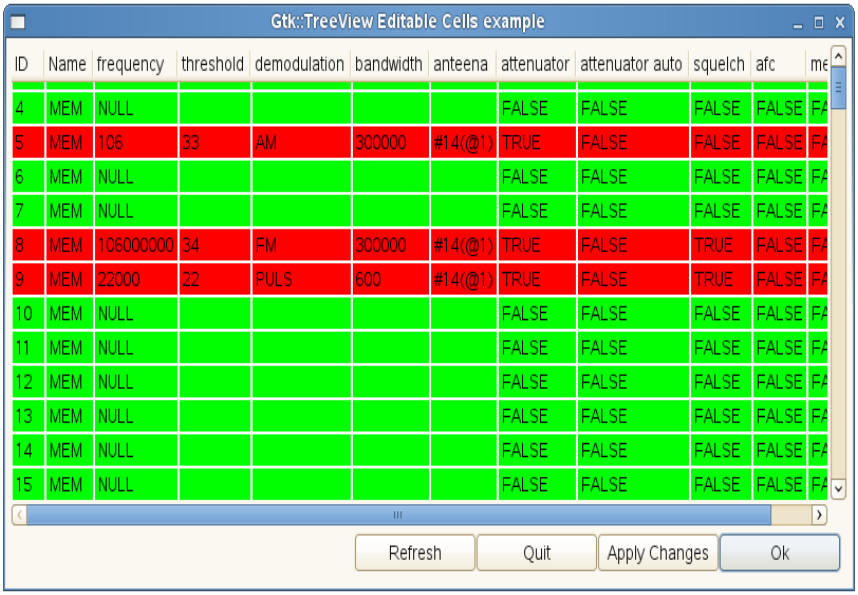


Figure 5-7: GUI-EditList

f) Squelch

Squelch can be set by setting different attributes in this window. State of the squelch can be set, similarly threshold can be set to a value which is between minimum and maximum values and also this threshold can be set to minimum and maximum level.

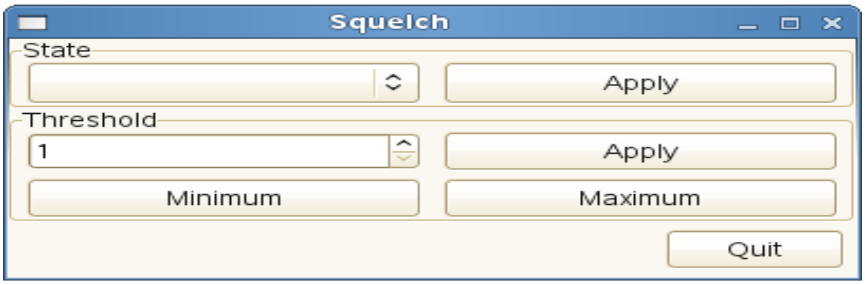


Figure 5-8: GUI-Squelch

g) MemContents

Memory contents can be set and updated using this window. Select a memory location from the spin box which is desired to be set or removed. Then in case of addition enter the relevant information in the remaining fields and press OK. In this way a new memory will be added or updated. For the purpose of deletion, provide only the memory number and press delete.

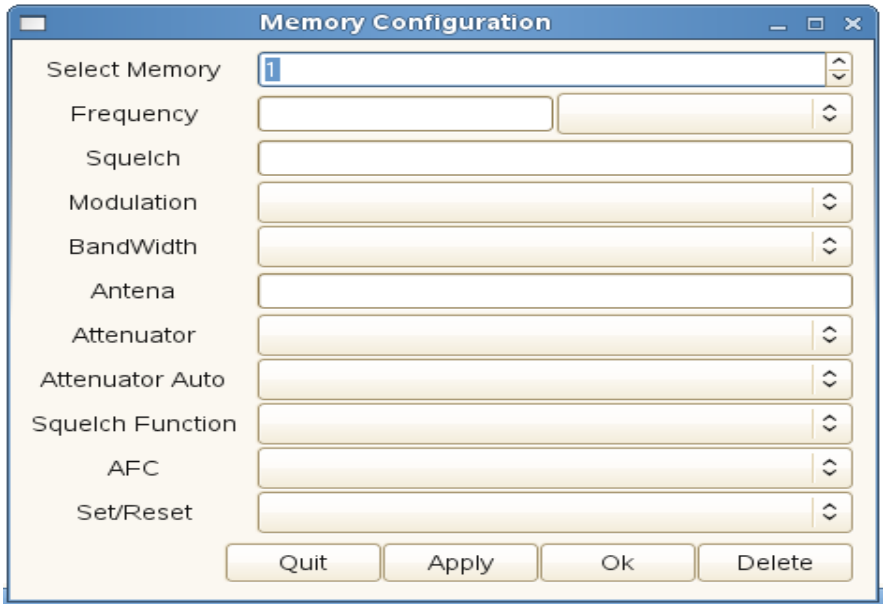


Figure 5-9: GUI-MemContents

h) Tone

Tone can be set to a desired level by selecting any value of the tone from the spin box. Tone can be set to minimum or maximum level. Similarly tone can be set to On or Off.

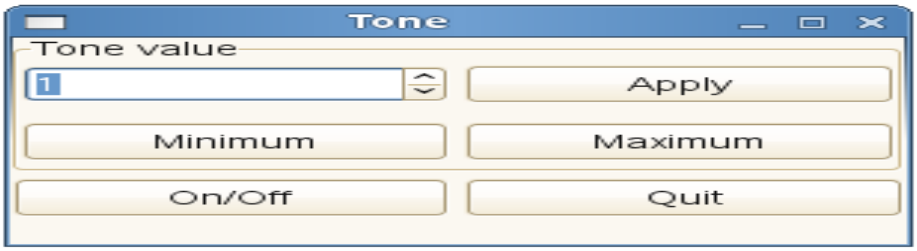


Figure 5-10: GUI-Tone

i) Gain

Gain can be set by using this window. Gain control mode can be selected; gain control value can be selected. Similarly Gain can be set to Minimum or Maximum value.

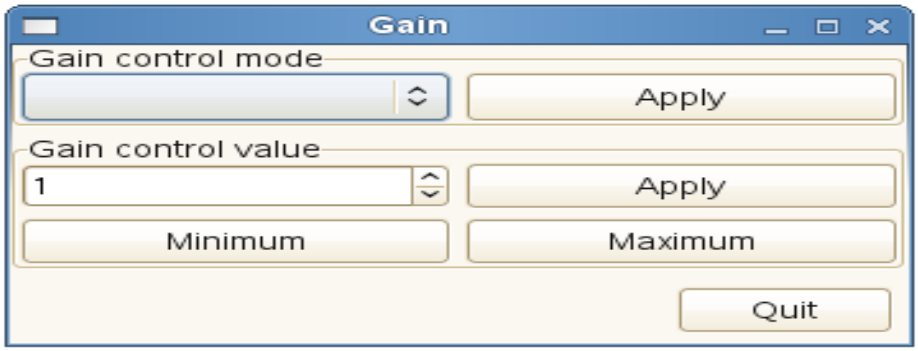


Figure 5-11: GUI-Gain

j) Common

Common controls of the receiver can be handled.

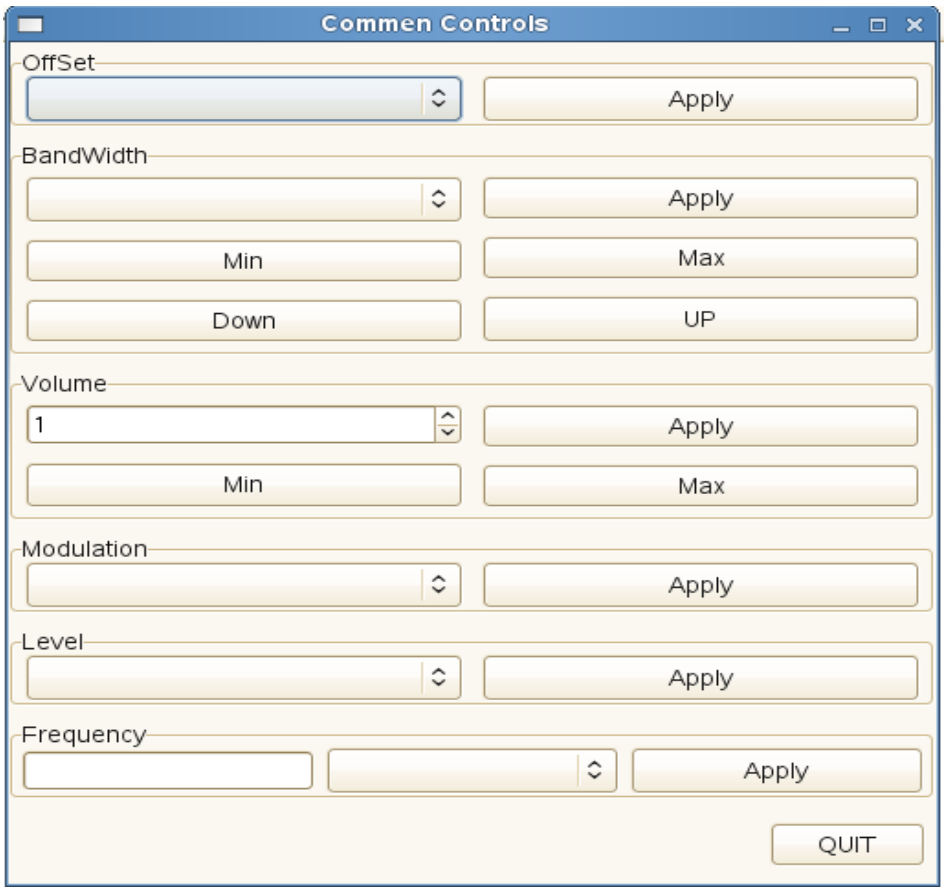


Figure 5-12: GUI-Common Controls

All classes implementing control module implement different GUI windows for the user. All windows are invoked by separate control module panel buttons on main window.

