HOW TO DEVELOP AN AUTOMATED SYSTEM TO MONITOR THE RF CHARACTERISATION OF RADIOS

Nafiz Uddin Chowdury

Bachelor of Engineering
Electronics Engineering

Department of Electronic Engineering
Macquarie University

June 05, 2017

Academic Supervisor: Professor Stephen Hanly
Industry Supervisor: John Dalton

ACKNOWLEDGMENTS

I would like to acknowledge the help of my academic supervisor Professor Stephen Hanly and my industry supervisor John Dalton from RF Technology. This thesis work is aimed to create and implement an automated testing system at RF Technology. This concept will be explained throughout my paper.

I would also like to take this opportunity to show gratitude to a key engineer, Chris Mcknight, from RF Technology for the tremendous support he has given me starting from answering silly questions to giving me enough time beyond his busy schedule. I really appreciate for the time and attention he has given me throughout this project.

STATEMENT OF CANDIDATE

I, Nafiz Uddin Chowdury, declare that this report, submitted as part of the requirement for the award of Bachelor of Engineering in the Department of Electronic Engineering, Macquarie University, is entirely my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualification or assessment at any other academic institution.

Student's Name: Nafiz Uddin Chowdury

Student's Signature: Nafiz Uddin Chowdury

Date: 5th June, 2017

ABSTRACT

A radio frequency (RF) device is a small electronic device that is used to transmit and/or receive electronic signals such as a walkie talkie. It has a base-station that relays certain electronic signals (i.e. microwaves). It is mainly composed of two different components. One of them is a receiver end module and the other is a transmitter module. This paper consists of the work done with the receiver module and different tests that has been conducted from the instruction manual provided by RF Technology.

This paper discusses the steps that involve creating a process of automated tests which are required to configure the receiver module of the RF device. Currently, these tests are carried out manually and are very time-consuming and prone to human errors. Therefore, to save time and avoid human errors automation of these tests are highly essential. This in turn will give faster and more accurate results. The following section in this paper discusses the outcome of this project and highlights the problems and hurdles faced and the solutions devised.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last few decades the telecommunication industry has developed beyond imagination. Since the introduction of electricity, advancements have been made throughout, starting from telegraphs and now to cell-phones. Cell phones were first introduced to the public back in 1983 with no features at all. It took 10years to introduce text messaging while other features such as internet, camera were introduced much later.

The quality of calls made using a cell-phone or communications done through any sort of wireless medium (i.e. walkie-talkie) is highly important. Clear, uninterrupted and smooth voices are desired and the ability to make them as smooth and clear as possible meets the challenge.

This brings us down to the scope of this project. This project was entirely conducted at RF Technology whose main work is to design base-stations used in communication services. RF Technology [6] manufactures and markets radio communication all around the world. They currently have one branch in Thornleigh, NSW Australia and the other at Texas, USA. This company was established back in 1989 and since then has manufactured different types of base-station with different features. Currently RF Technology has two different types of base-station. They are known as Ultra-High Frequency unit (UHF) which has a range between 380 MHz and 560 MHz and Very High Frequency unit (VHF) that operates between 136 MHz and 174 MHz. These two units mainly cover all the channels that the Australian Government has allocated for public services. RF Technology sells radio base station units to NSW Police, Ambulance, Bush Fire and other state teams who then operate these stations under the allocated frequency.

Base station is a key component for wireless communication and its design and manufacture plays a vital role for having clear uninterrupted speech quality. Base stations are designed and manufactured [5]within RF Technology. Two of its important components are the two modules that are built into it namely receiver module and the transmitter module. The receiver modules main work is to pick up any signal that is transmitted by other base stations. The transmitter is used to transmit signal.

The radios that are used outdoors to communicate can face a lot of interruptions that can result in poor voice quality and miscommunication. The interruptions can occur for many reasons such as people using the same channel to communicate or stronger signal with high amplitude and modulation from a nearby channel can weaken the desired signal. It is inevitable that these base-stations pass rigorous testing in order to be certified and used in real-life scenarios. This thesis involves working on the tests that a receiver module of a base station needs to pass in order to be certified and used. The Telecommunication Industry Association (TIA) has set up certain standard [9] for these tests and RF Technology follows them. These tests ensure that a receiver module performs correctly in real-life situations.

RF Technology performs all the tests as set in the standards by TIA and ensures the two modules pass them before they are placed into the base-stations and shipped to different states within Australia and rest of the world. There are four key tests that are performed on the receiver module. These tests are done manually and can take long hours to complete. However they still can fail the standards due to human errors and poor instrument calibration.

The next few sections and chapters of this document discuss in depth the fundamental process of how we can send remote commands to instruments such as signal generators and automate these tests so that they are more reliable and less time consuming. More details, background theory and work are explained in the next few chapters regarding instrument automation.

1.1 Motivations and Challenges

Instrument automation can help a user to perform the tests on-site or even remotely. The aim of this project is to setup an automation process to make the tests faster, more reliable and ensure that it doesnt involve human errors. There are prior works and documentation regarding instrument automation however it doesnt involve automating the tests that are performed on the receiver module. This motivated me to work on this project as I believe this research document will help others in near future when further work needs to be carried out using instrument automation and base-stations.

However there were quite a few obstacles that were faced in order to complete the project. Most importantly the hardest part was to establish a remote connection between a base-station and a signal generator. Python programming was needed and Standard Commands for Programmable Instrument (SCPI) were also learnt. Capabilities of Virtual Instrument Software Architecture (VISA) [11]are recognised as it is needed to make sure SCPI commands perform correctly. Usages of Linux as an operating system were also required as the signal generators that were used for automation had Linux system kernel built-in already. In the next few chapters all these problems and their solutions are discussed in detail to give a clear concept on how this project was finished.

1.2 Contributions

In this section the project outcomes are outlined without much technical details. This section will also discuss the short and long term impact of the thesis undertaken.

1.2.1 Project outcomes

- A detailed literature review on background work is provided.
- Virtual Instrument Software Architecture, SCPI commands and its history and current standards are discussed.
- Detailed explanation on how to set up a remote connection between a signal generator, signal to noise and distortion (SINAD) meter and a base-station is also provided.
- Manual tests undertaken for future reference.
- SCPI commands and python scripts for instrument automation are explained.
- Finally automated tests and its results compared to the manual tests.

All the work done in this project contributes directly to the efforts for studying and exploring possible ways to improve the current status of the reliability of the tests performed on the receiver module. Furthermore, this paper can serve as a viable and reliable literature that involves instrument automation using python programming and SCPI.

1.3 Thesis overview

In this section a brief description of the whole project is outlined. The project plans and deliverables are outlined in the next section. This project has been supervised by Professor Stephen Hanly, in the Department of Engineering, Macquarie University and John Dalton from RF Technology. A regular meeting was set up every fortnight to check with the current progress of the thesis and discuss any problems encountered. As a result the project is completed in time and meets the schedule and deliverables set in the project plan. The consultation meeting attendance form is listed at the end of this document. The breakdown of this document is as follows:
Chapter 2 provides a thorough literature review and related background theory to this project. The topics discussed include history of instrument automation, VISA and the standard commands for programmable instruments (SCPI).
Chapter 3 discuss the manual tests undertaken and its results.

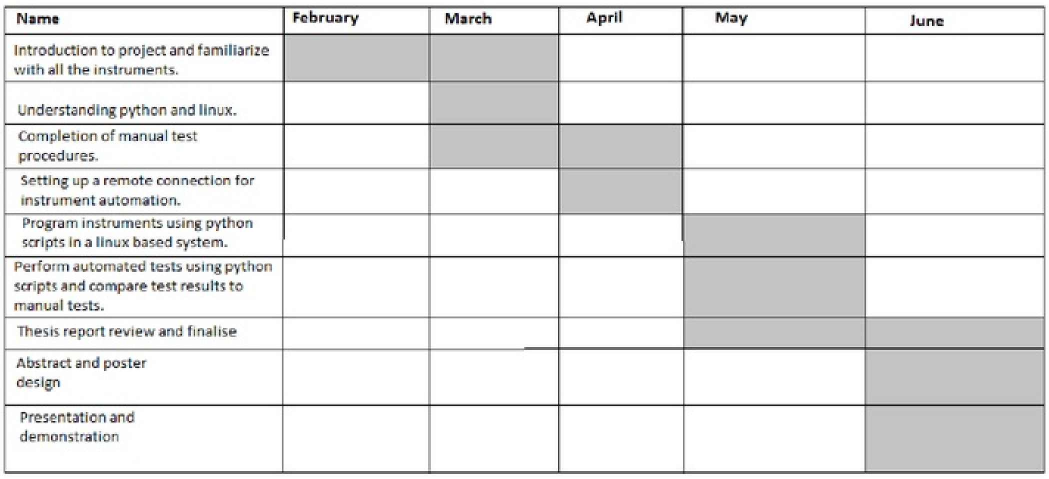
Chapter 4 explains how a remote connection is made between signal generators, SINAD meter and base station using VISA, SCPI and python scripts.
Chapter 5 describes the automated test procedures and compares the result to the manual tests.
Finally chapter 6 concludes and summarizes the current thesis work and provides a discussion for potential future research.

1.4 Project plan

This section provides the Gantt Chart showing the project deliverables.

Table 1.1: Breakdown of the entire project

Name	**Start Date**	**End Date**
Introduction to project and familiarize with all the instruments	27/02/2017	03/03/2017
Understanding python, SCPI codes and Linux.	06/03/2017	31/03/2017
Completion of manual test procedures.	27/03/2017	07/04/2017
Setting up a remote connection for instrument automation.	10/04/2017	28/04/2017
Program instruments using python scripts in a Linux based system.	01/05/2017	12/05/2017
Perform automated tests using python scripts and compare test results to manual tests.	15/05/2017	26/05/2017
Thesis report review and finalise	29/05/2017	02/06/2017
Abstract and poster design	01/06/2017	15/06/2017
Presentation and demonstration	01/06/2017	19/06/2017



Name	February	March	April	May	June
Introduction to project and familiarize with all the instruments.	▨	▨			
Understanding python and linux.		▨			
Completion of manual test procedures.		▨	▨		
Setting up a remote connection for instrument automation.			▨		
Program instruments using python scripts in a linux based system.				▨	
Perform automated tests using python scripts and compare test results to manual tests.				▨	
Thesis report review and finalise				▨	▨
Abstract and poster design					▨
Presentation and demonstration					▨

Figure 1.1: Gantt Chart showing the project deliverables

1.5 Project baseline review

This project was scheduled to be completed between 27th February 2017 and 19th June 2017. Baseline plan was made to ensure that all available days throughout the entire semester was utilised including weekends and mid-semester break.

1.5.1 Time budget and financial budget review

The table below shows the number of days spent in completion to the project. It also provides the percentage of project that has been completed so far. All the activity was monitored closely to ensure it met the allocated time. As a result no change was made to complete the project however some of the activity was not entirely completed in its own allocated time.

Table 1.2: Time budget review

Estimated work	112days
Completed work	97days
% of completion	87%

This project was initially allocated \$ 300 from Macquarie University, Department of Engineering. However this project was conducted entirely at RF Technology therefore no financial aid was required at all. The hardware that was used was free to access on site and software was available to download for free. As a result no purchasing was required at all.

Chapter 2

Background Theory

This chapter provides a detailed background concerning the instruments used in this project and how they can be used to set up a remote connection. It covers background research on Virtual Instrument Software Architecture, base stations and Standard Commands for Programmable Instrument. It also covers research work relevant to this project.

2.1 Base station and its background

This section discusses the role that a base station and its components play in communications system. Base station [5] is important for the user as it permits the usage of system capabilities that the network operator has to offer. Throughout the ages the usage of telephones has varied in many ways. This technology was originally initiated as a mean of communication between two people over hard-wired lines. At the same time two-way radios were introduced known as Mobile Radios but they were in the hands of emergency services and were not made available to the public. They were heavy and had a high power consumption for which the infrastructure of the Mobile Radios were kept at a fixed central location which is now known as the Base Station.

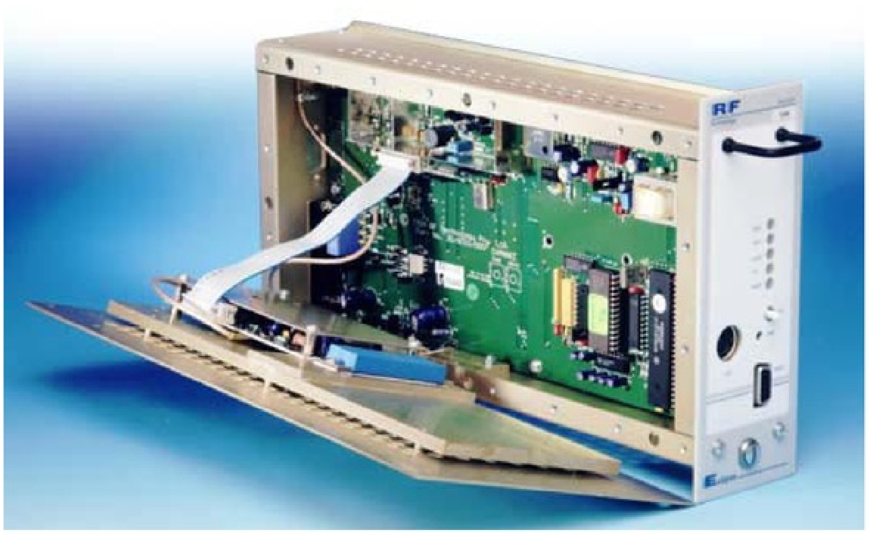


Figure 2.1: Base station used at RF Technology [5]

However with the development of modern technology, base-stations now-a-days have become smaller, much efficient and in most cases portable. This has been possible due to the sophisticated micro-chip designs and advancements in material science.

The two major components of any base station are its receiver and transmitter module. Receiver module are designed and built in a way so that it can receive a particular signal that has the same frequency that it is designed to operate under (typically between 130MHz and 600MHz). It also has design features that helps it to filter out any unwanted noise or signals that can cause distortion within the system. On the other hand a transmitter module is designed to transmit signals within the same frequency range.

2.2 History of instrument automation

Instrument automation is the main scope of this project as the goal is to create an automated test system to monitor the RF characterisation of radios. This section describes the evolution of automation from an early age.

Process control [4]is an engineering discipline that deals with the design, mechanism and algorithm for maintaining the output of a specific process within a desired range. During the early days of process control, indicators and valves were often monitored by someone who had to walk around the test site. The operator had to adjust the valves and other instruments to obtain desired output by varying the readings of the instruments. As technology evolved advanced controlling systems were invented and placed into the test sites. This greatly reduced the amount of time any operator had to spend fixing the instrument readings to obtain the desired output.