All the windows have the same functionality for implementing different functions of the receiver. All classes have class **ExampleWindow** as their friend class. When a user invokes a window and set different parameters in the widgets, they all are set in private variables of the classes. When the user presses **apply** button they all will invoke C**ommands(int n)** of the **ExampleWindow** class. They all give their separate identity by setting variable n. In **Commands(int n)** a switch statement is implemented. Since **ExampleWindow** is maintaining a TCP connection with receiver therefore separate commands are sent for every different n to the receiver; which execute that command.

Diagrammatically we can see the procedure as follows:

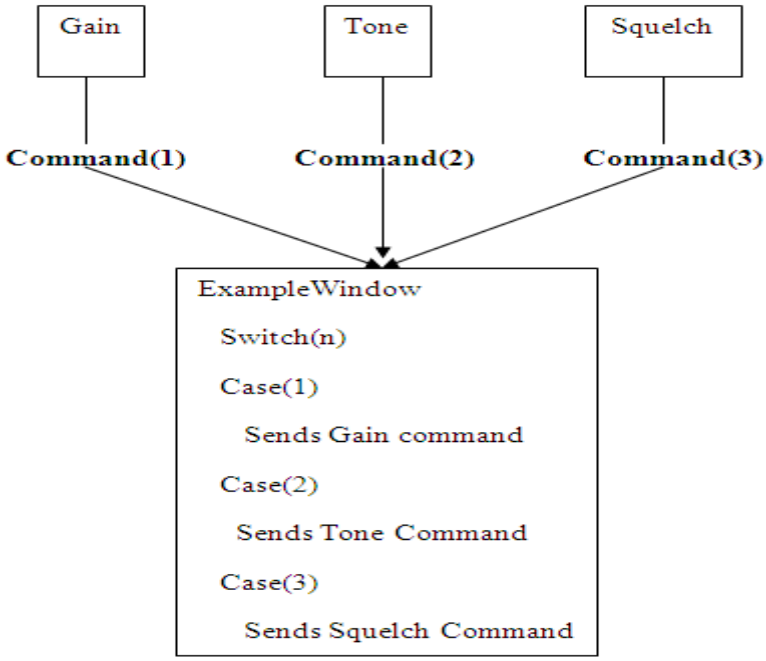


Figure 5-13: Control Module

5.4.3 Sound Module:

This module contains the following classes:

- Sound
- Efile

Main class for sound module is **Sound.** This class also implements a GUI window having four buttons

- Play
- Stop

On pressing **Play** button; a thread is initiated. This thread reads the "unsigned char data[800] " of the sound module and implements the playing of audio data.

5.4.4 Data Storage Module:

This module contains the following classes:

- MessageList
- MessageList2
- Thread
- Threadaudio

When the program starts; two threads start immediately along with the MainThread. These two threads are implemented as following classes:

- Thread
- Threadaudio

After the program startup, these two threads immediately start looking any updated data in shared memory. When the **Start** button is pressed in the main window, two things happen.

1. Second process is started
2. Open( ) function of MessageList and MessageList2 is called.

Open function does two things.

1. Opens the Data Base
2. Stores the data supplied by **Thread** and **Threadaudio** into the Data Base.

Once the **Stop** button on main GUI is pressed two things do occur

1. Process 2 (Parsing Process) closes
2. Data Base is closed

The whole process can be shown by following diagram:

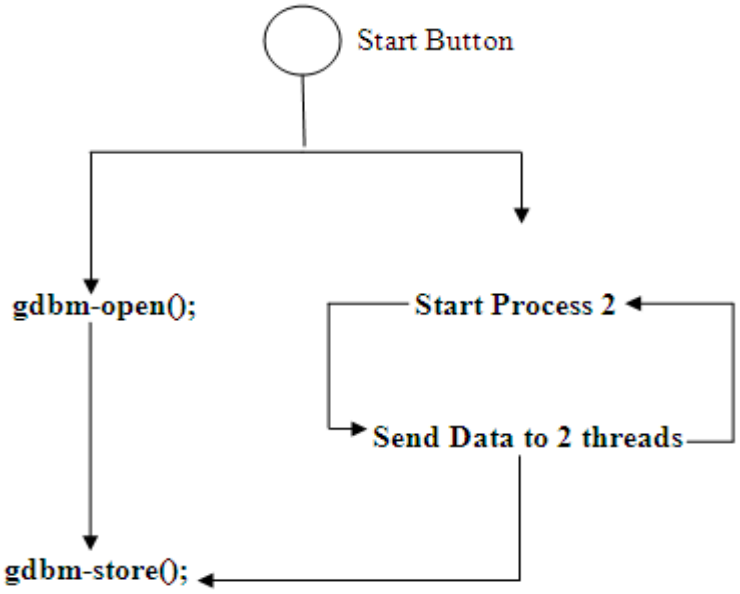


Figure 5-14: Data Storage Module

gdbm-store() is called every time new data is received by the threads.

5.4.5 Data Viewing:

Two types of data i.e. audio data and data other than audio are automatically stored in two different data bases with extension **.audio** and **.other**. When we want to open stored data we select the data base through file selection dialog. MessageList and MessageList2 function **Openfile(Glib::ustring)** is called supplying it with data

base name. In O**penfile()** first of all **gbdm-open( )** called, i.e. opening data base in read only mode.

In second step **gdbm-fetch( )** is called which fetches the data from the data base and updates the view variables. In the last step **gdbm-close()** is called which closes the data base.

5.5 Commands:

Following commands have been used in the implementation of the control module:

1. `"SYSTem:AUDio:VOLume MAX\n"`
2. `"FREQuency:AFC %s\n"`
3. `"BANDwidth %s\n"`
4. `"FREQuency %s%s\n"`
5. `"SYSTem:AUDio:VOLume 0.%d\n"`
6. `"DEModulation %s\n"`
7. `"DETector %s\n"`
8. `"BANDwidth MIN\n"`
9. `"BANDwidth MAX\n"`
10. `"BANDwidth UP\n"`
11. `"BANDwidth DOWN\n"`
12. `"SYSTem:AUDio:VOLume MIN\n"`
13. `"SYSTem:AUDio:VOLume MAX\n"`
14. `"ABORt\n"`
15. `"FREQuency:MODE SWEep\n"`
16. `"SWEep DIRection UP\n"`
17. `"INITiate\n"`
18. `"SWEep DIRection DOWN\n"`
19. `"INITiate:CONM\n"`

20. "SWEep:DWELl %d\n"
21. "SWEep:COUNt MINimum\n"
22. "SWEep:COUNt MAXimum\n"
23. "SWEep:COUNt INFinity\n"
24. "FREQuency:STARt %s%s\n"
25. "FREQuency:STOP %s%s\n"
26. "SWEep:STEP %s%s\n"
27. "FREQuency:MODE MSCan\n"
28. "MSCan DIRection UP\n"
29. "MSCan DIRection DOWN\n"
30. "MSCan:COUNt MINimum\n"
31. "MSCan:COUNt MAXimum\n"
32. "MSCan:COUNt INFinity\n"
33. "MSCan:DWELl %d\n"
34. "MSCan:COUNt %d\n"
35. "FREQuency:MODE DSCan\n"
36. "CALCulate:DSCan:MARKer:MAXimum:NEXT\n"
37. "CALCulate:DSCan:MARKer:MAXimum\n"
38. "FREQuency:DSCan:SPEed LOW\n"
39. "FREQuency:DSCan:SPEed NORMal\n"
40. "FREQuency:DSCan:SPEed HIGH\n"
41. "VOLTage:AC:RANGe UP\n"
42. "VOLTage:AC:RANGe DOWN\n"
43. "VOLTage:AC:RANGe MIN\n"
44. "VOLTage:AC:RANGe MAX\n"
45. "MEASure:TIME MINimum\n"
46. "MEASure:TIME MAXimum\n"

47. "MEASure:TIME DEFault\n"
48. "VOLTage:AC:RANGe %d\n"
49. "DSCan:COUNt %d\n"
50. "MEASure:TIME %d\n"
51. "FREQuency:DSCan:FCHannel %s\n"
52. "FREQuency:DSCan:STARt %s%s\n"
53. "FREQuency:DSCan:STOP %s%s\n"
54. "FREQuency:DSCan:MARKer %s%s\n"
55. "FREQuency:DSCan:CENTer %s%s\n"
56. "FREQuency:DSCan:SPAN %s%s\n"
57. "FREQuency:DSCan:RESolution:AUTO %s\n"
58. "INPut:ATTenuation:AUTO %s\n"
59. INPut:ATTenuation:STATe %s\n"
60. "OUTPut:SQUelch %s\n"
61. "OUTPut:SQUelch:THReshold MIN\n"
62. "OUTPut:SQUelch:THReshold MAX\n"
63. "OUTPut:SQUelch:THReshold %d\n"
64. "GCONtrol MAX\n"
65. "GCONtrol:MODE %s\n"
66. "GCONtrol %d\n"
67. "GCONtrol MIN\n"
68. "OUTPut:TONE:THReshold MIN\n" [2]

5.6 Conclusion:

This chapter incorporated the details of the classes implemented. . The classes have been distributed among the two basic processes of the system which are Main thread and Parsing thread. Parsing thread communicates with the receiver and receives the data sent from receiver. Data received by parser is put in a shared memory, from where it is being read by main thread. This process reads the data from shared memory and uses this data for viewing and storage purposes.