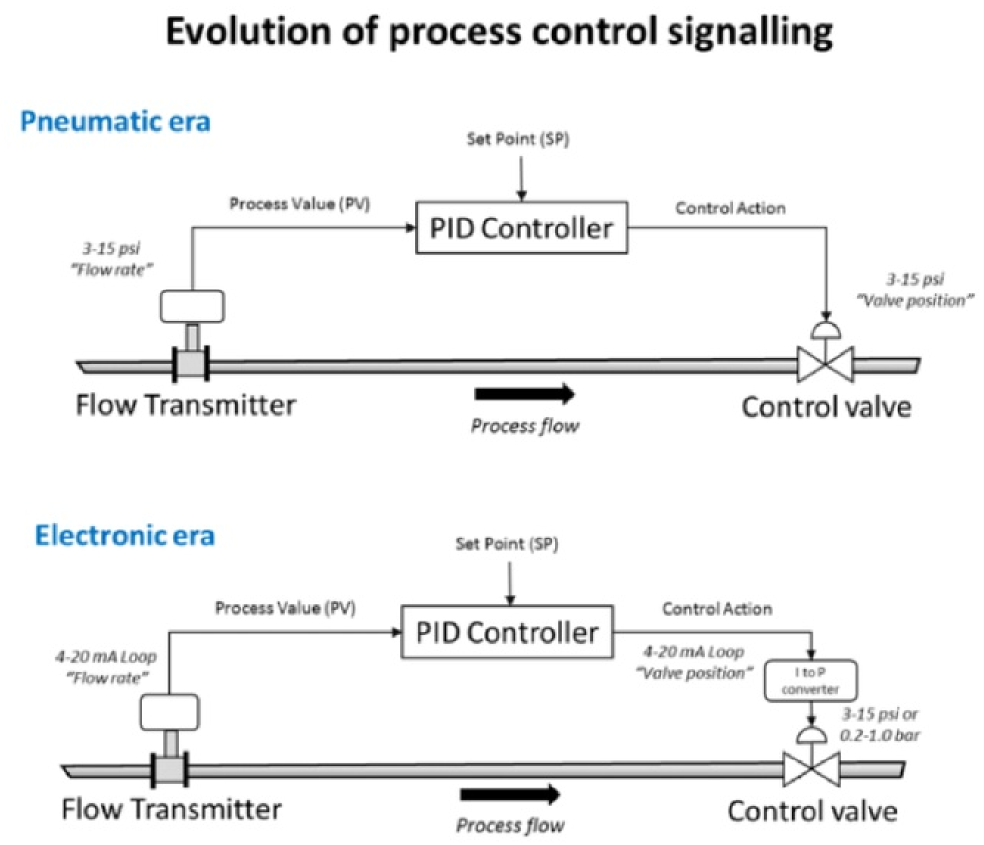


Figure 2.2: The evolution of control system from analogue and pneumatic to electronic era.

The automated process is more like a feedback loop as mentioned above. It involves sending remote commands to signal generators and monitor the output received from it. To send commands and receive output from the signal generator usage of Virtual Instrument Software Architecture (VISA) is necessary. VISA is a popular application programming interface [11] through which a communication medium can be established between test and measurement instruments (such as signal generators) and the computer so that the input and output can be monitored. It is an industry standard introduced by many test and measurement companies such as Keysight, Rohde and Schwarz, National Instruments etc. The VISA standard provides guidelines over how to communicate through High Speed LAN Protocol (HiSLIP), Universal Serial Bus (USB) and General Purpose Interface Bus (GPIB which is also known as IEEE488 bus).

This project has been completed following two of the guidelines mentioned above. A Hi-SLIP [3]connection was established between the signal generators and a GPIB connection was made using the SINAD meter. The signal generators used in this project are manufactured by Rohde and Schwarz and they have all three features to make a connection using the above standards. The SINAD meter however is manufactured by Aeroflex and does not have the USB and HiSLIP feature. Therefore, a different approach (GPIB connection) was necessary to make a successful remote connection.

HiSLIP is an IP based protocol that was introduced so that test and measurement instruments could be operated remotely. HiSLIP is also used via a library that incorporates VISA application programming interface. However GPIB or IEEE 488 was in market long before HiSLIP was introduced. In the year 1960 there was a breakthrough in short-range communication when Hewlett Packard (HP) introduced the IEEE488 in the form of HP-IB (interface bus) to communicate with their own manufactured devices such as multimeters and logic analysers. IEEE stands for Institute of Electrical and Electronics Engineering. Soon after ,HP-IB was commonly known to everyone by the name of general purpose interface bus (GPIB).IEEE has given GPIB [12] its own specification number (488) in 1978 as a result it is commonly known as IEEE488. It was originally created to combine automated test equipment and had a lot of success during the late 1970s and early 1980s when it was used in microcomputers as a peripheral bus.

Figure 2.3: A standard GPIB cable

The GPIB bus allows [12]dataflow between the bus at a suitable speed even for the slowest performing instrument which makes the entire system flexible. This is because the slowest instrument determines the speed of data transfer. It is possible to connect up to 15 instruments in total provided that the bus length does not exceed 20meters. Each instrument has a unique GPIB address between 0 and 30 and the same address cannot be shared by different instruments. However it is possible to assign an address manually to each instrument. Despite having so many features there are still some disadvantages. For example one of the major limitations was that there was no standard format for the instructions sent across the bus.

To overcome this problem, during the year 1987, IEEE introduced standard codes, formats, protocols and common commands and labelled it is IEEE 488.2 [12]. It gave users basic syntax and conventions for formatting. Finally in the year 1990, standard commands for programmable instrument (SCPI) were introduced. SCPI [8]followed the IEEE 488.2 standards and had generic commands however it also worked with IEEE 488 standard. The introduction of SCPI was due to the problems that were arising as there were no instrument specific commands. As an example any command to control a multimeter or logic analyser would vary based on its design and manufacturer. Therefore SCPI was introduced to solve this issue.
Over the years usage of SCPI has improved a lot alongside the development of technology. Initially it was only designed to work with GPIB but now it can also be used with HiSLIP, Bluetooth, USB, RS-232 etc. The signal generator used in this project all supports SCPI version1999.

SCPI commands can be categorized into different groups. However the two most important command structures are the write and query commands. Write commands are used to assign any command according to the users preference. Since its a remote command a user can query the same command to verify the write command that was sent earlier and check if it is correct or not.

For an example to assign the frequency of a signal generator with 155 MHz the following SCPI write command is used:
```
SOUR:FREQ 155 MHz
```

To ensure that this command was sent and received successfully by the signal generator the user can use the following query command:
```
SOUR:FREQ?
```

The Linux terminal would then display only a value of 155000000 without any units showing that the frequency of the signal generator was set to be 155MHz. The allowed physical quantity units can start from Nano and go all the way up to Giga. However if no units are provided basic unit is used. SCPI commands are mostly used in an abbreviated form. SOUR:FREQ actually stands for Source and Frequency where source is the signal generator and frequency is what the signal generator is set to. However it is entirely up to the user in which format they wish to use the commands. Just to illustrate if the user wanted to use the full form they would have typed the code in the following way:
```
SOURce:FREQuency 155 MHz
```

Both of the above approach is accepted by any instrument that supports SCPI standard.

Sub-commands (in this case frequency) within the hierarchy (source) are always assigned with a colon (:). If a user wishes to write multiple commands to an instrument it can be done in a single string separated by a semicolon. For an example,

```
SOUR:FREQ 155MHz; SOUR:FM:DEV: 1.5 kHz
```

This command assigns the signal generator with a frequency of 155MHZ with a modulated deviation of 1.5 kHz. Structure of a command is also important as it decides how fast we can get a response back from the signal generator.

```
SOUR:FM:DEV: 1.5kHz; SOUR:FREQ 155MHz
```

The above line would have done the same work as the first one however the response time would be more. This is slower because it is important to set the frequency first always then set the internal components such as frequency modulation or amplitude modulation.

2.3 Chapter summary

This chapter gave detailed explanations about the history of base stations, instrument automation and VISA applications. In the next few chapters usage of SCPI is discussed more in detail and how the commands can be compiled into a python script that allows successful remote connection. Moreover we would also look into the manual test standards and procedures.

Chapter 3

Manual test measurements and results

When a cell phone or any wireless communication device is used in real-life scenarios it is vulnerable to many factors that can affect its performance. It can face disruptions due to many factors that can result in poor voice quality. Therefore it is highly recommended that both the receiver and the transmitter module undergo rigorous testing in order to be certified for correct usage. This section covers the manual test standards and procedures that a receiver module needs to undergo before it can finally be placed on the base station for using outdoors.

3.1 TIA standards and definition

Telecommunications Industry Association (TIA) [9]has set some standards known as the TIA-568. These standards address commercial building cabling for telecommunications product and services. These standards provide definition, methods of measurement and performance standards for radio equipment that uses frequency modulation or phase modulation with a maximum frequency of 1GHz. Based on the standards that are described in Land Mobile FM or PM Communications Equipment Measurement and Performance Standards [9]the receiver and transmitter module needs to pass certain tests in order to put them finally into a base station. The object of the standard is to standardize parameter titles, definitions, test conditions and methods of measurement to ensure the performance of equipment and to make a possible and meaningful comparison of the test results made by different people on different equipment.

In order for a receiver module to be certified and placed into the base station it needs to undergo and pass four major tests. These tests are important because it comprises of all the scenarios that can affect its applications. For an example if a signal is transmitted from a very long distance it can become weak by the time it is picked up by the receiver. Therefore its ability to pick up weak signals is also an important fact and the test determines how well it does it compared to the TIA standards. Another example can be its ability to reject any unwanted signals that might cause distortion or interfere with the wanted signal. Therefore its ability to tolerate or filter out unwanted signals is also important.

RF Technology [6]performs all the four tests on either a UHF unit or a VHF unit. If multiple units are manufactured they perform the tests only on one unit from a particular line of production. This is because the tests are long enough to be completed on each unit separately. As a result they rely on the design criteria of the units and perform the tests on only one of them. Therefore if that particular unit passes the tests it can be said the rest of the units will pass as well because they are designed and manufactured in the same way and the internal components are same. This is where the scope of this project arises as why the tests need to be automated. Performing all four tests on each base station is time consuming. It takes nearly half a day to set up all the instruments and perform the tests one after another. On top of that the base station is prone to human reading errors and errors due to poor instrument calibration. Therefore to overcome these factors and make the process faster instrument automation is necessary which in turn will automate the tests. Before moving on to the manual test procedures it is important to provide few related definitions as it will help to understand the test better.

- Standard modulation of an input signal this is the modulation due to an input signal of 1000 Hz at a level to produce 60% of the maximum permissible frequency or phase deviation.
- Standard SINAD Standard SINAD is the output of the audio meter with a desired value of 12dB. It is desired to be 12dB because this is the level where human speech quality is perfectly audible.
- Standard input signal frequency this is the frequency of the receiver that can vary depending whether its a UHF or VHF unit.
- Standard input signal level Standard input signal level is -47 dBm.
- Standard input signal this is defined as a RF signal at standard input signal level with standard input signal frequency at the standard modulation of the input signal.

3.2 Manual test measurements and results

RF Technology has designed a software called IP Commander. It is used to assign frequency to the receiver module depending whether its a UHF unit or a VHF unit. The signal generators used in the tests are all assigned the same frequency as the receiver module at the start of each test.

This section covers the entire process of how the four different tests are performed on a UHF unit. The tests are carried out at RF Technology following the TIA standards. The results of the tests are recorded so that they can me compared to the automated test procedures mentioned in chapter 5 of this document. The instruments used for testing the receiver module are mentioned below:

- Rohde and Schwarz signal generator - SMA100A.
- Rohde and Schwarz signal generator - SMB100A.
- Rohde and Schwarz vector signal generator - SMBV100A.
- Aeroflex digital radio test set 3920B.

More details about this instrument and its features and how it can be controlled remotely to automate the tests are given in chapter 4 and chapter 5. The four tests that are required to perform on the receiver module are given below:

- Reference Sensitivity
- Adjacent Channel Rejection
- Spurious Response Rejection
- Inter-modulation rejection

3.2.1 Reference Sensitivity

The reference sensitivity is the level of receiver input signal at a specified frequency with specified modulation which will result in the standard SINAD at the output of the receiver.

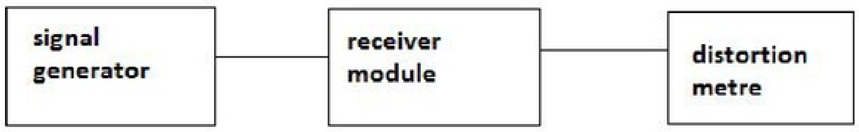


Figure 3.1: Test set-up

The instruments are set up according to the figure above. The signal generator and the SINAD meter is connected to the main A.C power supply and the base station with receiver module is connected to a 12V D.C power supply. In this test only the SMA100A [7] signal generator is used. The test is carried out by following the methods given below:

- The instruments are set-up and connected as per figure3.2.1.1.
- A standard input signal (-47dBm) is applied at the receivers input terminal.

- The reading of SINAD meter is recorder after applying the standard input signal.
- After that the power level of the signal generator is adjusted so that we get a reading of 12dB from the SINAD meter.
- The value at which we get a reading of 12dB from the SINAD meter is recorded. This value is the reference sensitivity or P_{REF}.

After completing the test the reference sensitivity was found to be -110 dB. The TIA standard suggests the value should be lower than -108 dB for better performance.

3.2.2 Adjacent Channel Rejection

The adjacent channel rejection is the ability of the signal generator to reject any signals in the neighbouring channel that can cause distortion. The test setup is given below in figure 3.2.1.2. The signal generators SMA100A (A) and SMB100A (B) are used along with Aeroflex and all of them are connected to the mains A.C. The base station is connected to a 12V D.C power supply.

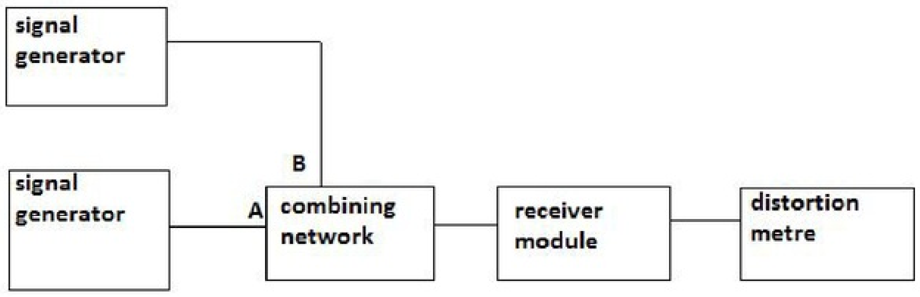


Figure 3.2: Set up for testing adjacent channel rejection

The test is carried out by following the next few steps.

- The instruments are set-up and connected according to the figure.
- A second signal generator, SMB100A (unwanted signal source) is connected to terminal B of the combining network.
- SMB100A is kept off by keeping its RF output off.
- In its absence, the standard input signal is applied at terminal A.
- The reference sensitivity is then calculated using the same procedure as the previous test and the value for which we get an output of 12dB from the SINAD meter is recorded as P_{REF}.

- After recording, P_{REF}, is increased by 3dB. Next the unwanted input signal is applied at terminal B, modulated with 400HZ at 60% of the maximum permissible frequency (12.5 kHz) deviation.
- The frequency of SMB100A is then increased by 12.5 kHz and the power level is adjusted until we got a standard SINAD reading of 12 dB from Aeroflex. This value was recorded as P_{HIGH}.
- The frequency of SMB100A is now decreased by 12.5 kHz and the power level is adjusted until we got a standard SINAD of 12 dB from Aeroflex. This value was recorded as P_{LOW}.
- The adjacent channel rejection is calculated by the following:

  $$\text{Adjacent channel rejection high} = P_{HIGH} - P_{REF}$$
  $$\text{Adjacent channel rejection low} = P_{LOW} - P_{REF}$$

  The smaller value of the above is the adjacent channel rejection.

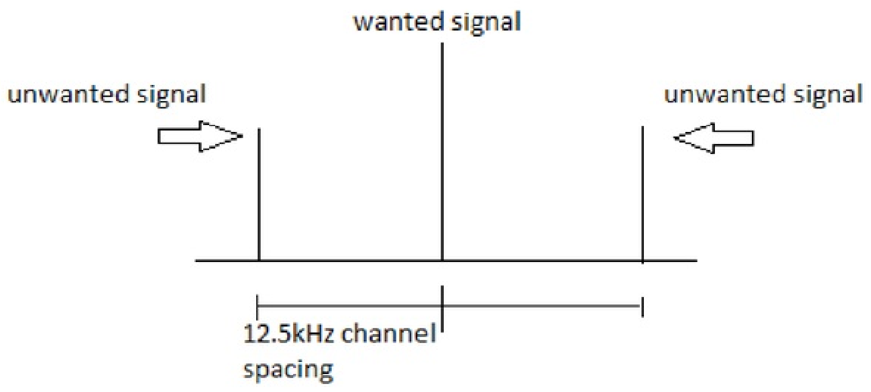


Figure 3.3: Channel spacing between different signals

The Australian government has allocated certain frequency range that can be used by public. The channel spacing between two channels operating on different frequency is 12.5 kHz. Therefore this test checks the ability of the receiver to filter unwanted signals in the neighbouring channels that might cause distortion. After carrying out the test the adjacent channel rejection was found to be 62 dB. This means that the receiver module can tolerate unwanted signals that has a power of maximum 62 dB.However TIA standard suggests that the value should be higher than 60 dB and the higher the value the better is the ability of the receiver to reject unwanted signals.

3.2.3 Spurious Rejection

The spurious response rejection is the ability of a receiver to discard a single unwanted signal from causing degradation to the reception of a desired signal. The test setup and all the equipment are same as the previous test.

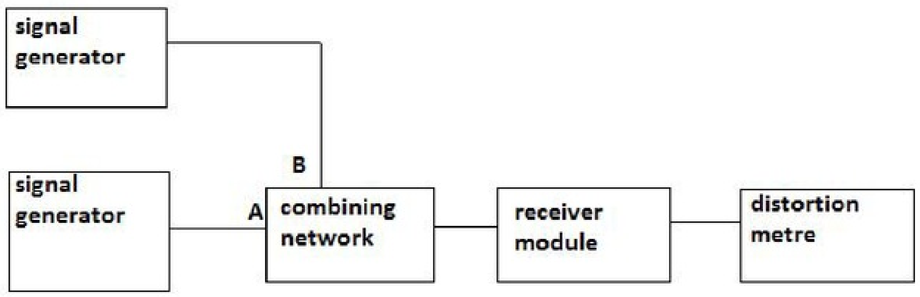


Figure 3.4: Set up for testing adjacent channel rejection

The instruments are set up and connected in the same way as the previous test. RF output of SMB100A is kept off until P_{REF} was calculated.

- The instruments are connected according to figure 3.2.3.1.
- P_{REF} is calculated in the same way as the previous tests.
- After that P_{REF} is increased by 3dB and the frequency of SMB100A is increased by 90 MHz.
- Finally the power of SMB100A is adjusted until we get a 12 dB output reading from the SINAD meter. This value is recorded as P_{HIGH}.

The spurious rejection is calculated as

$$P_{HIGH} - P_{REF}$$

.

After completing this test, the spurious rejection was found to be 75 dB. According to TIA standards the value should be more than 70 dB and the higher the better.

3.2.4 Intermodulation Rejection

The inter-modulation rejection is the ability of a receiver to discard two unwanted input signals from causing distortion to the reception of a desired signal. In the previous test we saw how one unwanted signal was discarded. In this test we will see how two equal signal can be discarded to provide a better performance for the receiver module.

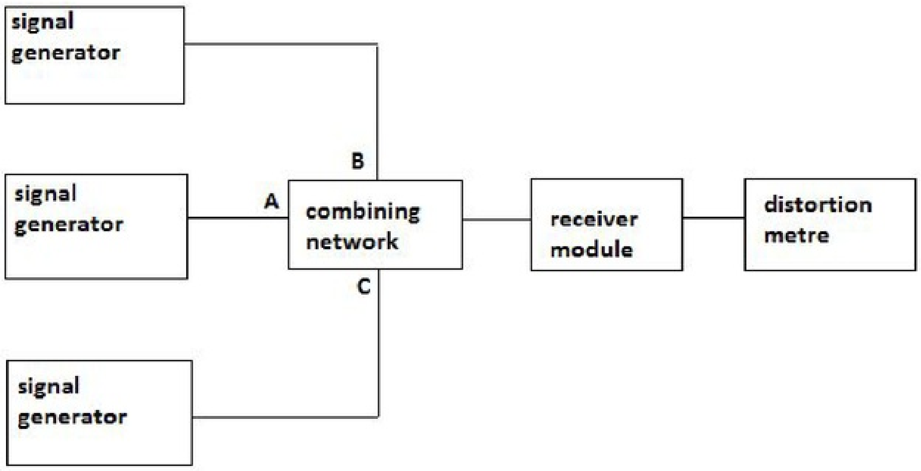


Figure 3.5: Test setup for inter-modulation rejection

The instruments used in this test are same as the previous one with the exception of another signal generator (C) added. The third signal generator is SMBV100A. All the test instruments are connected to the mains A.C line and the base-station to a 12V D.C supply.

- The instruments are set-up and connected according to figure 3.2.4.1.
- Two signal generators (unwanted signal sources) are connected to terminal B and C respectively.
- P_{REF} is calculated in the same way as the previous tests and by keeping the RF output of SMB100A and SMBV100A turned off.
- P_{REF} is then increased by 3 dB.
- SMB100A (signal generator B) is used to apply an unwanted and unmodulated input signal at terminal B. The frequency of SMB100A is made to be 50 kHz more than the frequency of SMA100A.
- Next SMBV100A is used to apply a signal at terminal C modulated with 400 Hz and 60% of the permissible frequency deviation. Frequency of SMBV100A is made to be 100 kHz more than the frequency of SMA100A.
- Next the power levels of both the unwanted signal generators are made equal.

- After that they are both adjusted by keeping the power value equal until we get a standard SINAD reading of 12 dB from the Aeroflex instrument. The value for which we get a 12 dB reading is recorded as P_{HIGH}.
- The above steps are repeated by decreasing the frequency of SMB100A by 50 kHz and the frequency of SMBV100A by 100 kHz.
- The value for which we get a reading of 12 dB from the SINAD meter is recorded as P_{LOW}.
- The inter-modulation rejection is calculated as follows:
$$\text{Inter-modulation rejection high} = P_{HIGH} - P_{REF}$$

According to the TIA standards the inter-modulation rejection should be higher than 79 dB and the higher the better. After completing the above steps the inter-modulation rejection was found to be 80 dB.

3.3 Chapter summary

In this chapter we discussed the TIA standards and the four major tests that the receiver module needs to undergo in order to be certified usable in real life scenarios. Detailed explanation about the test setup, methods for measurement and results were also provided so that it can be compared to the automated test results in Chapter 5 of this document. In the next chapter we will see two different approaches on how to establish a remote connection with a signal generator, SINAD meter and how SCPI commands can be used to assign commands remotely.

Chapter 4

Setting up a remote connection

In this chapter we will discuss how a remote connection is made between the test instruments (signal generators and SINAD meter) and the computer. We will be focusing on two approaches. One of them is a remote Ethernet connection between the signal generators and the computer and the other is a GPIB connection between the SINAD meter and the computer. A detailed explanation for both the approaches is provided within the next few sections of this chapter.

4.1 Virtual Box and Linux

The signal generators that were used for this research project are manufactured by Rohde and Schwarz and the SINAD meter by Aeroflex. By default they have Linux based system installed within them. For this reason we had to work with a computer that had a Linux system kernel installed in it. The reason for doing so is because the SCPI commands (mentioned in chapter 1 and 2) sent to the signal generator or SINAD meter wont be read by the instruments if it is sent from any other operating system other than Linux. As a result both the computer and the test instruments needed the same operating system.

Since the computer we used had windows operating system in it we had to find a way to install Linux on it. After a thorough online research we came across a software called Virtual Box (VB). A Virtual Box is a software package that can be installed on any operating system as an application. It allows additional operating system that can be installed on it and can be run in a virtual environment. [11]

We downloaded the latest free version of Virtual Box designed by Oracle. Since Linux is open-source, there are heaps of operating system variants such as Debian, Ubuntu etc. In our project we have used Ubuntu as it is more popular and user-friendly compared to the other variants. We followed the step-by-step instructions from Oracle on how to install Ubuntu correctly. The process was fairly simple as we just had to pick the Linux operating system and Ubuntu from a list of options. Then we had to choose the system memory size that had to be allocated to the operating system. By following these steps

Virtual Box created a Linux system kernel with Ubuntu as an operating system variant.

Linux has a command line known as the shell or terminal. It is same as the command prompt in windows. This is where we can run the python scripts, change directories of any saved file, import any python libraries etc. If there are any errors while carrying out any commands or python scripts, the detailed error message and the possible reason will be displayed in the terminal window.

4.2 Python programming language

We needed to understand the python language as it is highly recommended by RF Technology. Most of their work is done using python programming. As they will be referring to the automated test scripts in future it is important that they find it easier to use and change it if necessary.

We installed python as it is free to download from the vendor. Python files have a .py extension. They can be written and edited in the Linux writing tool known as Gedit. It is not too hard to learn python if anyone has previous programming language experience. It is similar to other languages such as C, C+, C#, Java etc. The loops (for and while) within a python code are same as any other language. The best part is we can import any python library from within a script instead of downloading it separately. The few libraries that we needed for this project are discussed later in this chapter.

4.3 Test instruments and automation

This section discusses in detail regarding how a remote connection was made between the signal generators and the SINAD meter. A thorough explanation is given about how each line of SCPI codes generated the output desired even there were two different types of connection.

The signal generators used throughout this project are manufactured by Rohde and Schwarz and as mentioned before they all have a Linux operating system. The three signal generators are R&S SMA100A, SMB100A and SMBV100A. They all can be controlled using the same SCPI codes as they have the same manufacturer however their features are different as they have different frequency range built-in. For simplicity we will just discuss the remote connection made using RS SMA100A in this chapter.

4.3.1 Remote connection of a signal generator

The next step to establish a remote connection is to import few python packages. The two packages that were imported for the signal generators are Numpy and VISA (Virtual Instrument Software Architecture). Numpy [13] is the basic package used in Python for scientific computing. It contains useful tool for computing linear algebra, Fourier transform and random numbers. It also contains tools to integrate C/C++ and other codes. To install Numpy using Linux terminal the following code is used:

```
sudo apt-get install python-numpy
```

Linux terminal would then run this command and install numpy from python libraries.

Another package that is required to establish a successful remote connection is PyVISA [14]. This is required because it enables us to control the desired instrument independent of its interface (i.e Ethernet, GPIB, RS-232). Since we have two different methods of setting up a remote connection, PyVISA is required. VISA is a standardized software interface library providing input and output functions to communicate with instruments. In order to make sure that PyVISA works we need to have a suitable backend. Rohde and Schwarz have their own VISA library that can be downloaded from their website for free. PyVISA can work with any bits of VISA libraries but Python and PyVISA needs to be of same bits. It can be installed through Linux terminal using the following piece of code:

```
$ pip install U pyvisa
```

To check that PyVISA is installed correctly we can create a resource manager. The following piece of code does the checking:

```
import visa
rm = visa.ResourceManager()
print (rm.list_resources())
```

Resource Manager is a class that is stored at rm and it is later called to open the IP address of the signal generator. The signal generator has a Dynamic Host Configuration Protocol (DHCP) [2] and a static internet protocol (IP) address. DHCP has a dynamic address range and can take up any value from 0 to 254. This means the test instrument can take up any IP address between these values if we assign it to DHCP. However we can assign any particular IP address by selecting static IP from the control panel of the instrument. RF Technology has designed its own DHCP network throughout the building. As a result all the signal generators are assigned to have DHCP starting with 192.168.1.(0-254).

The signal generator has a built-in Ethernet, USB and GPIB interface. A Local Area Network (LAN) cable is attached to the Ethernet port of the signal generator and the other end to a switch of the DHCP server that is connected to the buildings network. The IP address of the instrument can now be found from the control panel of the instrument. However if a user wants to change it to a static IP it can be done manually using the configurations on the control panel.

The next step is to talk to the signal generator by sending remote SCPI commands through a python script. As mentioned earlier python scripts have a .py extension. In

order to run a file the following command is used:
```
./filename.py
```

Linux will then run the file successfully provided there are no errors within the script. The first line of any python script can have this command: #!/usr/bin/python. This means Linux is directly asking python to run all the codes that are written below that line. However it is possible to run a script without this line and the could would be
```
python filename.py
```

Next step is to import the python package numpy and the VISA library using the following piece of code:
```
import numpy as np
import visa
```

After that the IP address of the signal generator is mentioned by calling the Resource Manager. As discussed before the IP address of the signal generator can be found from the control panel after an Ethernet cable is connected to the signal generator and the DHCP server switch.
```
rm = visa.ResourceManager(@py)
my_instrument1 = rm.open_resource(uTCPIP::192.168.1.18::INSTR)
```

In the above line my_instrument.1 is the signal generator SMA100A. The VISA Resource Manager is calling PyVISA so that VISA libraries that were imported before can be used. TCPIP stands for Transmission Control Protol/Internet Protocol. The IP address is part of the visa resource string that is used by the programs to identify and control the instrument.

SCPI command structure [8] has a status reporting system. It stores all the information in registers regarding the current state of the instrument and errors if any occurred. It is possible to query both of them using certain SCPI commands. There are five different parts of each status registers and each register has a width of 16bits (0 to 15). Each bit represents a different meaning in the status reporting system. In this project we only used two status registers called the status byte register and status event register. The following two tables give a clear description regarding the meaning of each bits.

Table 4.1: Meaning of bits used in the status event register

Bit Number	Meaning
0	Operation is complete (meaning all commands have been executed and the event status is updated to bit 1)
1	Not Used
2	Query error (there is a query command that is faulty and cannot be executed)
3	Device,dependant error (a number between -300 and -399 is shown meaning there is an,error)
4	Execution error (meaning the command sent is syntactically correct however it cannot be executed for other reasons and a number between -200 and -300 is shown)
5	Command error (meaning the command sent is undefined or syntactically incorrect and a number between -100 and -200 is shown)
6	User request (this means that the instrument has exited from remote to manual mode following users request)
7	Power on

The following table shows how the bits were used in status byte register.

Table 4.2: Meaning of bits used in status byte register.

Bit Number	Meaning
0 and 1	Not Used
2	Error queue not empty (meaning this bit is set whenever there is an error)
3	Questionable status register (this bit is set if any event occurs in the status register)
4	Message,available (meaning there is a message in the output that can be read)
5	Error,status bit (there is a major error)
6	Status summary bit (meaning there is a service request)
7	Register,summary bit (this bit means that the instrument is performing an action)

Before we go to the last part of this section the following commands needs to be mentioned and explained:

- *IDN? (Identification) This is a query only command. Once this command is sent it returns the identification of the instrument in the format Device name, device type, serial number, firmware version.

- *RST (Reset) Its a setting only and is neither a write nor a query command. It sets the instrument into a default state.

- *CLS (Clear Status) It clears the current status of the instrument along with all the outputs.