Chapter 6

6 Testing

6.1 Introduction:

Testing is a very important phase in the software development process. Once the coding process is completed, then the software goes under the testing process which involves checking the code for errors and bugs. It involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. This chapter involves all the testing techniques which have been employed in the project and the conclusions which have been deduced on the basis of the results of the testing procedures. Test cases for different units and components have been drafted illustrating their expected behaviors on the success and failure of each test. The output of each test is then compared with the one documented in the test case to make sure that the system behaves in the same way in which it is meant to behave.

6.2 Testing Process:

The testing process has been carried out throughout the development process as an iterative approach has been used in the project for development. Each phase of development was visited several times making sure that the testing process goes in parallel with the development process. The testing was basically done at three levels, Unit testing, Integration testing and System testing.

6.2.1 Unit Testing:

Unit testing has been done to determine that whether the individual units of the source program work in the same way in which they are expected to work. The units

in the project include those methods which cannot be tested by simple inspection and those classes which cannot be broken down into smaller units for testing. The identified units of the project along with the corresponding test cases are illustrated under the following headings.

6.2.1.1 CEB200UdpSock.cc:

This is the class which lies on operator's machine and is used to perform the parsing operation. The expected results on success and failure can be observed from following table. On success the parsing module starts receiving UDP packets and starts parsing them. On failure this class will not receive UDP datagram.

Table 6-1: Test case for CEB200UdpSock.cc

Identity		CEB200UdpSock.cc
Category		Unit testing
Description		This class is used to initiated parsing module
Set up		Gtkmm is needed as supporting environment and this class is dependent on ExampleWindow.cc
Expected Results	Success	Parser starts and starts setting parsed data in shared memory
	Failure	This class will not receive UDP datagram

6.2.1.2 ExampleWindow.cc:

This is the central class of the application. It is used to start central and main thread of process1. Furthermore two more threads are also initiated when the constructor of this class is called. It shows main GUI for the application.

Table 6-2: Test case for ExampleWindow.cc

Identity		ExampleWindow.cc
Category		Unit testing
Description		This class is used to show the main graphical user interface of the application.
Set up		Gtkmm is needed as supporting environment
Expected Results	Success	Main GUI is displayed.
	Failure	The main GUI is unable to start.

6.2.1.3 Command () of ExampleWindow.cc:

This function takes as an argument an integer value and uses a switch statement to send a command.

Table 6-3: Test case for Command ( ) of ExampleWindow.cc

Identity		Command() of ExampleWindow.cc
Category		Unit testing
Description		This function takes as an argument an integer value and uses a switch statement to send a command
Expected Results	Success	On success FScan window will be launched
	Failure	On failure FScan will not be launched

6.2.1.4 MessageList2.cc:

This class is used for two main functions. First is to view the audio data and second is store the audio data in gdbm database. Openfilwe() is used to read data from gdbm database and then display it. Void getters() is used to store data in gdbm.

Table 6-4: Test case for MessageList2.cc

Identity		MessageList2.cc
Category		Unit testing
Description		This class is used to view and store the data
Set up		Gtkmm is needed as supporting environment and this class is dependent on ExampleWindow.cc
Expected Results	Success	When openfile function is called, the system reads data from gdbm and displays it in the main window.
		When getters() is called, the data will be stored in gdbm.
	Failure	In case of failure system will not read data from gdbm and will not display this data in the main window.
		Data will not be stored in gdbm.

6.2.1.5 MessageList.cc:

This class is used for two main functions. First is to view the other data and second is store that data in gdbm database. Openfilwe() is used to read data from gdbm database and then display it. Void getters() is used to store data in gdbm.

Table 6-5: Test case for MessageList.cc

Identity		MessageList.cc
Category		Unit testing
Description		This class is used to view and store the data
Set up		Gtkmm is needed as supporting environment and this class is dependent on ExampleWindow.cc
Expected Results	Success	When openfile function is called, the system reads data from gdbm and displays it in the main window.
		When getters is called, the data will be stored in gdbm.
	Failure	In case of failure system will not read data from gdbm and will not display this data in the main window.
		Data will not be stored in gdbm.

6.2.1.6 Thread.cc:

This class starts defines the thread which is used to continuously pick the data up from shared memory and update the variable of MessageList class. This class starts when constructor of main class is called and runs continuously till the application is alive.

Table 6-6: Test case for Thread.cc

Identity		Thread.cc
Category		Unit testing
Description		This class is used to get data other than audio from shared memory
Set up		Gtkmm is needed as supporting environment and this class is dependent on ExampleWindow.cc and CEB200UdpSock.cc
Expected Results	Success	Data will be gathered from shared memory
	Failure	In case of failure data gathering will be failed.

6.2.1.7 Threadaudio.cc:

This class starts defines the thread which is used to continuously pick the audio data up from shared memory and update the variable of MessageList class. This class starts when constructor of main class is called and runs continuously till the application is alive.

Table 6-7: Test case for Threadaudio.cc

Identity		Threadaudio.cc
Category		Unit testing
Description		This class is used to get audio data other than audio from shared memory
Set up		Gtkmm is needed as supporting environment and this class is dependent on ExampleWindow.cc and CEB200UdpSock.cc
Expected Results	Success	Audio Data will be gathered from shared memory
	Failure	In case of failure, audio data gathering will be failed.

6.2.1.8 MScan.cc:

This class is used to launch window of memory scan from the main window. The window of this class is launched when a button from main window is pressed. This class takes different parameters from user as inputs and calls command (**int n**) function of the ExampleWindow which then sends command to the receiver which is then executed.

Table 6-8: Test case for MScan.cc

Identity		MScan.cc
Category		Unit testing
Description		This class takes different parameters from user as inputs and calls command (**int n**) function of the ExampleWindow which then sends command to the receiver which is then executed.
Set up		Gtkmm is needed as supporting environment and this class is dependent on ExampleWindow.cc
Expected Results	Success	On success MScan window will be launched
	Failure	On failure MScan will not be launched

6.2.1.9 DScan.cc:

This class is used to launch window of digital scan from the main window. The window of this class is launched when a button from main window is pressed. This class takes different parameters from user as inputs and calls command (**int n**) function of the ExampleWindow which then sends command to the receiver which is then executed.

Table 6-9: Test case for DScan.cc

Identity		DScan.cc
Category		Unit testing
Description		This class takes different parameters from user as inputs and calls command (**int n**) function of the ExampleWindow which then sends command to the receiver which is then executed.
Set up		Gtkmm is needed as supporting environment and this class is dependent on ExampleWindow.cc
Expected Results	Success	On success DScan window will be launched
	Failure	On failure DScan will not be launched

6.2.1.10 FScan.cc:

This class is used to launch window of frequency scan from the main window. The window of this class is launched when a button from main window is pressed. This class takes different parameters from user as inputs and calls command (**int n**) function of the ExampleWindow which then sends command to the receiver which is then executed.

Table 6-10: Test case for FScan.cc

Identity		FScan.cc
Category		Unit testing
Description		This class takes different parameters from user as inputs and calls command (**int n**) function of the ExampleWindow which then sends command to the receiver which is then executed.
Set up		Gtkmm is needed as supporting environment and this class is dependent on ExampleWindow.cc
Expected Results	Success	On success FScan window will be launched
	Failure	On failure FScan will not be launched

6.2.2 Component Testing:

Different units together form a component. After unit testing, the components have been tested to make sure that they behave in the expected way. The test cases for different components of the system are elucidated and shown under the following headings.

6.2.2.1 Parser Module:

The parser module is basically used to receive and parse the data coming from the receiver.

1. Receive data from receiver
2. Check the datagram whether it belongs to receiver or not.
3. Parse the incoming data as per its type e.g. FScan, DScan and MScan etc
4. Dispatch the parsed data to its respective functions

This module runs as a child process of the main process (GUI) and it transfers data to main process by sharing the memory with it. The tests conducted on this

component were very successful as it behaved in the same way as expected according to the test case given in Table 6.11.

Table 6-11: Test case for Parser Module

Identity		Parser Module
Category		Component testing
Description		The parser module has the following functionalities 1. Receive data from receiver 2. Check the datagram whether it belongs to receiver or not. 3. Parse the incoming data as per its type e.g. FScan, DScan and MScan etc 4. Dispatch the parsed data to its respective functions
Set up		Gtkmm is needed as supporting environment
Expected Results	Success	This module is run separately. Receiver is turned to different settings e.g. it is set for digital scan. The corresponding values for DScan are observed by printing values in a console. If the printed values are same as the receiver settings then it is a success.
	Failure	If printed are not same a receiver settings then it is a failure.

6.2.2.2 Control Module:

Control module is used to control the receiver remotely. The tests that were performed on this component according to the test case in Table 6.12 were quite successful, validating the test case.