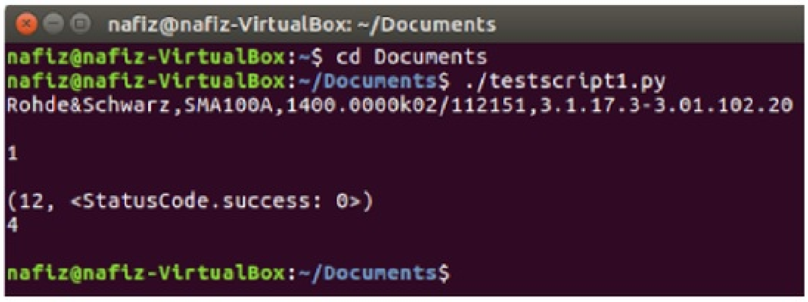
- *OPC (Operation Complete) This means that operation is complete and it blocks the status registers until all the commands have been executed.

- *ESR?(Error Status Register) Its a query only command and reads the current status of the event.

The following piece of code is now used to establish a successful remote connection between the signal generator and the computer using a python script.

```
!/usr/bin/python
import numpy as np
import visa
rm = visa.ResourceManager (@py)
my_instrument1 = rm.open_resource(uTCPIP::192.168.1.18::INSTR)
idn = my_instrument1.query(*IDN?)
print(idn)
my_instrument1.write(*rst; status:preset; *cls)
print my_instrument1.query(*OPC?)
print my_instrument1.write(SYST:SERR?)
print my_instrument1.query(ESR?)
```

After executing these remote commands the following output is received in the Linux terminal showing that the connection was successful.



```
nafiz@nafiz-VirtualBox: ~/Documents
nafiz@nafiz-VirtualBox:~$ cd Documents
nafiz@nafiz-VirtualBox:~/Documents$ ./testscript1.py
Rohde&Schwarz,SMA100A,1400.0000k02/112151,3.1.17.3-3.01.102.20

1

(12, <StatusCode.success: 0>)
4

nafiz@nafiz-VirtualBox:~/Documents$
```

Figure 4.1: Successful remote connection between signal generator and the computer.

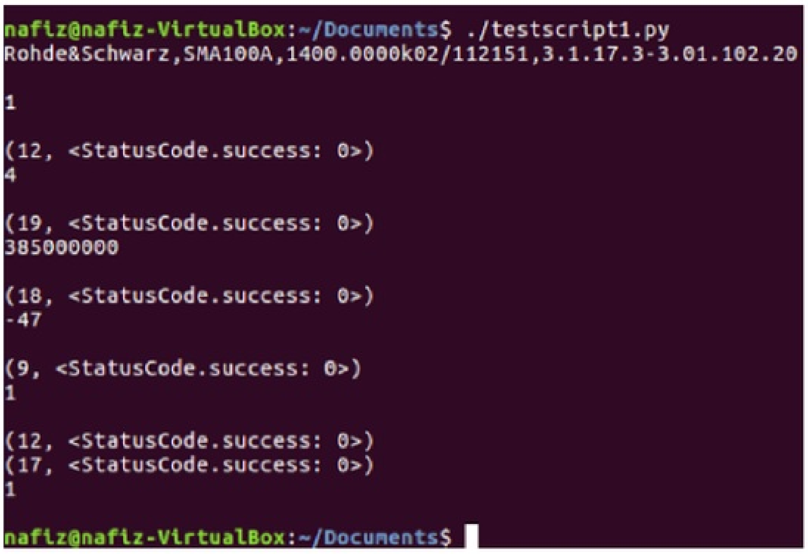
From the figure above it can be said that the status registers are not being used (1) and there is a message available to read(4).

The filename was saved as testscript1.py in the documents folder. Therefore the current directory (cd in the above image) was changed at first before executing the file. The output of the *IDN command is returned as Rohde and Schwarz, ¡device type¿, ¡serial number¿, ¡firmware version¿. The status byte and status event register bits are shown as well and the description of them are given above in the tables above.

After the connection is successful it is now possible to send SCPI commands to the generator to do various tasks. A simple code is given below to provide an example.

```
print my_instrument1.write(SOUR:FREQ 385 MHz)
print my_instrument1.query(SOUR:FREQ?)
print my_instrument1.write(SOUR:POW -47 dBm)
print my_instrument1.query(SOUR:POW?)
print my_instrument1.write(OUTP ON)
print my_instrument1.query(OUTP?)
print my_instrument1.write(FM:STAT ON)
print my_instrument1.query(FM:STAT?)
```

The above code assigns a frequency of 385MHz and a power of -47dBm to the signal generator and then turns on the RF output and Modulation.



```
nafiz@nafiz-VirtualBox:~/Documents$ ./testscript1.py
Rohde&Schwarz,SMA100A,1400.0000k02/112151,3.1.17.3-3.01.102.20

1

(12, <StatusCode.success: 0>)
4

(19, <StatusCode.success: 0>)
385000000

(18, <StatusCode.success: 0>)
-47

(9, <StatusCode.success: 0>)
1

(12, <StatusCode.success: 0>)
(17, <StatusCode.success: 0>)
1

nafiz@nafiz-VirtualBox:~/Documents$
```

Figure 4.2: Assigning frequency and power and turning on the outputs.

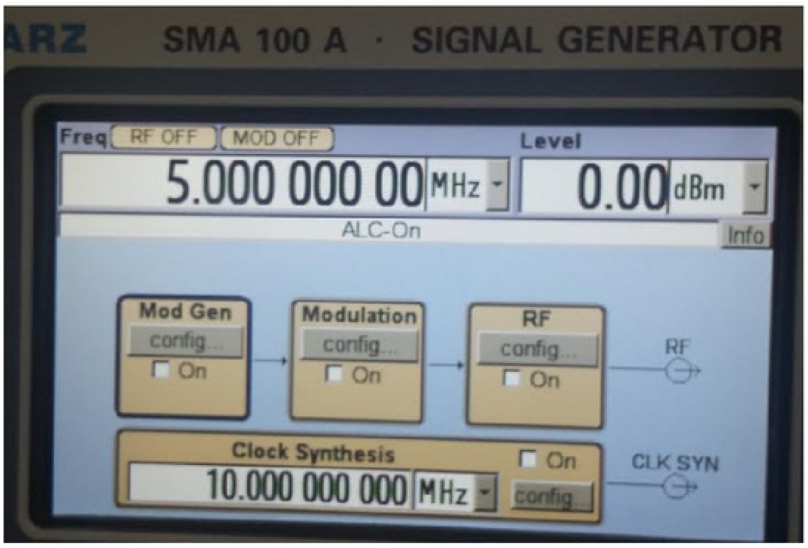


Figure 4.3: Display of signal generator before executing the above code

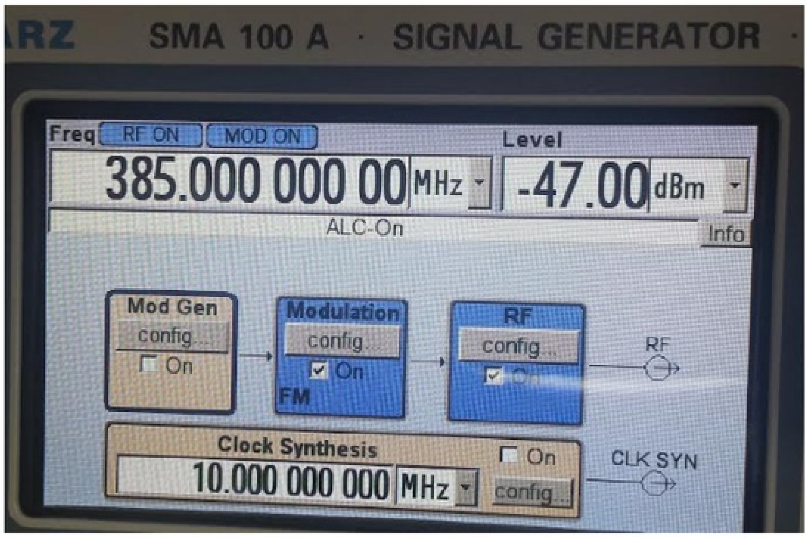


Figure 4.4: Display of signal generator after the execution of the code.

4.3.2 Remote connection of the SINAD meter (Aeroflex)

The SINAD meter used in this project is manufactured by Aeroflex and its model number is 3920B. Unlike the signal generators that have multiple interfaces manufactured by Rohde and Schwarz, this has only a GPIB interface built in it. As a result the steps to establish the remote connection were a bit different.

At first a telnet library was required to setup the connection. Telnet is a protocol that allows user to connect to remote computer known as hosts over a TCP/IP network" [10]. A connection with a telnet server is possible if the computer is running a telnet client server. After the telnet establishes a connection to the host it is possible to control it remotely. The telnetlib module in python can implement the telnet protocol. The following command was used to import the telnet protocol in python.

```
import telnetlib
```

If the above code is written in the top of the script python automatically imports and implements the telnet protocol [10]. While researching online regarding how to implement the remote connection using the SINAD meter we came across a python file called nfcli.py that had to be downloaded from python for free. This file was created so that new users didnt have to write all the codes that help to implement the telnet protocol. It contains codes that searches for any GPIB interface available in the network and allows modifying it. The following command was used to search for the available interfaces

```
python nfcli.py -list
```

Prologix GPIB-Ethernet controller was used to connect one end of the GPIB cable to the SINAD meter and the other to the controller. The Ethernet port of the controller was then connected to the DHCP server switch using a LAN cable as a result the IP address of the controller was found. After downloading the file it was run in Linux terminal using the code: python nfcli.py. Since it was a precompiled script it didnt have the #!/usr/bin/python at the top as a result python was put at the beginning of the file name. After executing the file the GPIB controller was found with its own IP address.

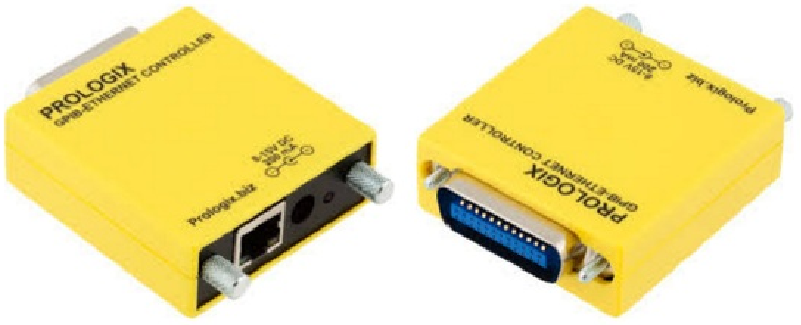
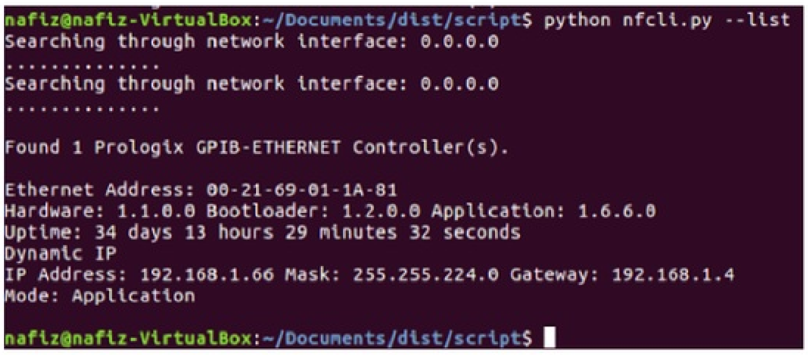


Figure 4.5: Prologix GPIB-Ethernet controller used to establish a remote connection.



```
nafiz@nafiz-VirtualBox:~/Documents/dist/script$ python nfcli.py --list
Searching through network interface: 0.0.0.0
..............
Searching through network interface: 0.0.0.0
..............

Found 1 Prologix GPIB-ETHERNET Controller(s).

Ethernet Address: 00-21-69-01-1A-81
Hardware: 1.1.0.0 Bootloader: 1.2.0.0 Application: 1.6.6.0
Uptime: 34 days 13 hours 29 minutes 32 seconds
Dynamic IP
IP Address: 192.168.1.66 Mask: 255.255.224.0 Gateway: 192.168.1.4
Mode: Application

nafiz@nafiz-VirtualBox:~/Documents/dist/script$
```

Figure 4.6: Identifying the IP address of the controller.

The Aeroflex 3920B has a default telnet port number of 1234 and a dynamic GPIB address of 3. After this a python script was made to communicate to the instrument. It was possible to talk to the signal generator using its own IP address however the SINAD meter didnt have an Ethernet port. Therefore the IP address of the GPIB controller was used which then communicated to the SINAD meter that had a GPIB address of 3. As a reason there was a delay in receiving the output from the SINAD meter. This was a problem as the telnet connection timed out after few seconds without returning any values.

Therefore, we had to import another python module called time. It basically holds the telnet connection for a certain amount of time until the output is received from the SINAD meter. The amount of time is mentioned by the user in the python script. The following code was used at the beginning of the script to import time:
`import time.`

The Prologix controller in this case was set to behave as a controller unlike the computer in the previous remote connection. This is because the SINAD meter is connected to the controller through its only available interface. As a result it is thinking that it is connected to the computer and would accept any SCPI commands sent to it. The SINAD meter generates an output audio value that is not steady. It fluctuates all the time and for this reason we needed a SCPI code that would perform an average function on the SINAD meter. The following code gives an example of how the SINAD meter can give an average output by using a SCPI code.

```
:CONFigure:AF:ANALyzer:SINad:AVERage 3
```

The most widely used abbreviated form is
`:CONF:AF:ANAL:SIN:AVER 3`

The query command is
`FETCh:AF:ANAL :SIN ?`

The SINAD is asked to return an audio output by taking an average of 3 readings. Therefore the python script to communicate with the SINAD meter has the following code:

```
#!/usr/bin/python
import time
import telnetlib
tn = telnetlib.Telnet(192.168.1.66 , 1234)
tn.write (++mode 1\n)
tn.write(++auto 1\n)
tn.write(++addr 3\n)
tn.write(*IDN?\n)
tn.write(CONF:AF:ANAL:SIN:AVER 3\n)
tn.write(FETC:AF:ANAL:SIN?\n)
```

The IP address used in the above code is of the GPIB-Ethernet controller and 1234 is the default telnet port number of Aeroflex. After that it is set to be in the controller mode and GPIB address 3 of the Aeroflex instrument was accessed. The *IDN works the same way as the signal generator and prints the instrument name, type, serial number etc.

4.4 Chapter summary

In this chapter we saw how two different approaches were used to connect the two different test instruments. For the signal generator, all the necessary libraries and how to download/import and install them were mentioned. Detailed explanation about the status registers and bits were given. Examples of simple SCPI commands to were given to remotely set the frequency and power of the signal generator.

For the SINAD meter the two libraries and telnet protocol was also discussed. Moreover how the GPIB-Ethernet controller was used to control the instrument was also mentioned. Simple SCPI commands were also included to provide an understanding of average readings. In the next chapter we will see how the two methods discussed above can be combined together and create an automated test process that were manually done in Chapter 3.

Chapter 5

Automated tests

In the previous chapter we discussed how it was possible to establish a remote connection between the test instruments. Two different connections were made a GPIB connection and a HiSLIP connection. In this chapter we will discuss how the remote commands can be compiled together using python programming and automate the tests that we did in chapter 3.

5.1 Reference Sensitivity

In this test we used the signal generator SMA100A and the Aeroflex as the SINAD meter. As mentioned in chapter 4, Aeroflex was connected using a GPIB and the signal generator using Ethernet. The frequency of the receiver module was assigned by the IP commander and the same frequency was assigned to the signal generator. The instruments were connected to the A.C mains whereas the base station was connected to the 12V D.C supply. These tests were carried out using a UHF unit which had a frequency of 155 MHz. The standard input signal of -47 dBm was applied using the following piece of code:

```
print my_instrument1.write(SOUR:POW -47 dBm)
```

To ensure that the command was executed successfully a query command was sent as well.

```
print my_instrument1.query(SOUR:POW?)
```

The output was -47 dBm in the display of SMA100A. The RF output and modulation output was turned on using the following remote command:

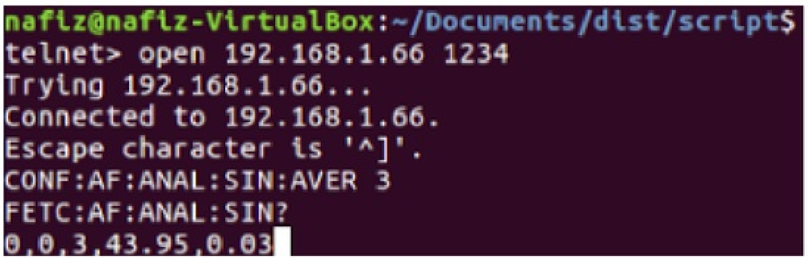
```
print my_instrument1.write(OUTP ON) RF output
print my_instrument1.write(FM:STAT ON) modulation output
```

The next step was to change the power level until we got an output of 12 dB from the SINAD meter. Before the code is given it is important to describe few terms that were used to automate these tests.

Since we are expecting a returned value of 12 dB from the SINAD meter a lower limit of 11 dB and an upper limit of 13 dB were created. This is because the SINAD meter does not give a steady reading as mentioned before. The code was made in such a way that we would get an average value from the SINAD meter and also if it had any value between 11 dB and 13 dB the code would exit the loop to give the power reading.

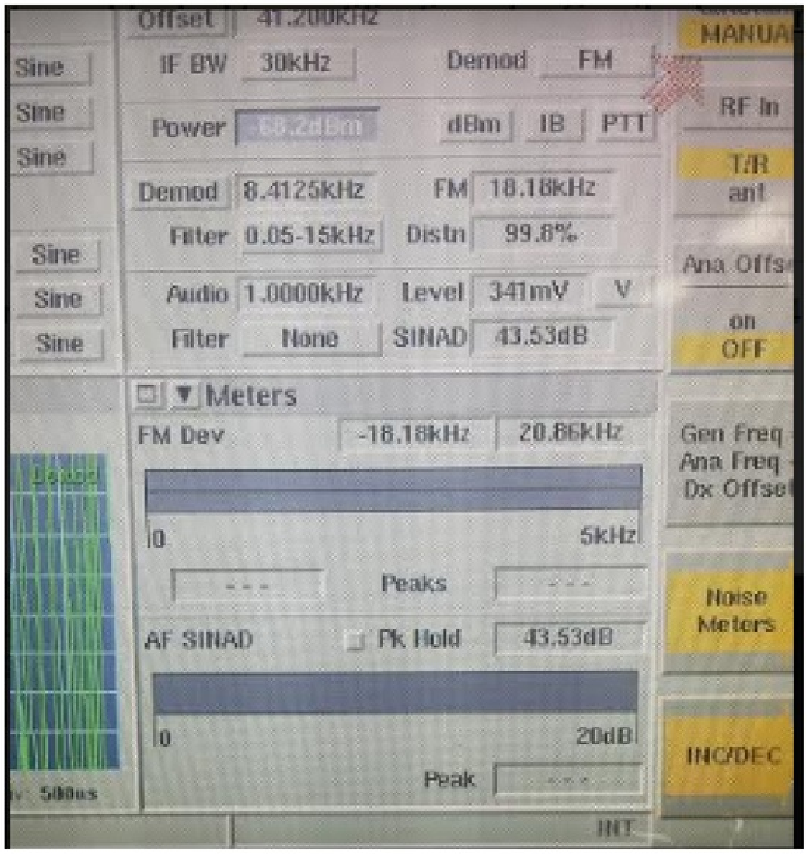
From the manual test it is known that the SINAD would return a value of 12 dB for reference sensitivity of -110 dB. As a result a step function was created. The signal generator was asked to start the readings from -117 dB in steps of 1 until it got to a value for which the SINAD was 12dB.

As mentioned earlier it takes time to receive readings from the Aeroflex because it was controlled by Prologix using a GPIB cable and the Prologix was connected to the computer through Ethernet. The SCPI commands was first sent to Prologix which would then send it to Aeroflex to perform actions. The output is received in the same process as a result there is a delay. Therefore, there is a high possibility that the signal generator would update its reading before the SINAD meter updated. As a result, we had to import the time library in python to implement a wait statement so that the audio value gets updated before the signal generator. The SINAD meter returns output in the following format:



```
nafiz@nafiz-VirtualBox:~/Documents/dist/script$
telnet> open 192.168.1.66 1234
Trying 192.168.1.66...
Connected to 192.168.1.66.
Escape character is '^]'.
CONF:AF:ANAL:SIN:AVER 3
FETC:AF:ANAL:SIN?
0,0,3,43.95,0.03
```

Figure 5.1: Output format of SINAD meter in the Linux terminal.



```
Offset 41.200kHz
IF BW 30kHz        Demod FM
Power -68.2dBm     dBm  IB  PTT
Demod 8.4125kHz    FM 18.18kHz
Filter 0.05-15kHz  Distn 99.8%
Audio 1.0000kHz    Level 341mV  V
Filter None        SINAD 43.53dB
Meters
FM Dev   -18.18kHz   20.86kHz
Peaks  ---  ---
AF SINAD   Pk Hold   43.53dB
Peak ---
```

Figure 5.2: Output displayed in the SINAD meter.

The output is displayed in the terminal as a string in the format status byte, fail byte, average count, average, wc. The output in the example above returns a string of 0, 0, 3, 43.95, 0.

This means that the status byte is valid (0) and all limits are checked and passed (0), average count is 3 and the output of the SINAD metre is 43.95 dB. WC represents worst case which is 0 meaning the commands were executed without any error. The full table that explains each bits of status byte [1] and fail byte are given below:

The code to read the audio value is given below:

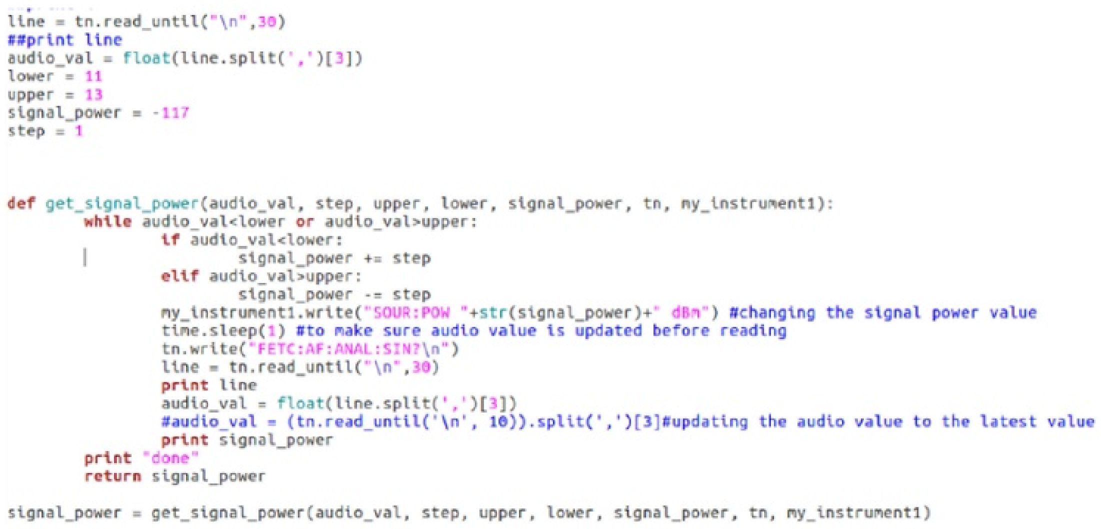
```
line = tn.read_untill(\n, 30)
audio_val = float(line.split( , )[3])
```

Table 5.1: Status Byte description for Aeroflex

Status Byte	**Bit number**	**Fail byte**
Valid	0	All limits checked and passed
Invalid	1	Not used
Settling	2	Average lower failed limit
Not used	3	Not used
Inaccurate	4	Not used
Not used	5	Not used
Settling and inaccurate	6	Not used
Settling, inaccurate and invalid	7	Not used
Not used	8	Worst case lower failed limit

The above code means that the connection needs to wait at least 30seconds before it is timed out or until a new line of code is generated (\n represents a new line). Since we are getting decimal values float is used. Moreover the output of the SINAD meter is the 4th value in the string. Therefore the above command splits all the commas and accesses the 4th value in the string and returns the output.

The code to carry out reference sensitivity is now given below:



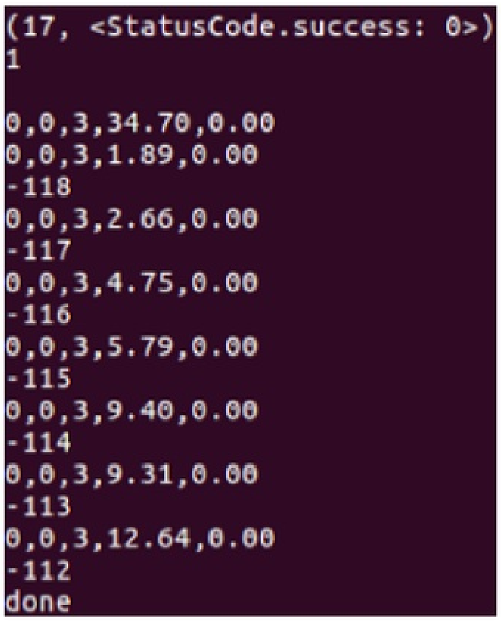
```
line = tn.read_until("\n",30)
##print line
audio_val = float(line.split(',')[3])
lower = 11
upper = 13
signal_power = -117
step = 1


def get_signal_power(audio_val, step, upper, lower, signal_power, tn, my_instrument1):
        while audio_val<lower or audio_val>upper:
                if audio_val<lower:
                        signal_power += step
                elif audio_val>upper:
                        signal_power -= step
                my_instrument1.write("SOUR:POW "+str(signal_power)+" dBm") #changing the signal power value
                time.sleep(1) #to make sure audio value is updated before reading
                tn.write("FETC:AF:ANAL:SIN?\n")
                line = tn.read_until("\n",30)
                print line
                audio_val = float(line.split(',')[3])
                #audio_val = (tn.read_until('\n', 10)).split(',')[3]#updating the audio value to the latest value
                print signal_power
        print "done"
        return signal_power

signal_power = get_signal_power(audio_val, step, upper, lower, signal_power, tn, my_instrument1)
```

Figure 5.3: Code for reference sensitivity.

The signal power is defined as a function because from the manual tests we have seen that reference sensitivity is always measured at the start of each test. Therefore it is easier to leave it as a function so that it can be called later. The audio value, step, upper, lower, signal power, tn and my_instrument1 are all stored as a variable within the function so that when it is edited, the value will get updated automatically throughout the code wherever it is used.

All the parameters are defined before the function is declared. A simple while loop is used to check the current signal power and audio value. If the audio value is lower than 11 dB the signal power is increased by steps of 1 and if it is higher than 13 dB the power is decreased by steps of 1. The power value updates automatically in the display of SMA100A and a 1second delay is made to make sure that the audio value gets updated before reading the signal power. This loop continues until an average floating point value of the SINAD meter is found between 11 dB and 13 dB.

The full code for this test is provided in the appendix section at the end of this document. After carrying out this experiment the reference sensitivity was found to be -112 dB. The figure below shows the final output after the execution of the code.



```
(17, <StatusCode.success: 0>)
1

0,0,3,34.70,0.00
0,0,3,1.89,0.00
-118
0,0,3,2.66,0.00
-117
0,0,3,4.75,0.00
-116
0,0,3,5.79,0.00
-115
0,0,3,9.40,0.00
-114
0,0,3,9.31,0.00
-113
0,0,3,12.64,0.00
-112
done
```

Figure 5.4: Automated test result for reference sensitivity.

5.2 Adjacent Channel Rejection

Two signal generators SMA100A and SMB100A were used in this test along with Aeroflex. Both the generators are assigned a frequency of 155 MHz using the same code mentioned above. The RF output of SMB100A is kept off by using the following code:

```
print my_instrument2.write(OUTP OFF)
```

SMB100A is modulated with 400 Hz and 60% of the permissible frequency using the following code:

```
print my_instrument2.write(SOUR:FM:DEV 1.5 kHz)
print my_instrument2.write(SOUR:LFO:FREQ 400 Hz)
```

The above code sets the FM deviation to be 1.5kHz and the LF frequency to be 400 Hz. The code to turn on the RF output and increase and decrease the frequency by 12.5 kHz respectively is given below:

```
print my_instrument2.write(OUTP ON)
print my_instrument2.write(SOUR:FREQ 155.0125 MHz)
print my_instrument2.write(SOUR:FREQ 154.9875 MHz)
```

The short code for the automated test is given below:

```
p_ref = signal_power
signal_power + = 3 # increase wanted signal power by 3 db
interfere_power = p_ref + 75
def get_interfere_power(audio_val, step, upper, lower, signal_power,
tn, my_instrument2, my_instrument3=None):
while audio_val<lower or audio_val>upper:
if audio_val<lower:
signal_power - = step
elif audio_val>upper:
signal_power + = step
my_instrument2.write("SOUR:POW "+str(signal_power)+" dBm")
changing the signal power value
if my_instrument3:
my_instrument3.write("SOUR:POW "+str(signal_power)+" dBm") changing
the signal power value
time.sleep(1) to make sure audio value is updated before reading
```

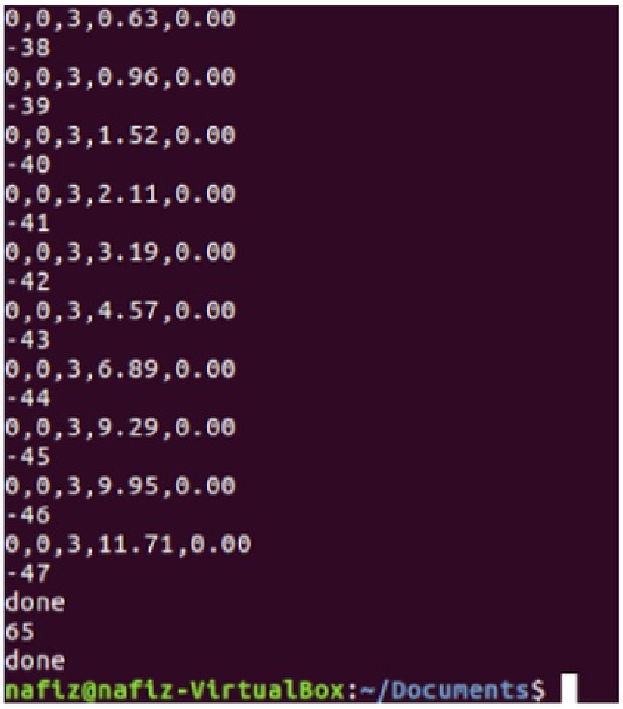
```
tn.write("FETC:AF:ANAL:SIN?\n")
line = tn.read_until("\n",30)
print line
audio_val = float(line.split(',')[3])
print signal_power
```

At start P_{REF} is calculated using the same procedure as the previous test. Then 3 dB is added to P_{REF}. After completing the manual test the adjacent channel rejection was found to be 62 dB. Therefore, we know that we are looking at a value around that. As a result the interfere power (power of SMB100A) was increased by 75 dB. By doing this the time required to complete the test would be less as the loop will now have less values to work with. The same function is used by making two changes. Since SMB100A is an interferer the function name is changed and SMBV100A is included as a variable so that this function can be called when we are working with a test that needs SMBV100A. However it doesnt affect the code because nothing is assigned to SMBV100A.