Table 6-12: Test case for Control Module

Identity		Control Module
Category		Component testing
Description		This module is used to control the receiver remotely
Set up		Gtkmm is needed as supporting environment
Expected Results	Success	When we open some control module window, sets the parameters and sends it to receiver, the receiver must execute that command which can be seen on front panel of receiver. If the parameters set on window are the same as that of front panel then test is successful.
	Failure	If parameters set are not same as on the front panel then that will a failure.

6.2.2.3 Storage Module:

This module is used to store the data received from the receiver in a database. The database used is gdbm. The test case for this component is illustrated in Table 6.13.

Table 6-13: Test case for Storage Module

Identity		Storage Module
Category		Component testing
Description		This module is used to store the data received from the receiver in a database. The database used is gdbm.
Set up		Gtkmm is needed as supporting environment
Expected Results	Success	When we press start button of the main window, the data starts pouring to the main window. When a name is given to the database, two types of databases are produced, one is with .audio extension and other is with .other extension. After data starts pouring in, the audio data is stored in database with .audio and other data is stored in database with .others extensions respectively. After stopping the data we will open these databases to see whether data was stored or not. If we view the data then the test is successful.
	Failure	If we do not see any data in the file, then it is a failure.

6.2.2.4 Sound Module:

This module is used to play the real time audio data coming from the receiver. The test case for this component is illustrated in Table 6.14.

Table 6-14: Test case for Sound Module

Identity		Sound Module
Category		Component testing
Description		This module is used to real time audio data coming from the receiver.
Set up		Gtkmm is needed as supporting environment
Expected Results	Success	If sound comes properly and continuously without noise then it is a success.
	Failure	If too much noise comes or sound does not appear to come then it is failure.

6.2.3 Integration Testing:

Integration testing means testing the functionality of the system stepwise while integrating the components or modules. While amalgamating the components, tests are carried out each time the components are integrated. If the tests are successful, then further integration of the system takes place. Otherwise the components are debugged and integrated again and again until the tests are successful.

In the project, the components were integrated in four main steps. First of all Data Parser and GUI were integrated and testing was done. If the test results were successful, then Control module was combined with these two and the system was tested again. After that, Sound Module was amalgamated with the rest of the system and tests were carried out again. Finally Data Storage Module was embedded with the rest of the system and tests were carried out. These steps have been elaborated as follows:

6.2.3.1 Integration of Data Parser Module with GUI:

Data parsing module comprising of different classes is responsible for receiving the datagram packets from the receiver. Then this module checks whether the coming datagram packet is of the receiver or not. If datagram is of receiver then it is accepted otherwise it is rejected. Initially the parsing module was integrated with GUI and their combined functionality was tested. After the integration of these two modules a connection between receiver and computer should be established through GUI, data should be checked and received in the GUI and data should be displayed in GUI. The results were correct, so further components were integrated with it. The test case which was considered to check the expected behavior of the integrated components is given below.

Table 6-15: Test case for Integrated Data Parser and GUI

Identity		Integrated Data Parser and GUI
Category		Integration testing
Description		As these modules are integrated, the combined effect should be that user should be able to connect the receiver to the computer, Start and end of the connection is also its functionality. GUI should only accept packets from the receiver, Data should be received by GUI and this data should be displayed by GUI.
Set up		Gtkmm should be there as a supporting environment, all classes which collectively form these module should be there.
Expected Results	Success	The combined effect should be that user should be able to connect the receiver to the computer, Start and end of the connection is also its functionality. GUI should only accept packets from the receiver, Data should be received by GUI and this data should be displayed by GUI. Fulfillment of all these functionalities will result in success.
	Failure	If any of the above functionalities is not performed then it is a failure.

6.2.3.2 Integration of Control Module, Data Parser and GUI:

This step is performed after successful integration of Data Parser and GUI modules. Control Module implements the command set of the receiver. By integrating the control module with the data parser and GUI module, user will be able to send a command to the receiver through computer. Criteria for success will be, that user should be able to send all possible commands available in GUI. If any of the command fails to executer which is present is GUI then this will be a failure. Test case was carried and it was successful. The following table shows the test case for this integration process.

Table 6-16: Test case for Integrated Control, Data parser and GUI.

Identity		Test case for Integrated Control, Data parser and GUI
Category		Integration testing
Description		As these classes are integrated, the collective functionality which is performed includes the performance of the control commands available in GUI successfully from the computer.
Set up		Gtkmm should be there as a supporting environment, all classes which collectively form these module should be there. Integration of data parser and GUI should be performed successfully before carrying this step.
Expected Results	Success	The collective functionality which is performed includes the performance of the control commands available in GUI successfully from the computer.
	Failure	If any command present in GUI fails to execute then this will be a failure.

6.2.3.3 Integration of Data Storage Module with Data Parser, Control Module and GUI:

This is the final step of the integration process. The collective functionality that should be performed is that user should be able to store the incoming data in a file. Based on the type of the data the system should save the data in files having .audio or .other extensions according to their data types. For that purpose a database is created and these files are kept in that database. The test case which was written to authenticate the combined functionality of the system is as follows:

Table 6-17: Test case for Integrated Data Storage Module, Data Parser Module, Control Module and GUI

Identity		Integrated Data Storage Module, Data Parser Module, Control Module and GUI
Category		Integration testing
Description		The combined functionality of this stage is that user should be able to store the coming data in a file with .audio or .other extension depending on the data type.
Set up		Gtkmm should be there as a supporting environment, all classes which collectively form these module should be there. Integration of data parser, control module and GUI should be performed successfully before carrying this step.
Expected Results	Success	If user is able to store the data according to correct data type then it is a success.
	Failure	If user is unable to store the data in the corresponding extension then it is a failure.

6.2.4 White Box Testing:

White box testing or structural testing uses an internal perspective of the system to design test cases based on internal structure. It requires programming skills to

identify all paths through the software [3]. The white box testing of the system has been done at both unit testing and component testing stages.

6.2.5 Black Box Testing:

Black Box Testing is testing without knowledge of the internal workings of the item being tested. It attempts to derive sets of inputs that will fully exercise all the functional requirements of a system [4]. For each set of inputs, outputs are known and in black box testing, the inputs are fed in and if the output matches the predicted output it means that the system delivers the expected functionality.

If we consider that data as valid data for controlling the receiver remotely, playing sound and storing data, following tests were conducted as part of the black box testing.

6.2.5.1 Checking the System on Valid Data:

First the system was checked on data for which system controls the receiver remotely, system plays the audio sound and system stores the incoming data in a database. The results were checked and examined carefully which proved the right functionality of the system on entering valid data.

6.2.5.2 Checking the System on Invalid Data:

Secondly the system was checked with invalid. So that this was the data where the system should return wrong outputs or system should performs any error. This step was carried out by placing invalid data values in control module and data storage module. As a result these commands were simply discarded and hence it does not show any wrong output.

6.2.5.2.1 Skipping the noncompulsory fields:

The control part takes the input from the user to execute certain commands. If in a control window user skips some fields then there is no error of failure in the application. These fields are checked and if without these fields command can be executed then it is executed and if command cannot be executed then such command will be discarded and no error is generated.

6.2.5.2.2 Skipping the compulsory fields:

Next while entering data, compulsory fields were skipped and it was observed that, the commands send to control the receiver remotely and store the data are simply discarded.

6.2.6 Static Analysis of Code:

Besides testing the code dynamically, static analysis of the code has been done as well to find defects, if any, in the blocks of code due to which it does not implement the exact requirement or to determine the ways by which the code can be optimized to make it fool proof.

The code has been statically analyzed in many ways which are briefly illustrated under following headings.

6.2.6.1 Control Flow Analysis

Control flow analysis has been carried out for the verification and validation of control blocks in the source code, for instance, the ‘for’, ‘while’ loops and the ‘if’ condition blocks. It has been observed that no unnecessary code has been included and all these blocks are optimized.

6.2.6.2 Data Analysis:

Data analysis has been done to find and remove improper initializations, unnecessary assignments and those variables that are declared but never used. All such unnecessary lines have been eliminated thus giving a refined code.

6.2.6.3 Interface Analysis:

Interface analysis has also been done to ensure consistency of interface, class, procedure declaration, definition and their use. It has been observed through tests that all the methods declared in the interface are correctly implemented in the classes and that there are no redundant methods.

6.3 Conclusion:

This chapter illustrated the testing process of the system that has been carried out and the corresponding results obtained. The testing of the system has been done in great detail. The test cases have been written for the three mains phases of testing, unit testing, component testing and integration testing. Using these test cases, the results of the tests have been authenticated. Both white box and black box testing have been carried out to determine that whether the system delivers all the functional requirements that it should be delivering. Even static inspection of the code has been carried out as well so that it become optimized and does not become redundant. All the test results were very successful proving that the system delivers all its functionalities in an efficient way.

Chapter 7

7 Future Work and Conclusion

7.1 Future Work:

The system that has been implemented in the project can be extended in many ways. First of all, the data stored by the system can be further used by some analysis software to analyze it. Future work can include the development of analysis software based on data stored by this application. Analysis may include the modulation, demodulation and bit stream analysis. Bit stream analysis is the technique is which analysis on each bit and stream of bits of data can be performed. This will be a challenging task.