A simple while loop is used which checks the current status of the audio value. If it is less than 11 dB it decreases the signal power by steps of 1 and if it is more than 13 dB it increases the power by steps of 1. The RF output is now turned on of SMB100A and the frequency is increased and decreased by using the codes in the previous page. The signal generator now updates its values each time until the SINAD meter gets a reading that is between 11 dB and 13dB. This step is repeated for both the high and low frequency. After that the adjacent channel rejection is calculated using the following code:

```
p_high =get_interfere_power(audio_val,step,upper,lower,
interfere_power,tn,my_instrument2)
p_low =get_interfere_power(audio_val,step,upper,lower,
interfere_power,tn,my_instrument2)
rejection_high = p_high - p_ref
rejection_low = p_low - p_ref
rejection = rejection_high if rejection_high < rejection_low else
rejection_low
print rejection
```

The full code for this test script is provided in the appendix section at the end of this document. After running this script the adjacent channel rejection was found to be 65 dB.



```
0,0,3,0.63,0.00
-38
0,0,3,0.96,0.00
-39
0,0,3,1.52,0.00
-40
0,0,3,2.11,0.00
-41
0,0,3,3.19,0.00
-42
0,0,3,4.57,0.00
-43
0,0,3,6.89,0.00
-44
0,0,3,9.29,0.00
-45
0,0,3,9.95,0.00
-46
0,0,3,11.71,0.00
-47
done
65
done
nafiz@nafiz-VirtualBox:~/Documents$
```

Figure 5.5: Automated test result for adjacent channel rejection

5.3 Spurious Rejection

In this test SMA100A and SMB100A are assigned the same frequency as that of the signal generator. The codes are same as before. Reference sensitivity, P_{REF} is calculated in the same way as done in the first automated test. After that P_{REF} is increased by 3 dB and the frequency of SMB100A is increased by 90 MHz. After this the RF output of SMB100A is turned on. The signal generator then adjusts its power level until the SINAD meter gives a reading between 11 dB and 13 dB. The level at which the SINAD gives the desired value is recorded as P_{HIGH}. After that the spurious response is calculated using the following code:

```
rejection_high = p_high - p_ref
print rejection
```

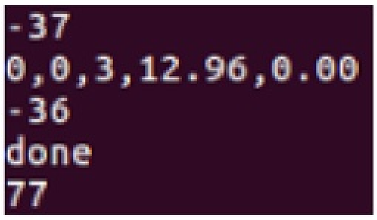
The short code for the above test is given below:

```
p_ref = signal_power
signal_power += 3 # increase wanted signal power by 3 db
```

```
interfere_power = p_ref + 75
my_instrument1.write("SOUR:POW "+str(signal_power)+" dBm")
my_instrument1.query("SOUR:POW?")
def get_interfere_power(audio_val, step, upper, lower, signal_power,
tn, my_instrument2, my_instrument3=None):
while audio_val<lower or audio_val>upper:
if audio_val<lower:
signal_power -= step
elif audio_val>upper:
signal_power += step
my_instrument2.write("SOUR:POW "+str(signal_power)+" dBm") changing
the signal power value
if my_instrument3:
my_instrument3.write("SOUR:POW "+str(signal_power)+" dBm") changing
the signal power value
time.sleep(1) to make sure audio value is updated before reading
tn.write("FETC:AF:ANAL:SIN?\n")
line = tn.read_until("\n",10)
print line
audio_val = float(line.split(',')[3])
print signal_power
print "done"
return signal_power
```

The full code for this test is given in the appendix section at the end of this document. After completing the manual test the spurious rejection was found to be 75 dB. The interfere power is made to be 75dB more than the P_{REF}. This is because the test will be faster as we know we are looking for a value around 75 dB.

SMB100A keeps on updating its power value until an output value between 11 dB and 13 dB is found from the SINAD meter. After the execution of the code the spurious rejection was found to be 77 dB. The final output that was received at the terminal is given below:



```
-37
0,0,3,12.96,0.00
-36
done
77
```

Figure 5.6: Automated test result for spurious rejection

5.4 Intermodulation Rejection

In this test all the three signal generators are used. Same frequency is assigned to all the generators. The reference sensitivity, P_{REF} is calculated in the same way as before. The RF output is kept off for SMB100A and SMBV100A. P_{REF} is then increased by 3 dB. The frequency of SMB100A is increased by 50 kHz and the frequency of SMBV100A is increased by 100 kHz. The RF output is now turned on for both. The next few lines of code does the same thing.

```
print my_instrument2.write("SOUR:FREQ 155.05 MHz")
print my_instrument2.write("OUTP ON")
print my_instrument3.write("SOUR:FREQ 155.1 MHz")
print my_instrument3.write("SOUR:FM:DEV 1.5 kHz")
print my_instrument3.write("SOUR:LFO:FREQ 400 Hz")
print my_instrument3.write("OUTP ON")
print my_instrument2.write("SOUR:FREQ 154.95 MHz")
print my_instrument2.write("OUTP ON")
print my_instrument3.write("SOUR:FREQ 154.9 MHz")
print my_instrument3.write("SOUR:FM:DEV 1.5 kHz")
print my_instrument3.write("SOUR:LFO:FREQ 400 Hz)
```

The signal generators are made equal by using the following code:

```
p_high = get_interfere_power(audio_val, step, upper, lower, interfere_power,
tn, my_instrument2, my_instrument3)
```

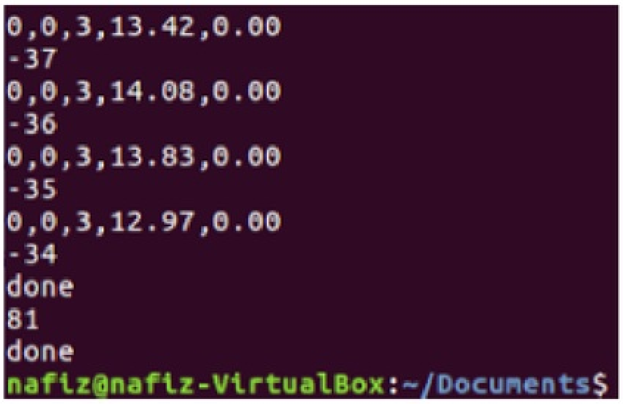
P_{LOW} is calculated in the same way as above after decreasing the frequency.

```
p_low = get_interfere_power(audio_val, step, upper, lower, interfere_power,
tn, my_instrument2, my_instrument3)
```

Intermodulation rejection is now calculated using the following code:

```
rejection_high = p_high - p_ref
rejection_low = p_low - p_ref
rejection = rejection_high if rejection_high < rejection_low else
rejection_low
rejection = p_high - p_ref
print rejection
print "done"
```

The complete code for this test is given in the appendix at the end of this document. After executing the complete code the intermodulation rejection was found to be 81 dB. The figure below shows the final output received in the terminal.



```
0,0,3,13.42,0.00
-37
0,0,3,14.08,0.00
-36
0,0,3,13.83,0.00
-35
0,0,3,12.97,0.00
-34
done
81
done
nafiz@nafiz-VirtualBox:~/Documents$
```

Figure 5.7: Automated test result for intermodulation rejection

5.5 Chapter summary

This chapter covered more detailed codes on how the tests were automated. The process was fairly simple as each line of instruction in the manual test procedures can be written in the form of SCPI commands. Including the previous chapter, the entire automation process is described starting from making different remote connections, SCPI commands to control instruments remotely and python scripts with compiled SCPI codes.

In the next and final chapter we will conclude by comparing the results between the manual and automated tests. An extension of this project for future work is also mentioned. At the end all the complete codes are provided in the Appendix and a bibliography is provided to acknowledge the research papers used in this document.

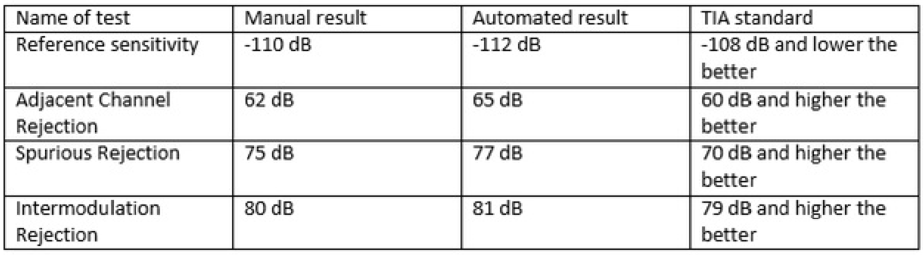
Chapter 6

Conclusion

This chapter gives a general overview about the work that has been done throughout the project. It also shows the comparison of test results between manual and automated processes. Furthermore it provides a section that discusses how this thesis work can be used for future work on extensions of this project.

6.1 Test results comparison

In chapter 3 we followed the TIA standards to complete the measurements for four tests manually. The results were recorded for future reference. In chapter 5 we completed the automated tests and the results were recorded as well. The table below is used to show a comparison between these two methods and TIA recommendations



Name of test	Manual result	Automated result	TIA standard
Reference sensitivity	-110 dB	-112 dB	-108 dB and lower the better
Adjacent Channel Rejection	62 dB	65 dB	60 dB and higher the better
Spurious Rejection	75 dB	77 dB	70 dB and higher the better
Intermodulation Rejection	80 dB	81 dB	79 dB and higher the better

Figure 6.1: Comparison of manual and automated test result

Based on the TIA standards and according to the figure above it can be said the automated tests produced a slightly better result than the manual tests.

6.2 Conclusion

The aim of this project was to develop an automated test system to monitor the RF characterisation of radios. In order to accomplish that instrument automation was necessary. We looked into two different methods on how to establish a remote connection. Detailed explanation about VISA libraries and different python module was also discussed. Moreover how to install Linux virtually in a windows machine was also mentioned.

Each chapter gave detailed explanation regarding all the requirements mentioned above. At the end the results were compared to give a clear picture why instrument automation was necessary. It can be said that automated tests produced a better and more reliable result compared to the manual tests. Therefore, we can say that the objective of this project was met. However, many things can be implemented and improved regarding this project and these are outlined in the next section.

6.3 Future work

According to the TIA standard both the receiver module and the transmitter module need to undergo rigorous testing in order to be certified for usage. This project discussed how the tests for receiver module can be automated. In an extension to this project, someone can work on how to automate the tests for the transmitter module. This research paper can be served as a proper guideline to accomplish that task.

The IP address of the signal generators are found after it is connected to a DHCP server switch using an Ethernet cable. However the IP address will change if the cable is moved from one port to the other. To solve this issue a similar python script such as nfcli.py can be made which will provide the IP address just by running the script. As a result the user wont have to manually identify the IP address of the signal generator using the display.

Another thing that can be changed is the way we assign frequency. Instead of assigning the same frequency to each generator individually, it can be stored as a functions variable. Therefore whenever we want to increase or decrease it, the function can be called and the variable can be updated. This will result in a shorter and faster script and a faster test process.

6.4 Chapter summary

In this chapter we summarized each stage of the project and how we met the project goal. The test results were compared and it was found that automated tests produced more reliable answers. In addition to that future extensions of this project and areas that can be improved are also mentioned.

Chapter 7

Abbreviations

RF	Radio Frequency
UHF	Ultra High Frequency
VHF	Very High Frequency
TIA	Telecommunication Industry Association
SCPI	Standard Commands for Programmable Instrument
VISA	Virtual Instrument Software Architecture
SINAD	Signal to Noise and Distortion
HiSLIP	High Speed Lan Interface Protocol
TCP	Transmission Control Protocol
IP	Internet Protocol
DHCP	Dynamic Host Control Protocol
GPIB	General Purpose Interface Bus
IEEE	Institute of Electrical and Electronics Engineering
FM	Frequency modulation
AM	Amplitude modulation

Bibliography

[1] “Aeroflex.” [Online]. Available: http://ats.aeroflex.com/radio-test-sets/land-mobile-radio-products/3920b-series-analog-and-digital-radio-test-platform

[2] “Dynamic Host Control Protocol.” [Online]. Available: http://whatismyipaddress.com/dhcp

[3] “High Speed LAN Interface Protocol.” [Online]. Available: https://pyvisa.readthedocs.io/en/stable/

[4] “Process control.” [Online]. Available: https://en.wikipedia.org/wiki/Process_control

[5] “RF Technology ; Base station.” [Online]. Available: http://www.rftechnology.com.au/rf_fsprod.html

[6] “RF Technology ; Introduction.” [Online]. Available: http://www.rftechnology.com.au/

[7] “Rohde and Schwarz.” [Online]. Available: https://cdn.rohde-schwarz.com/pws/dl_downloads/dl_common_library/dl_manuals/gb_1/s/sma/SMA100A_OperatingManual_en_14.pdf

[8] “Standard Commands for Programmable Instrument.” [Online]. Available: http://www.radio-electronics.com/info/t_and_m/scpi-standard-commands-for-programmable-instrumentation/basics-tutorial.php

[9] “Telecommunication Industry Association.” [Online]. Available: https://en.wikipedia.org/wiki/TIA/EIA-568

[10] “Telnet.” [Online]. Available: https://kb.iu.edu/d/aayd

[11] “Virtual Instrument Software Architecture.” [Online]. Available: http://www.tek.com/support/faqs/what-visa

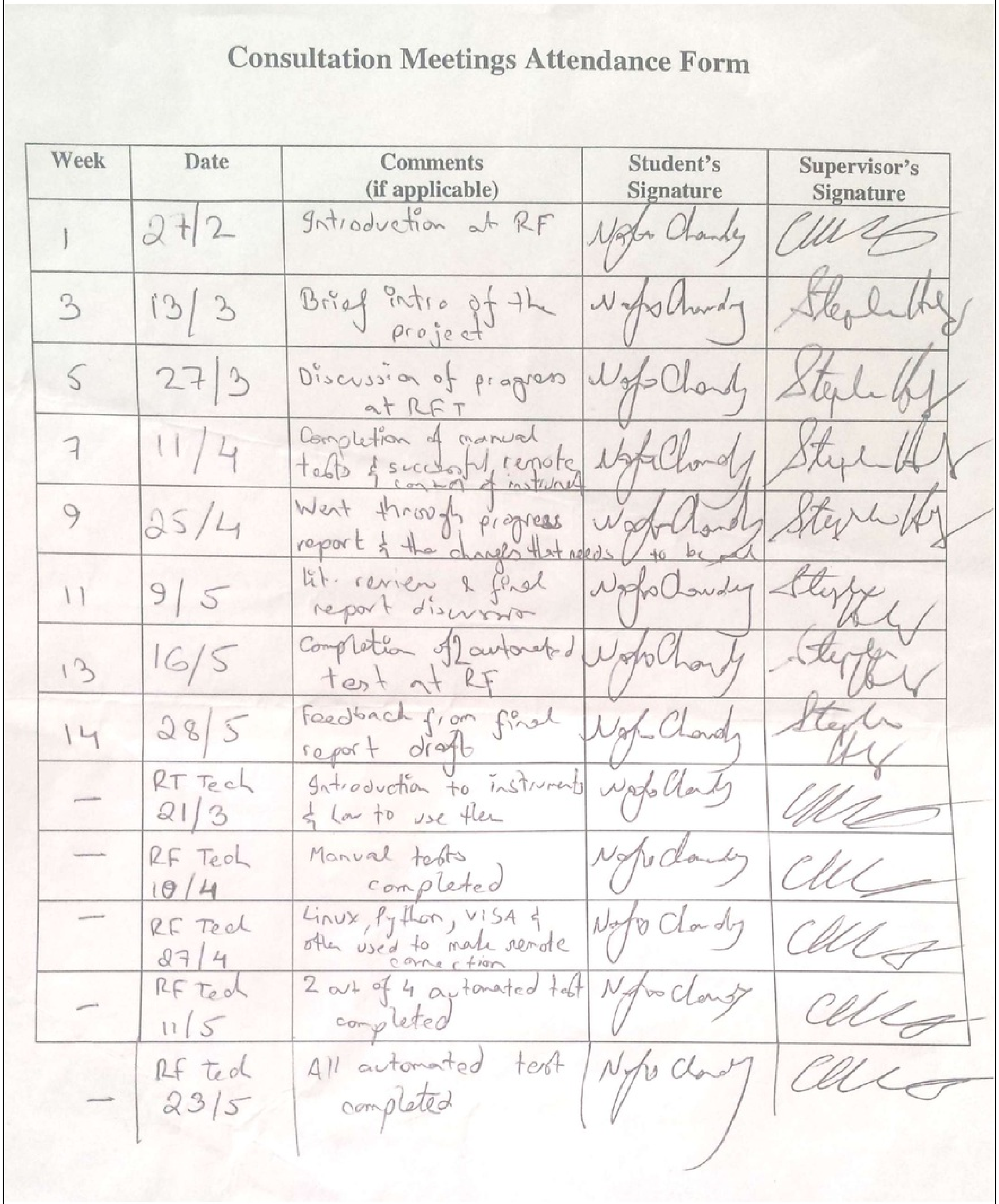
[12] “What is GPIB.” [Online]. Available: https://www.electronics-notes.com/articles/test-methods/gpib-ieee-488-bus/what-is-gpib-ieee488.php

[13] “What is numpy.” [Online]. Available: www.numpy.org

[14] “What is PyVISA.” [Online]. Available: https://pyvisa.readthedocs.io/en/stable/

Appendix A

Consultation Meeting Form



Consultation Meetings Attendance Form

Week	Date	Comments (if applicable)	Student's Signature	Supervisor's Signature
1	27/2	Introduction at RF	[illegible]	[illegible]
3	13/3	Brief intro of the project	[illegible]	[illegible]
5	27/3	Discussion of progress at RFT	[illegible]	[illegible]
7	11/4	Completion of manual tests & successful remote control of instruments	[illegible]	[illegible]
9	25/4	Went through progress report & the changes that needs to be [illegible]	[illegible]	[illegible]
11	9/5	lit. review & final report discussion	[illegible]	[illegible]
13	16/5	Completion of 2 automated test at RF	[illegible]	[illegible]
14	28/5	Feedback from final report draft	[illegible]	[illegible]
—	RT Tech 21/3	Introduction to instruments & how to use them	[illegible]	[illegible]
—	RF Tech 10/4	Manual tests completed	[illegible]	[illegible]
—	RF Tech 27/4	Linux, Python, VISA & other used to make remote connection	[illegible]	[illegible]
—	RF Tech 11/5	2 out of 4 automated test completed	[illegible]	[illegible]
—	RF Tech 23/5	All automated test completed	[illegible]	[illegible]

Scanned by CamScanner

Appendix B

Codes

1. Complete code that automates Reference Sensitivity

```
#!/usr/bin/python

import time
import numpy as np
import telnetlib
import visa

rm = visa.ResourceManager('@py')
my_instrument1 = rm.open_resource(u'TCPIP2::192.168.1.18::INSTR')
idn = my_instrument1.query('*IDN?')
print(idn)
my_instrument1.write("*rst; status:preset; *cls")
print my_instrument1.query("*OPC?") # Block until operation is
complete

print my_instrument1.write("SYST:SERR?")
print my_instrument1.query("*ESR?") # status register
print my_instrument1.write("SOUR:FREQ 155 MHz")
print my_instrument1.query("SOUR:FREQ?")
print my_instrument1.write("SOUR:POW -47 dBm")
print my_instrument1.query("SOUR:POW?")
print my_instrument1.write("OUTP ON")
print my_instrument1.query("OUTP?")
print my_instrument1.write("FM:STAT ON")

tn = telnetlib.Telnet("192.168.1.66", "1234")
tn.write("++mode 1\n")
tn.write("++auto 1\n")
tn.write("++addr 3\n")
tn.write("*IDN?\n")
tn.write("CONF:AF:ANAL:SIN:AVER 3\n")
tn.write("CONF:AF:ANAL:SIN:AVER?\n")
tn.write("FETC:AF:ANAL:SIN?\n")


line = tn.read_until("\n",30)
audio_val = float(line.split(',')[3])
lower = 11
upper = 13
signal_power = -117
step = 1

def get_signal_power(audio_val, step, upper, lower, signal_power,
tn, my_instrument1):
    while audio_val<lower or audio_val>upper:
        if audio_val<lower:
            signal_power += step
        elif audio_val>upper:
            signal_power -= step
        my_instrument1.write("SOUR:POW "+str(signal_power)+" dBm")
#changing the signal power value
        time.sleep(1) #to make sure audio value is updated before
reading
        tn.write("FETC:AF:ANAL:SIN?\n")
        line = tn.read_until("\n",30)
        audio_val = float(line.split(',')[3])
        print signal_power
        print "done"
```

2. Complete code for adjacent channel rejection

```
#!/usr/bin/python

import time
import numpy as np
import telnetlib
import visa

rm = visa.ResourceManager('@py')
my_instrument1 = rm.open_resource(u'TCPIP2::192.168.1.18::INSTR')
idn = my_instrument1.query('*IDN?')
print(idn)
my_instrument1.write("*rst; status:preset; *cls")
print my_instrument1.query("*OPC?") # Block until operation is
complete
print my_instrument1.write("SYST:SERR?")
print my_instrument1.query("*ESR?") # status register
print my_instrument1.write("SOUR:FREQ 155 MHz")
print my_instrument1.query("SOUR:FREQ?")
print my_instrument1.write("SOUR:POW -47 dBm")
print my_instrument1.query("SOUR:POW?")
print my_instrument1.write("OUTP ON")
print my_instrument1.query("OUTP?")
print my_instrument1.write("FM:STAT ON")

my_instrument2 = rm.open_resource(u'TCPIP2::192.168.1.64::INSTR')
idn = my_instrument2.query('*IDN?')
print(idn)
my_instrument2.write("*rst; status:preset; *cls")
print my_instrument2.query("*OPC?") # Block until operation is
complete
print my_instrument2.write("SYST:SERR?")
print my_instrument2.query("*ESR?") # status register
print my_instrument2.write("SOUR:FREQ 155 MHz")
print my_instrument2.query("SOUR:FREQ?")
print my_instrument2.write("SOUR:POW -47 dBm")
print my_instrument2.query("SOUR:POW?")
print my_instrument2.write("SOUR:FM:DEV 1.5 kHz")
print my_instrument2.query("SOUR:FM:DEV?")
print my_instrument2.write("SOUR:LFO:FREQ 400Hz")
#print my_instrument2.write("OUTP OFF")
#print my_instrument2.query("OUTP?")
print my_instrument2.write("FM:STAT ON")
print my_instrument2.write("POW:ALC:STAT ON")
print my_instrument2.query("POW:ALC:STAT?")


tn = telnetlib.Telnet("192.168.1.66", "1234")
tn.write("++mode 1\n")
tn.write("++auto 1\n")
tn.write("++addr 3\n")
tn.write("*IDN?\n")
tn.write("CONF:AF:ANAL:SIN:AVER 3\n")
tn.write("CONF:AF:ANAL:SIN:AVER?\n")
tn.write("FETC:AF:ANAL:SIN?\n")


line = tn.read_until("\n",10)
print line
audio_val = float(line.split(',')[3])
lower = 11
upper = 13
signal_power = -117
step = 1

def get_signal_power(audio_val, step, upper, lower, signal_power,
tn, my_instrument1):
    while audio_val<lower or audio_val>upper:
        if audio_val<lower:
            signal_power += step
        elif audio_val>upper:
            signal_power -= step
        my_instrument1.write("SOUR:POW "+str(signal_power)+" dBm")
#changing the signal power value
        time.sleep(1) #to make sure audio value is updated before
reading
        tn.write("FETC:AF:ANAL:SIN?\n")
        line = tn.read_until("\n",10)
        print line
        audio_val = float(line.split(',')[3])
```

```
         print signal_power
        print "done"
        return signal_power

# #starting adjacent channel


signal_power = get_signal_power(audio_val, step, upper, lower,
signal_power, tn, my_instrument1)



p_ref = signal_power
signal_power += 3 # increase wanted signal power by 3 db
interfere_power = p_ref + 75
my_instrument1.write("SOUR:POW "+str(signal_power)+" dBm")
my_instrument1.query("SOUR:POW?")

def get_interfere_power(audio_val, step, upper, lower, signal_power,
tn, my_instrument2, my_instrument3=None):
    while audio_val<lower or audio_val>upper:
        if audio_val<lower:
            signal_power -= step
        elif audio_val>upper:
            signal_power += step
        my_instrument2.write("SOUR:POW "+str(signal_power)+" dBm")
#changing the signal power value
        if my_instrument3:
            my_instrument3.write("SOUR:POW "+str(signal_power)+"
dBm") #changing the signal power value
        time.sleep(1) #to make sure audio value is updated before
reading
        tn.write("FETC:AF:ANAL:SIN?\n")
        line = tn.read_until("\n",20)
        print line
        audio_val = float(line.split(',')[3])
        print signal_power
        print "done"
    return signal_power

#measuring p_high
print my_instrument2.write("SOUR:FREQ 155.0125 MHz")
print my_instrument2.query("SOUR:FREQ?")
print my_instrument2.write("OUTP ON") #turning on RF output
print my_instrument2.query("OUTP?")
p_high = get_interfere_power(audio_val, step, upper, lower,
interfere_power, tn, my_instrument2)
```

```
#measuring p_low
print my_instrument2.write("SOUR:FREQ 154.9875 MHz")
print my_instrument2.query("SOUR:FREQ?")
p_low = get_interfere_power(audio_val, step, upper, lower,
interfere_power, tn, my_instrument2)


#measuring rejection


rejection_high = p_high - p_ref
rejection_low =  p_low - p_ref

rejection = rejection_high if rejection_high < rejection_low else
rejection_low

print rejection
print "done"
```

3. Complete code for Spurious Rejection

```
#!/usr/bin/python

import time
import numpy as np
import telnetlib
import visa

rm = visa.ResourceManager('@py')
my_instrument1 = rm.open_resource(u'TCPIP2::192.168.1.18::INSTR')
idn = my_instrument1.query('*IDN?')
print(idn)
my_instrument1.write("*rst; status:preset; *cls")
print my_instrument1.query("*OPC?") # Block until operation is
complete

print my_instrument1.write("SYST:SERR?")
print my_instrument1.query("*ESR?") # status register
print my_instrument1.write("SOUR:FREQ 155 MHz")
print my_instrument1.query("SOUR:FREQ?")
print my_instrument1.write("SOUR:POW -47 dBm")
print my_instrument1.query("SOUR:POW?")
print my_instrument1.write("OUTP ON")
print my_instrument1.query("OUTP?")
print my_instrument1.write("FM:STAT ON")
print my_instrument1.write("POW:ALC:STAT ON")
print my_instrument1.query("POW:ALC:STAT?")
```

```
my_instrument2 = rm.open_resource(u'TCPIP2::192.168.1.64::INSTR')
idn = my_instrument2.query('*IDN?')
print(idn)
my_instrument2.write("*rst; status:preset; *cls")
print my_instrument2.query("*OPC?") # Block until operation is
complete
print my_instrument2.write("SYST:SERR?")
print my_instrument2.query("*ESR?") # status register
print my_instrument2.write("SOUR:FREQ 155 MHz")
print my_instrument2.query("SOUR:FREQ?")
print my_instrument2.write("SOUR:POW -47 dBm")
print my_instrument2.query("SOUR:POW?")
print my_instrument2.write("SOUR:FM:DEV 1.5 kHz")
print my_instrument2.query("SOUR:FM:DEV?")
print my_instrument2.write("SOUR:LFO:FREQ 400Hz")
#print my_instrument2.write("OUTP OFF")
#print my_instrument2.query("OUTP?")
print my_instrument2.write("FM:STAT ON")
print my_instrument2.write("POW:ALC:STAT ON")
print my_instrument2.query("POW:ALC:STAT?")

tn = telnetlib.Telnet("192.168.1.66", "1234")
tn.write("++mode 1\n")
tn.write("++auto 1\n")
tn.write("++addr 3\n")
tn.write("*IDN?\n")
tn.write("CONF:AF:ANAL:SIN:AVER 3\n")
tn.write("CONF:AF:ANAL:SIN:AVER?\n")
tn.write("FETC:AF:ANAL:SIN?\n")


line = tn.read_until("\n",30)
audio_val = float(line.split(',')[3])
lower = 11
upper = 13
signal_power = -117
step = 1

def get_signal_power(audio_val, step, upper, lower, signal_power,
tn, my_instrument1):
    while audio_val<lower or audio_val>upper:
        if audio_val<lower:
            signal_power += step
        elif audio_val>upper:
            signal_power -= step
        my_instrument1.write("SOUR:POW "+str(signal_power)+" dBm")
#changing the signal power value
        time.sleep(1) #to make sure audio value is updated before
reading
```

```
        tn.write("FETC:AF:ANAL:SIN?\n")
        line = tn.read_until("\n",30)
        print line
        audio_val = float(line.split(',')[3])
        print signal_power
        print "done"
    return signal_power

#starting spurious rejection

signal_power = get_signal_power(audio_val, step, upper, lower,
signal_power, tn, my_instrument1)

p_ref = signal_power
signal_power += 3 # increase wanted signal power by 3 db
interfere_power = p_ref + 75
my_instrument1.write("SOUR:POW "+str(signal_power)+" dBm")
my_instrument1.query("SOUR:POW?")

def get_interfere_power(audio_val, step, upper, lower, signal_power,
tn, my_instrument2, my_instrument3=None):
    while audio_val<lower or audio_val>upper:
        if audio_val<lower:
            signal_power -= step
        elif audio_val>upper:
            signal_power += step
        my_instrument2.write("SOUR:POW "+str(signal_power)+" dBm")
#changing the signal power value
        if my_instrument3:
            my_instrument3.write("SOUR:POW "+str(signal_power)+"
dBm") #changing the signal power value
        time.sleep(1) #to make sure audio value is updated before
reading
        tn.write("FETC:AF:ANAL:SIN?\n")
        line = tn.read_until("\n",10)
        print line
        audio_val = float(line.split(',')[3])
        print signal_power
        print "done"
    return signal_power


##for p_high
print my_instrument2.write("SOUR:FREQ 244.99 MHz")
print my_instrument2.query("SOUR:FREQ?")
print my_instrument2.write("OUTP ON")
print my_instrument2.query("OUTP?")
p_high = get_interfere_power(audio_val, step, upper, lower,
interfere_power, tn, my_instrument2)
```

```
rejection = p_high - p_ref
print rejection
print "done"
```