Receiver does have the capability to monitor the frequency signals ranging from 9 KHz to 3GHz. So taking in mind this huge range, detection and monitoring of GSM transmission can be performed. GSM is protected and secured transmission, its transmission is encrypted and the process of eaves dropping is very difficult to be performed in such case. Real hard work, dedication and concentration will be required to achieve this task.

Audio tagging can be performed on the audio files being stored in this file. Audio tagging is a powerful technology to identify, search, and organize tons of music files. Unfortunately, many files have no information in the tags, or the information is not complete. This problem is annoying when you rip or download a huge number of music file, and then want to organize some collection or compilation.

After the process of tagging the audio files, these tags can be used for efficient retrieval of these audio files. This process will be very fast and effective.

7.2 Conclusion:

The software provides a platform for data analysis on the data stored by this software. Also this can serve as a baseline for the detection and monitoring of GSM transmission. Audio tagging can be performed on audio files and these tags can be to efficiently retrieve the audio files.

APPENDIX A

Hardware and Software Requirements

Hardware and Software Requirements

Hardware Requirements:

- 1.0 GHz Processor or More
- 256 MB of RAM or More
- 1 GB free space on Hard Disk

Software Requirements:

- **Platform:**

  Gtkmm 2.4

- **Operating System:**

  Linux OpenSuse 11.0

APPENDIX B

User Manual

User Manual

Following steps should be followed for the easy use of the software:

Main Menu:

1. Press Configure button to open a window which will ask you to enter IP address and port to connect with receiver. Fig2 will be displayed.
2. Click Connect button to start connection between receiver and computer.
3. Click Open button to go to Fig3 which will ask to enter a file name for data storage.
4. Click Start button to start the raw data storage in a file.
5. Press Stop button to stop the process of data storage in a file.

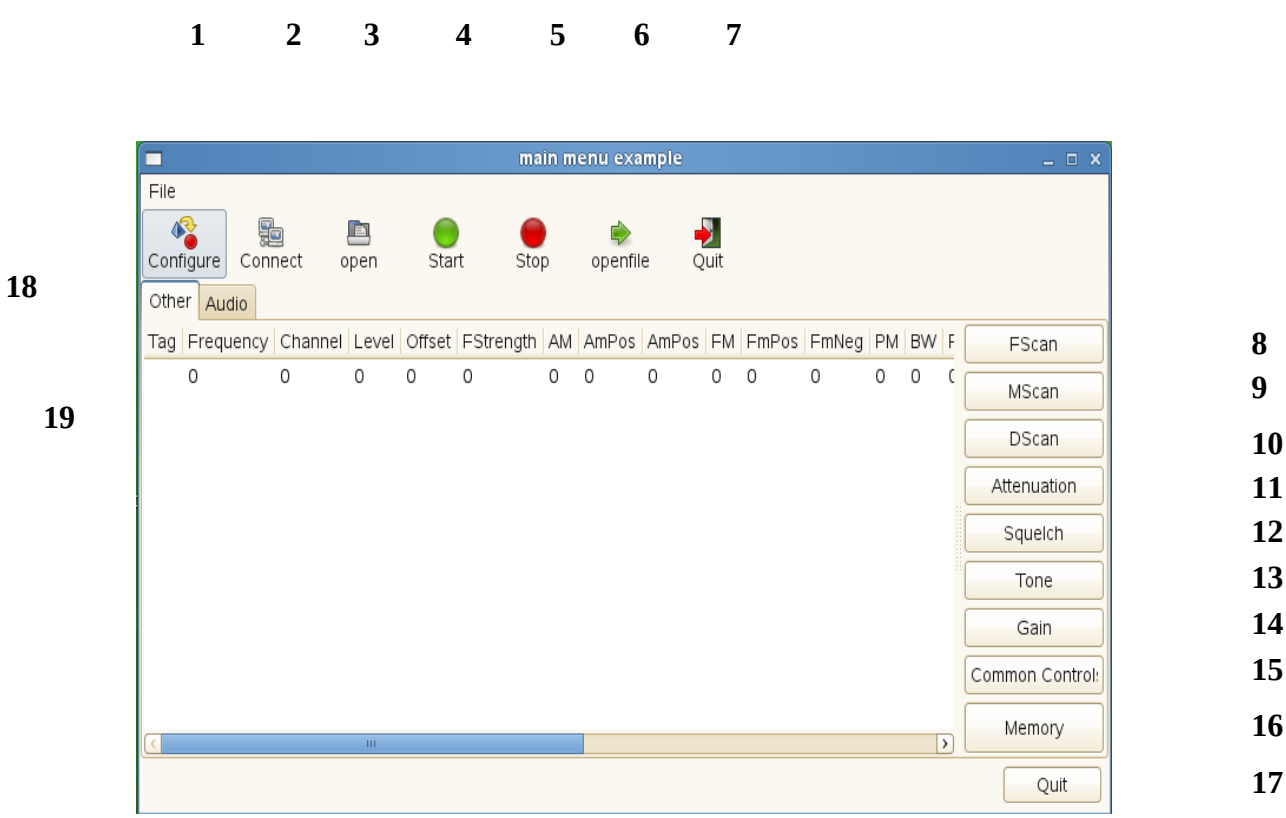


Fig 1: Main Menu

6. Press Openfile button to proceed to Fig4 which will ask user to select a file name to view data stored in that file in GUI.

7. Press Quit to exit from application.
8. Press FScan button to proceed to Fig5 which will ask the user to set different values to start the frequency scan.
9. Press MScan button to proceed to Fig6 which will the user to set different values to start the memory scan.
10. Press DScan button to proceed to Fig7 which will ask the user to set different values to start the digital scan.
11. Press Attenuation to proceed to Fig8 which will the user to set the Attenuation.
12. Press Squelch to proceed to Fig9 which will prompt the user to set Squelch.
13. Press Tone to proceed to Fig10 which will prompt the user to set the Tone.
14. Press Gain to proceed to Fig11 which will ask the use to set Gain.
15. Press Common Controls button to proceed to Fig12 which will ask user to enter certain values to control the receiver.
16. Press Memory button to go to Fig13 which will enable user to configure the memory locations of the receiver.
17. Press Quit button to exit the application.
18. Press the Other tab to display data with .other extension.
19. Press Audio tab to display the audio data.

Connection Settings:

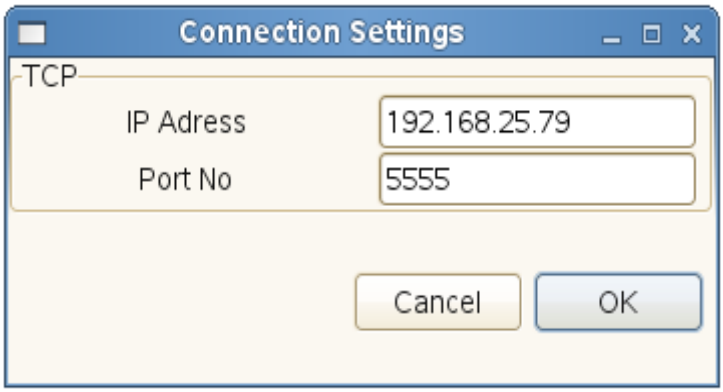


Fig 2: Connection Settings

1. Enter a valid IP address which set on receiver.
2. Enter a valid port number.
3. Press OK to establish connection.
4. Press cancel to cancel the settings and go to Fig1.

File Selection:

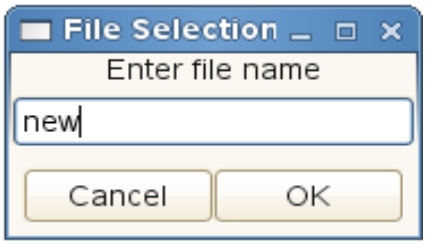


Fig 3: File Selection

1. Enter a valid name for the file
2. Press Ok button to proceed further
3. Press cancel button to abort the process and go to Fig1.

File Opening:

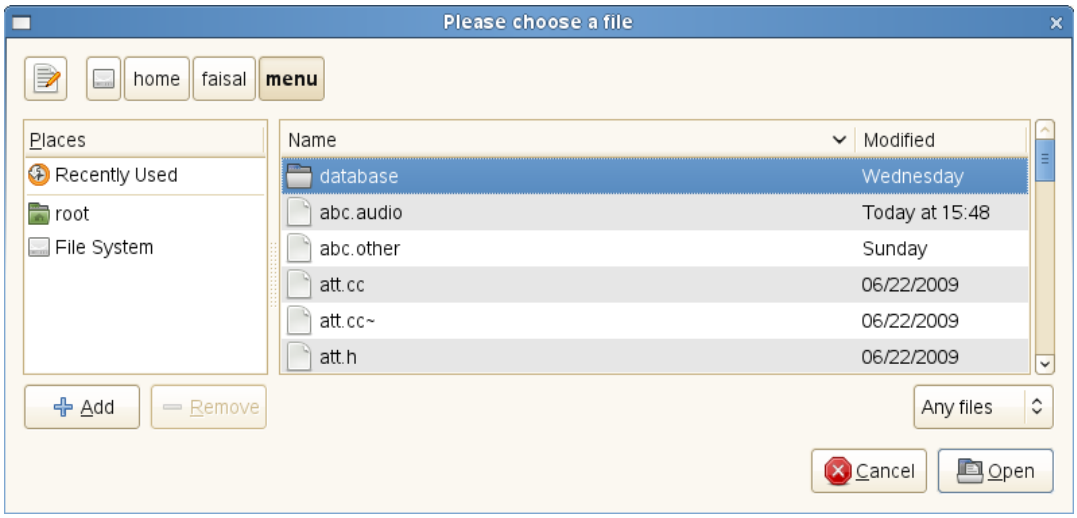


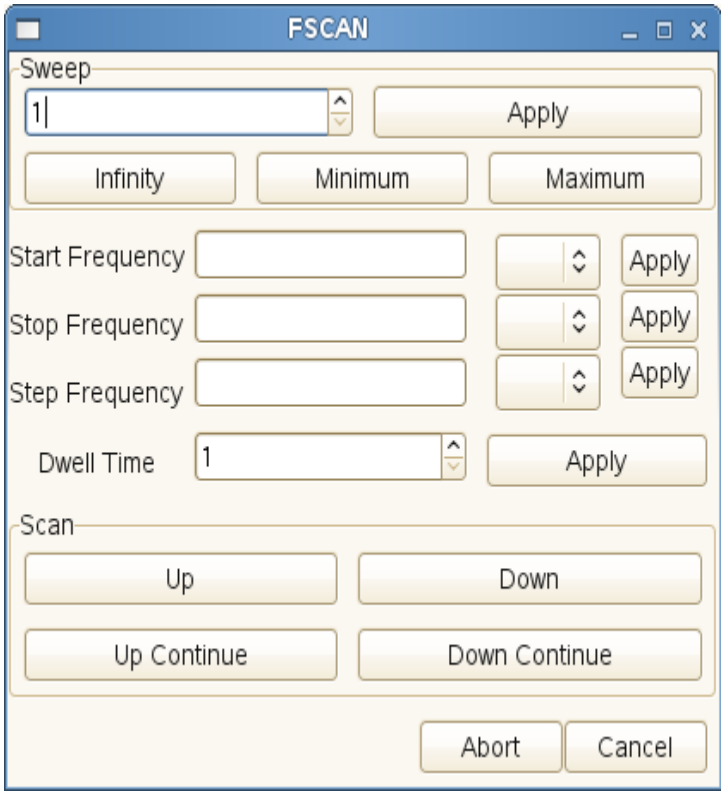
Fig 4: File Opening

1. Select the valid option from the tab.
2. Press open button to proceed further and display corresponding data in the main window (Fig1), data with .other extension will be displayed in other tab and data with .audio extension will be displayed in audio tab of the main window.
3. Press cancel button to go to Fig1.

FScan:

1. Enter a value for sweep from the spin box.
2. Press Apply button to apply changes.
3. Press infinity button to set the value to sweep to infinity.
4. Press maximum button to set the value of sweep to maximum.
5. Enter a value for the start frequency.
6. Press apply button to make the changes in start frequency.
7. Enter a value for the stop frequency.
8. Press apply button to make changes in stop frequency.

19



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 20

Fig 5: FScan

18

9. Enter a value for the step frequency.
10. Press apply button to make changes in step frequency.
11. Set a value for dwell time from the spin box.
12. Press apply button to save value of dwell time.
13. Press the Up button to start the scan to the upward direction.
14. Press the Down button to start the scan to the downward direction.
15. Press the Up Continue button to continue the scan to the upward direction.
16. Press the Down Continue button to continue the scan to the downward direction.
17. Press Cancel button to cancel the settings.
18. Press Abort button to abort and go to Fig1.
19. Click Minimum button to select minimum value for sweep.
20. Select value for frequency in the spin box.

MScan: **4**

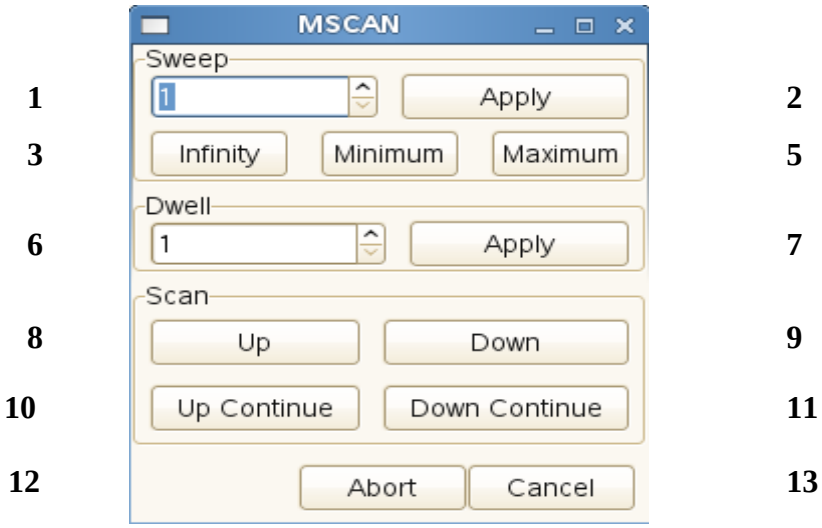


Fig 6: MScan

1. Enter a value for sweep from the spin box.
2. Press Apply button to apply changes.
3. Press infinity button to set the value to sweep to infinity.
4. Press minimum button to set the value of sweep to minimum.
5. Press the maximum button to set the value of sweep to maximum.
6. Set a value for dwell from the spin box.
7. Press apply button to save value of dwell time.
8. Press the up button to start the scan to the upward direction.
9. Press the down button to start the scan to the downward direction.
10. Press the up continue button to continue the scan to the upward direction.
11. Press the down continue button to continue the scan to the downward direction.
12. Press cancel button to cancel the settings.
13. Press abort button to abort and go to Fig1.

DScan:

1. Enter a value for the start frequency.
2. Select a valid range of frequency from the combo box. Possible values are KHz, MHz and GHz.
3. Press apply button to set changes in value of start frequency.
4. Enter a value for the stop frequency.
5. Press the apply button to set changes in value of stop frequency.
6. Set a value for mark frequency.
7. Press apply button to set value of mark frequency.
8. Set a value for centre frequency.
9. Press apply button to set value of Centre Frequency.
10. Enter a value for Span Frequency.
11. Press apply button to set changes in value of Span frequency.
12. Enter a value for BW (Band Width) Zoom Mode.
13. Press the apply button to set changes in value of BW Zoom Mode.
14. Set a value for Number of scans from the spin box.
15. Press apply button to set value of Number of Scans.
16. Press Low button to set the speed of scan to minimum value.
17. Press Normal button to set the speed of scan to moderate speed.
18. Click the High button to set the speed of scan to high.
19. Select value for the Ref Level from the spin box.
20. Press apply button to set value of the Ref Level.
21. Press the Down button to start the Ref Level backwards.

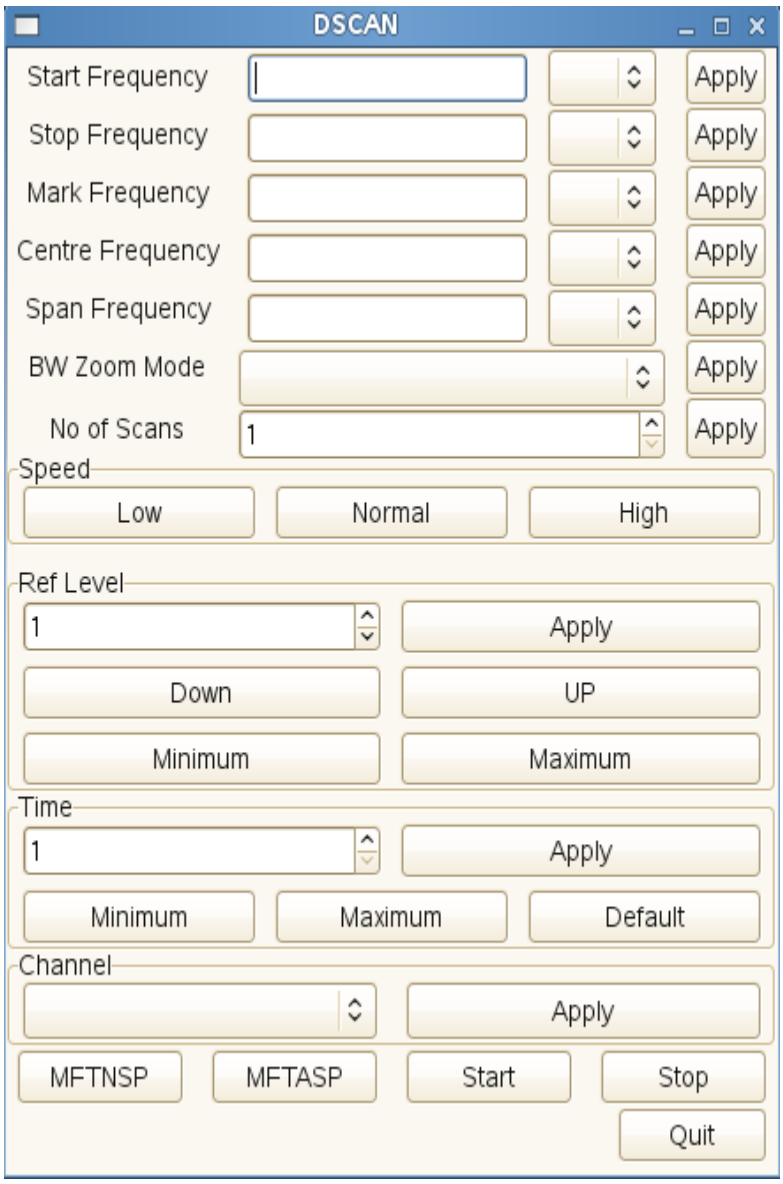


Fig 7: DScan

22. Press the Up button to start increasing the value of Ref Level upwards.

23. Press the Minimum button to set the Ref Level to minimum value.

24. Press the Maximum button to set the Ref Level to maximum value.

25. Select the Time for the Scan from the spin box.

26. Press the apply button to set the Time of scan.

27. Press the Minimum button to set the value of Time of scan to minimum.

28. Press the Maximum button to set the value of Time of scan to maximum.

29. Press the Default button to set the value of Time of scan to default, which was set as default from the manufacturer.
30. Select the Channel from combo box.
31. Press apply button to set the channel.
32. Press MFTNSP to set MFTNSP.
33. Press MFTASP to set MFTASP.
34. Press the Start button to start the DScan.
35. Press the Stop button to stop the running DScan.
36. Press Quit button to exit this window and return to Fig1 which is the main window.