4. Complete code for intermodulation rejection

```
#!/usr/bin/python

import time
import numpy as np
import telnetlib
import visa

rm = visa.ResourceManager('@py')
my_instrument1 = rm.open_resource(u'TCPIP2::192.168.1.18::INSTR')
idn = my_instrument1.query('*IDN?')
print(idn)
my_instrument1.write("*rst; status:preset; *cls")
print my_instrument1.query("*OPC?") # Block until operation is
complete

print my_instrument1.write("SYST:SERR?")
print my_instrument1.query("*ESR?") # status register
print my_instrument1.write("SOUR:FREQ 155 MHz")
print my_instrument1.query("SOUR:FREQ?")
print my_instrument1.write("SOUR:POW -47 dBm")
print my_instrument1.query("SOUR:POW?")
print my_instrument1.write("OUTP ON")
print my_instrument1.query("OUTP?")
print my_instrument1.write("FM:STAT ON")


my_instrument2 = rm.open_resource(u'TCPIP2::192.168.1.64::INSTR')
idn = my_instrument2.query('*IDN?')
print(idn)
my_instrument2.write("*rst; status:preset; *cls")
print my_instrument2.query("*OPC?") # Block until operation is
complete
print my_instrument2.write("SYST:SERR?")
print my_instrument2.query("*ESR?") # status register
print my_instrument2.write("SOUR:FREQ 155 MHz")
print my_instrument2.query("SOUR:FREQ?")
print my_instrument2.write("SOUR:POW -47 dBm")
print my_instrument2.query("SOUR:POW?")
print my_instrument2.write("OUTP OFF")
print my_instrument2.query("OUTP?")
print my_instrument2.write("FM:STAT ON")
print my_instrument2.write("POW:ALC:STAT ON")
print my_instrument2.query("POW:ALC:STAT?")


my_instrument3 = rm.open_resource(u'TCPIP2::192.168.1.43::INSTR')
idn = my_instrument2.query('*IDN?')
print(idn)
my_instrument3.write("*rst; status:preset; *cls")
print my_instrument3.query("*OPC?") # Block until operation is
complete
print my_instrument3.write("SYST:SERR?")
print my_instrument3.query("*ESR?") # status register
print my_instrument3.write("SOUR:FREQ 155 MHz")
print my_instrument3.query("SOUR:FREQ?")
print my_instrument3.write("SOUR:POW -47 dBm")
print my_instrument3.query("SOUR:POW?")
print my_instrument3.write("SOUR:FM:DEV 1.5 kHz")
print my_instrument3.query("SOUR:FM:DEV?")
print my_instrument3.write("SOUR:LFO:FREQ 400Hz")
print my_instrument3.write("OUTP OFF")
print my_instrument3.query("OUTP?")
print my_instrument3.write("FM:STAT ON")
print my_instrument3.write("POW:ALC:STAT ON")
print my_instrument3.query("POW:ALC:STAT?")



tn = telnetlib.Telnet("192.168.1.66", "1234")
tn.write("++mode 1\n")
tn.write("++auto 1\n")
tn.write("++addr 3\n")
tn.write("*IDN?\n")
tn.write("CONF:AF:ANAL:SIN:AVER 3\n")
tn.write("CONF:AF:ANAL:SIN:AVER?\n")
tn.write("FETC:AF:ANAL:SIN?\n")


line = tn.read_until("\n",30)
audio_val = float(line.split(',')[3])
lower = 11
upper = 13
signal_power = -117
step = 1
```

```
def get_signal_power(audio_val, step, upper, lower, signal_power,
tn, my_instrument1):
    while audio_val<lower or audio_val>upper:
        if audio_val<lower:
            signal_power += step
        elif audio_val>upper:
            signal_power -= step
        my_instrument1.write("SOUR:POW "+str(signal_power)+" dBm")
#changing the signal power value
        time.sleep(1) #to make sure audio value is updated before
reading
        tn.write("FETC:AF:ANAL:SIN?\n")
        line = tn.read_until("\n",30)
        print line
        audio_val = float(line.split(',')[3])
        print signal_power
        print "done"
    return signal_power

#starting intermodulation

signal_power = get_signal_power(audio_val, step, upper, lower,
signal_power, tn, my_instrument1)
p_ref = signal_power
signal_power += 3 #increasing p_ref by 3 dB.
interfere_power = p_ref + 75
my_instrument1.write("SOUR:POW "+str(signal_power)+" dBm")

def get_interfere_power(audio_val, step, upper, lower, signal_power,
tn, my_instrument2, my_instrument3=None):
    while audio_val<lower or audio_val>upper:
        if audio_val<lower:
            signal_power -= step
        elif audio_val>upper:
            signal_power += step
        my_instrument2.write("SOUR:POW "+str(signal_power)+" dBm")
#changing the signal power value


if my_instrument3:
my_instrument3.write("SOUR:POW "+str(signal_power)+" dBm")


        time.sleep(1) #to make sure audio value is updated before
reading
        tn.write("FETC:AF:ANAL:SIN?\n")
        line = tn.read_until("\n",30)
        print line
        audio_val = float(line.split(',')[3])
        print signal_power
        print "done"
    return signal_power


## for p_high

print my_instrument2.write("SOUR:FREQ 155.05 MHz")—increasing
frequency
print my_instrument2.query("SOUR:FREQ?")
print my_instrument2.write("OUTP ON")
print my_instrument2.query("OUTP?")
print my_instrument3.write("SOUR:FREQ 155.1 MHz")
print my_instrument3.query("SOUR:FREQ?")
print my_instrument3.write("SOUR:FM:DEV 1.5 kHz")—modulation at 60%
print my_instrument3.query("SOUR:FM:DEV?")
print my_instrument3.write("SOUR:LFO:FREQ 400Hz")
print my_instrument3.write("OUTP ON")
print my_instrument3.query("OUTP?")

p_high = get_interfere_power(audio_val, step, upper, lower,
interfere_power, tn, my_instrument2, my_instrument3)

##for p_low
print my_instrument2.write("SOUR:FREQ 154.95 MHz")—decreasing
frequency
print my_instrument2.query("SOUR:FREQ?")
print my_instrument2.write("OUTP ON")
print my_instrument2.query("OUTP?")
print my_instrument3.write("SOUR:FREQ 154.9 MHz")
print my_instrument3.query("SOUR:FREQ?")
print my_instrument3.write("SOUR:FM:DEV 1.5 kHz")
print my_instrument3.query("SOUR:FM:DEV?")
print my_instrument3.write("SOUR:LFO:FREQ 400Hz")
print my_instrument3.write("OUTP ON")
print my_instrument3.query("OUTP?")

p_low = get_interfere_power(audio_val, step, upper, lower,
interfere_power, tn, my_instrument2, my_instrument3)

rejection_high = p_high - p_ref
rejection_low =  p_low - p_ref

rejection = rejection_high if rejection_high < rejection_low else
rejection_low
rejection = p_high - p_ref
print rejection
print "done"
```

5. Complete code for nfcli.py

```
import getopt
import socket
import sys
import os
import platform
from nfutil import *
from enumip import *
#-------------------------------------------------------------------
----------
def usage():
    print
    print "Usage: nfcli --list --eth_addr=ADDR --ip_type=TYPE --
ip_addr=IP, --netmask=MASK, --gateway=GW"
    print "Search and configure Prologix GPIB-ETHERNET
Controllers."
    print "--help          : display this help"
    print "--list          : search for controllers"
    print "--eth_addr=ADDR : configure controller with Ethernet
address ADDR"
    print "--ip_type=TYPE  : set controller ip address type to TYPE
(\"static\" or \"dhcp\")"
    print "--ip_addr=IP    : set controller address to IP"
    print "--netmask=MASK  : set controller network mask to MASK"
    print "--gateway=GW    : set controller default gateway to GW"


#-------------------------------------------------------------------
----------
def enumIp():
    if platform.system() in ('Windows', 'Microsoft'):
        return socket.gethostbyname_ex(socket.gethostname())[2];

    return enumIpUnix()


#-------------------------------------------------------------------
----------
def ValidateNetParams(ip_str, mask_str, gw_str):

    try:
        ip = socket.inet_aton(ip_str)
    except:
        print "IP address is invalid."
        return False

    try:
        mask = socket.inet_aton(mask_str)
    except:
        print "Network mask is invalid."
        return False

    try:
        gw = socket.inet_aton(gw_str)
    except:
        print "Gateway address is invalid."
        return False

    # Validate network mask

    # Convert to integer from byte array
    mask = struct.unpack("!L", mask)[0]

    # Exclude restricted masks
    if (mask == 0) or (mask == 0xFFFFFFFF):
        print "Network mask is invalid."
        return False

    # Exclude non-left-contiguous masks
    if (((mask + (mask & -mask)) & 0xFFFFFFFF) != 0):
        print "Network mask is not contiguous."
        return False


    # Validate gateway address

    octet1 = ord(gw[0])

    # Convert to integer from byte array
    gw = struct.unpack("!L", gw)[0]

    # Exclude restricted addresses
    # 0.0.0.0 is valid
    if ((gw != 0) and ((octet1 == 0) or (octet1 == 127) or (octet1 >
223))):
        print "Gateway address is invalid."
        return False

    # Validate IP address

    octet1 = ord(ip[0])

    # Convert to integer from byte array
    ip = struct.unpack("!L", ip)[0]

    # Exclude restricted addresses
    if ((octet1 == 0) or (octet1 == 127) or (octet1 > 223)):
        print "IP address is invalid."
        return False

    # Exclude subnet network address
    if ((ip & ~mask) == 0):
        print "IP address is invalid."
        return False

    # Exclude subnet broadcast address
    if ((ip & ~mask) == (0xFFFFFFFF & ~mask)):
        print "IP address is invalid."
        return False

    return True

#--------------------------------------------------------------------
----------
#def ValidateAddress(address):

#    if address is None:
#        return False

#    parts = address.split(".")

#    if len(parts) != 4:
#        return False

#    try:
#        for item in parts:
#            if not 0 <= int(item) <= 255:
#                return False
#    except:
#        return False

#    return True


#--------------------------------------------------------------------
----------
def main():

    invalid_args = False
    showhelp = False
    search = False
    ip_type = None
    ip_addr = None
    netmask = None
    gateway = None
    eth_addr = None

    try:
        opts, args = getopt.getopt(sys.argv[1:], '', ['help',
'list', 'eth_addr=', 'ip_type=', 'ip_addr=', 'netmask=',
'gateway='])
    except getopt.GetoptError, err:
        print str(err)
        sys.exit(1)

    # Check for unparsed parameters
    if len(args) != 0:
        usage()
        sys.exit(1)

    for o, a in opts:
        if o == "--help":
            showhelp = True
        elif o == "--list":
            search = True
        elif o == "--eth_addr":
            eth_addr = a
        elif o == "--ip_type":
            ip_type = a
        elif o == "--ip_addr":
            ip_addr = a
        elif o == "--netmask":
            netmask = a
        elif o == "--gateway":
            gateway = a

    if (len(opts) == 0) or (showhelp):
        usage()
        sys.exit(1)

    if search:
        if not eth_addr is None:
            print "--list and --eth_addr are not compatible."
            invalid_args = True
        if not ip_type is None:
            print "--list and --ip_type are not compatible."
            invalid_args = True
        if not ip_addr is None:
            print "--list and --ip_addr are not compatible."
            invalid_args = True
        if not netmask is None:
            print "--list and --netmask are not compatible."
            invalid_args = True
        if not gateway is None:
            print "--list and --gateway are not compatible."
            invalid_args = True
    else:

        try:
            eth_addr = eth_addr.strip().replace(":", "").replace("-
", "")
            eth_addr = eth_addr.decode('hex')
        except:
            print "Invalid Ethernet address."
            sys.exit(1)

        if len(eth_addr) != 6:
            print "Invalid Ethernet address."
            sys.exit(1)

        if ip_type in ["Static", "static"]:
            ip_type = NF_IP_STATIC
        elif ip_type in ["Dynamic", "dynamic", "Dhcp", "dhcp"]:
            ip_type = NF_IP_DYNAMIC
        else:
            print "--ip_type must be 'static' or 'dhcp'."
            sys.exit(1)

        if ip_type == NF_IP_STATIC:

            if not ValidateNetParams(ip_addr, netmask, gateway):
                invalid_args = True


#            if not ValidateIP(ip_addr):
#                print "Invalid, or no, IP address specified."
#                invalid_args = True

#            if not ValidateIP(netmask):
#                print "Invalid, or no, netmask specified."
#                invalid_args = True

#            if not ValidateIP(gateway):
#                print "Invalid, or no, gateway address specified."
#                invalid_args = True
        else:
            if ip_addr is None:
                ip_addr = "0.0.0.0"
            else:
                print "--ip_addr not allowed when --ip_type=dhcp."
                invalid_args = True
            if netmask is None:
                netmask = "0.0.0.0"
            else:
                print "--netmask not allowed when --ip_type=dhcp."
                invalid_args = True
            if gateway is None:
                gateway = "0.0.0.0"
            else:
                print "--gateway not allowed when --ip_type=dhcp."
                invalid_args = True

    if invalid_args:
        sys.exit(1)



    global seq
    sys.stdout = os.fdopen(sys.stdout.fileno(), 'w', 0)

    iplist = enumIp();
    if len(iplist) == 0:
        print "Host has no IP address."
        sys.exit(1)


    devices = {}

    for ip in iplist:
        print "Searching through network interface:", ip

        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

        port = 0

        try:
            s.bind((ip, port))
        except socket.error, e:
            print "Bind error on send socket:", e
            sys.exit(1)

        port = s.getsockname()[1]

        r = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        r.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        r.setblocking(1)
        r.settimeout(0.100)

        try:
            r.bind(('', port))
        except socket.error, e:
            print "Bind error on receive socket:", e
            sys.exit(1)

        d = Discover(s, r)
        print

        for k in d:
            d[k]['host_ip'] = ip
            devices[k] = d[k]

        s.close()
        r.close()

    if search:
        print
        print "Found", len(devices), "Prologix GPIB-ETHERNET
Controller(s)."
        for key in devices:
            PrintDetails(devices[key])
            print
    else:
        if eth_addr in devices:

            print "Updating network settings of Prologix GPIB-
ETHERNET Controller", FormatEthAddr(eth_addr)

            device = devices[eth_addr]

            if (device['ip_type'] == NF_IP_STATIC) or (ip_type ==
NF_IP_STATIC):
                s = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
                s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
1)
                s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST,
1)

                port = 0

                try:
                    s.bind((device['host_ip'], port))
                except socket.error, e:
                    print "Bind error on send socket:", e
                    sys.exit(1)

                port = s.getsockname()[1]

                r = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
                r.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
1)

                r.setblocking(1)
                r.settimeout(0.100)

                try:
                    r.bind(('', port))
                except socket.error, e:
                    print "Bind error on receive socket:", e
                    sys.exit(1)

                result = Assignment(s, r, eth_addr, ip_type,
ip_addr, netmask, gateway)
                print

                if len(result) == 0:
                    print "Network settings update failed."
                else:
                    if result['result'] == NF_SUCCESS:
                        print "Network settings updated
successfully."
                    else:
                        print "Network settings update failed."
            else:
                print "Prologix GPIB-ETHERNET Controller",
FormatEthAddr(eth_addr), "already configured for DHCP."
        else:
            print "Prologix GPIB-ETHERNET Controller",
FormatEthAddr(eth_addr), "not found."



#---------------------------------------------------------------------
----------
if __name__ == "__main__":
    main()
```