Attenuation:

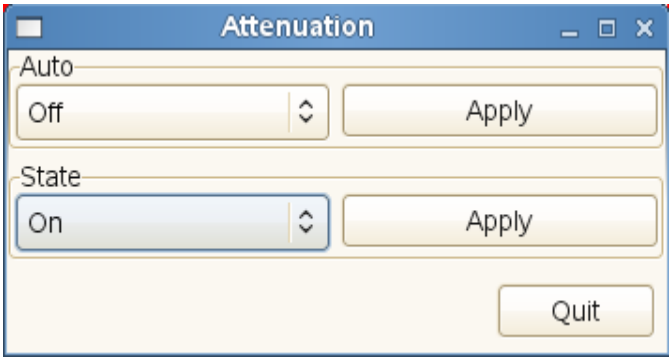


Fig 8: Attenuation

1. Select value for auto mode of Attenuation from the combo box, possible values are ON and OFF.
2. Press apply button to set the value of Auto mode.
3. Turn the value of State ON or OFF from the combo box.
4. Press the apply button to set the value of state.
5. Press Quit button to exit the window and return to main menu Fig1.

Squelch:

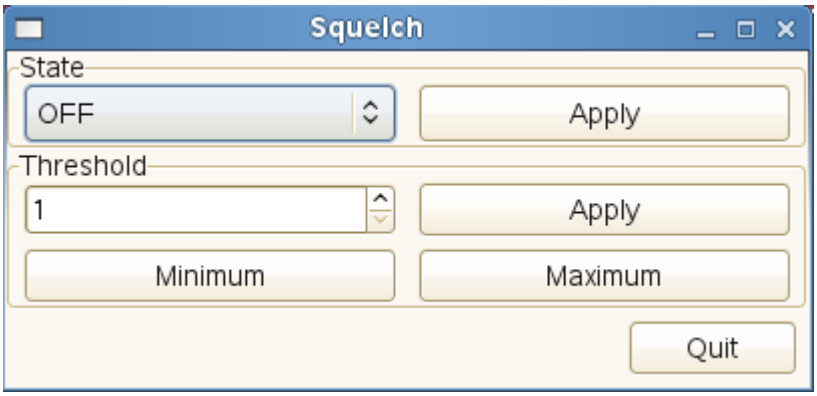


Fig 9: Squelch

1. Turn the value of State ON or OFF from the combo box.
2. Press apply button to set the value of State.
3. Select the value of Threshold from the spin box.
4. Press the apply button to set the value of Threshold.
5. Press Minimum button to set the value of Threshold to Minimum.
6. Press Maximum button to set the value of Threshold to Maximum.
7. Press Quit button to exit the window and return to main menu Fig1.

Tone:

1. Turn the value of Tone ON or OFF from the combo box.

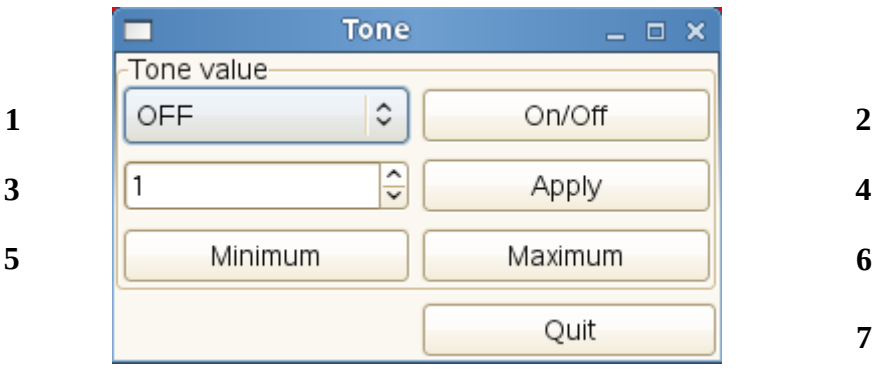


Fig 10: Tone

2. Press the On/Off button to make the Tone Value toggle.
3. Select Tone Value from the spin box.
4. Press apply button to set the value of Tone.
5. Click Minimum button to set the value of Tone to Minimum.
6. Click Maximum button to set the value of Tone to Maximum.
7. Click Quit button to exit this window and go to main window Fig1.

Gain:

1. Select the Gain Control Mode from the combo box.
2. Click the apply button to set the Gain Control Mode.
3. Select the Gain Control Value from the spin box.
4. Click apply button to set the Gain Control Value.
5. Click Minimum button to set the value of Gain Control to minimum.
6. Click Maximum button to set the value of Gain Control to maximum.
7. Click Quit button to exit this window and go to main window Fig1.

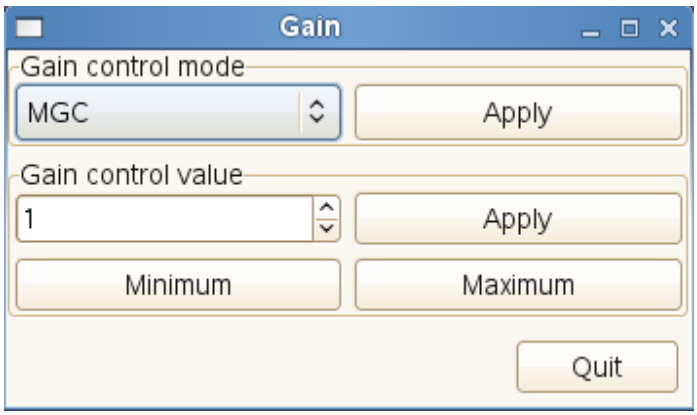


Fig 11: Gain

Common Controls:

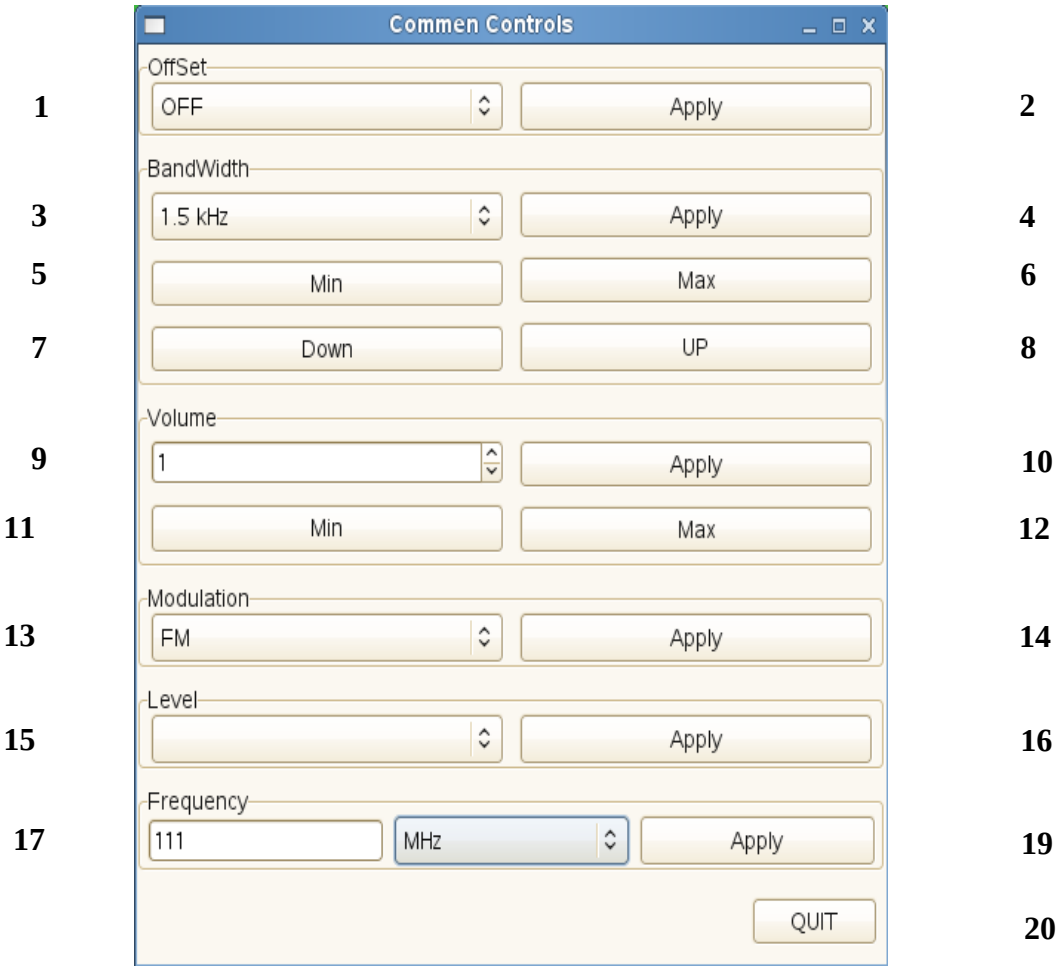


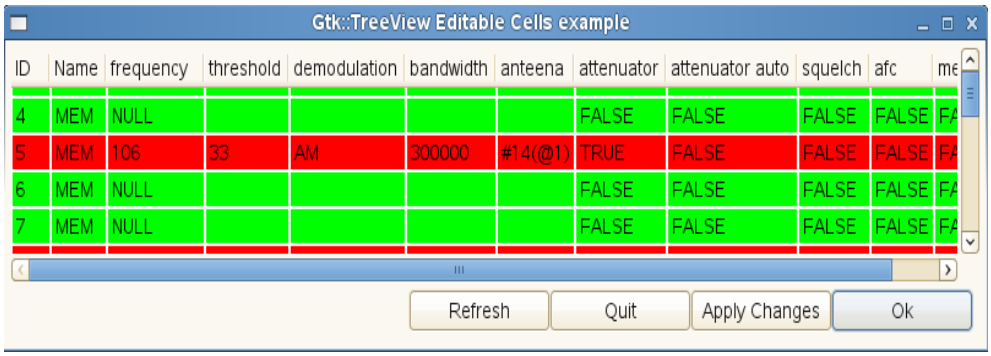
Fig 12: Common Controls

1. Select the Offset to On or OFF mode.
2. Click the apply button to set the Offset.
3. Select the Band Width from the combo box.
4. Click apply button to set the Band Width.
5. Click the Minimum button to send the band width to minimum.
6. Click the Maximum button to send the band width to maximum.
7. Click the Down button to set the band width level down by one step.
8. Click the Up button to set the band width level up by one step.
9. Select the value of Volume from the spin box.

10. Press apply button to set the value of volume to the receiver.
11. Press Minimum button to set the minimum value of the volume.
12. Press Maximum button to set the maximum value of the volume.
13. Select the value of Modulation from the combo box.
14. Press the apply button to set the type of Modulation.
15. Select the Level from the combo box.
16. Press the apply button to set the Level.
17. Click Quit button to exit this window and go to main window Fig1.

Memory View:

1. Press OK button display the memory contents.
2. Press apply button to go to Fig13 which will enable user to edit the memory locations.
3. Press Quit button to go to main menu Fig1.
4. Press Refresh button to display the changed memory contents.



4 3 2 1

Fig 13: Memory View

Memory Contents:

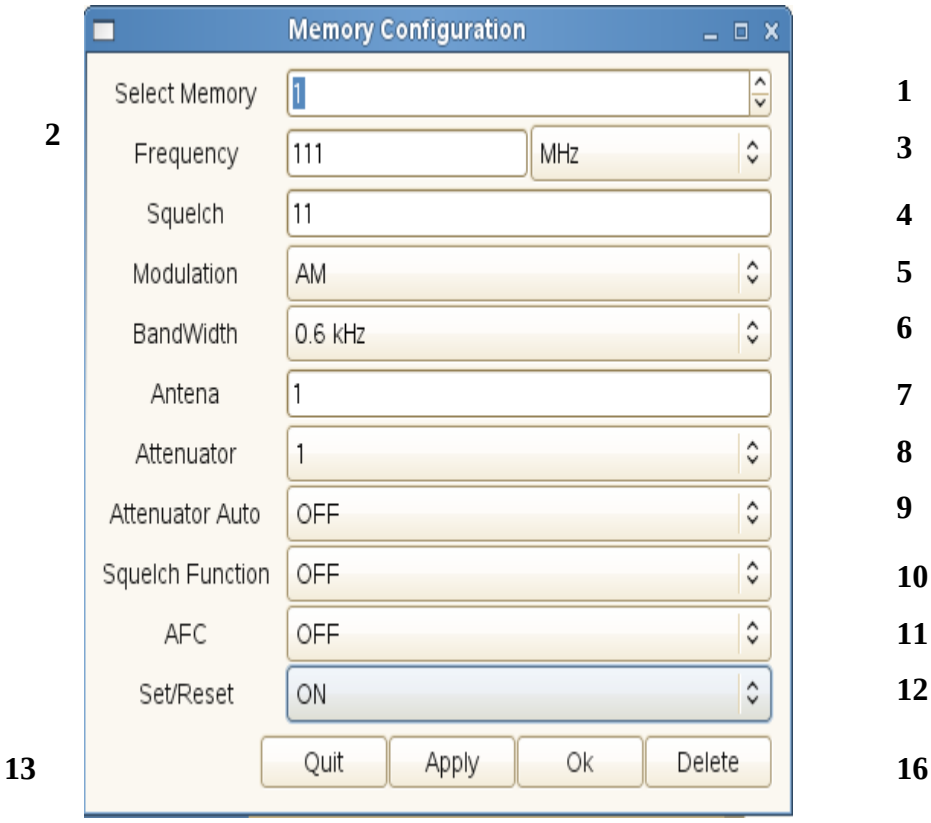


Fig 14: Memory Contents

1. Select Memory from the spin box.
2. Enter a value for the frequency in the text box.
3. Select the frequency from the combo box.
4. Enter a value for the Squelch in the text box.
5. Select Modulation from the combo box.
6. Select Band Width from the combo box.
7. Enter a value for Antenna number in the text box e.g. for antenna1 give (@1) in text box, and for antenna2 enter (@2).
8. Select a value of Attenuator from the spin box.
9. Set Attenuator Mode to ON or OFF from combo box.
10. Set Squelch Function from the combo box.

11. Set AFC from the combo box.
12. Enable or disable the Set/Reset option by selecting ON or OFF from the combo box.
13. Click the Quit button to go to Fig13.
14. Press apply button to apply changes but the window will not disappear.
15. Press OK button to first save the changes and then window will disappear and control will be shifted to Fig13.
16. Press Delete button to delete a memory location.

Sound:

1. Click on the Configure button to go to Fig16 which will ask user to provide IP address and port number to connect with receiver.
2. Click the Connect button to establish the connection.
3. Press Quit button to exit the application.
4. Press Play button to play the audio.
5. Press stop button to stop the audio.

2 **3**

1

4 **5**

Fig 15: Sound

Sound Connection Settings:

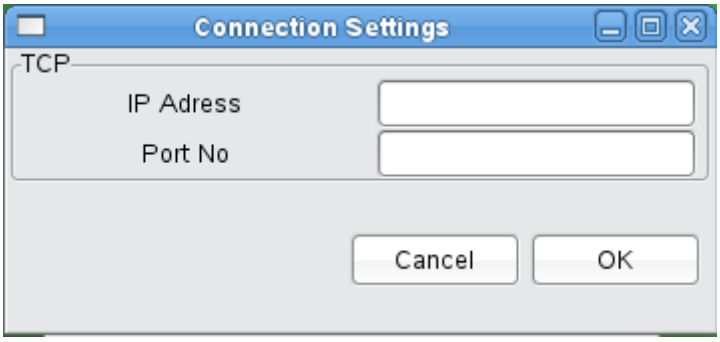


Fig 16: Sound Connection Settings

1. Enter IP address of the receiver.
2. Enter port number of the receiver.
3. Press OK to accept settings and return to Fig15.
4. Press Cancel to remove the settings and return to Fig15.

APPENDIX C

Deployment Manual

Deployment Manual

System will be deployed in following steps

1. Install the Gtkmm2.4 in the computer
2. Unzip the folder **Receiver.tar**
3. Create a folder named **receiver** in **/home** directory
4. Copy the unzipped files in **/home/receiver** directory
5. Open the terminal
6. Change your working directory to **/home/receiver**
7. Type in terminal the command:

   "**./myapp**"
8. Type in command "**./Sound**" to start the sound module.

APPENDIX D

Symbols and Abbreviations

Symbols and Abbreviations

R&S: Rohde and Schwarz

TCP: Transmission Control Protocol

IP: Internet Protocol

UDP: User Datagram Protocol

CMOS: Complementary Metal Oxide Semiconductor

RAM: Random Access Memory

PPP: Point to Point Protocol

LAN: Local Area Network

SCPI: Standard Commands for Programmable Instruments

IEEE: Institute of Electrical and Electronics Engineers

GUI: Graphical User Interface

FScan: Frequency Scan

MScan: Memory Scan

Dscan: Digital Scan

CW: Compact Wave

IFPAN: Intermediate Frequency Panorama

IPC: Inter Process Cmmunication

APPENDIX E

Bibliography

References

[1] Rohde & Schwarz “R&S ESMB user manual”
Published by Rohde & Schwarz

[2]Rohde & Schwarz “R&S ESMB Command Set”
Published by Rohde & Schwarz

[3] “White Box Testing,” https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/white-box/259-BSI.html

[4] Thomas Raishe, “Black Box Testing” Courses for CEN4010-SE 2002, Computer Science and Engineering Department, Florida Atlantic University. http://www.cse.fau.edu/~maria/COURSES/CEN4010-SE/C13/black.html