

IMPLEMENTATION OF WEATHER SATELLITE RECEIVER USING SOFTWARE DEFINED RADIO

Submitted in partial fulfillment of the requirements for the award of Bachelor of
Engineering Degree in Electronics and Communication Engineering

By

PUVVADA JITHENDRA MANIKANTA (Register No.37130332)

L.YESHWANTH KUMAR REDDY (37130220)



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING
SCHOOL OF ELECTRICAL AND ELECTRONICS
ENGINEERING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI – 600119

APRIL 2021



SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with "A" grade by NAAC
Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600 119
www.sathyabama.ac.in



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING**

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **P. JITHENDRA MANIKANTA (Reg. No:37130332)** and **L.YESHWANTH KUMAR REDDY (37130220)** who carried out the project entitled **"IMPLEMENTATION OF WEATHER SATELLITE RECEIVER USING SOFTWARE DEFINED RADIO "** under my supervision from December 2020 to March 2021.

Internal Guide

Dr. M. SUGADEV, M.Tech., Ph.D

Head of the Department

Dr. T. RAVI M.E., PhD.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

We, **P. JITHENDRA MANIKANTA (Reg. No: 37130332)** and **L.YESHWANTH KUMAR REDDY (37130220)** hereby declare that the Project Report entitled **“IMPLEMENTATION OF WEATHER SATELLITE RECEIVER USING SOFTWARE DEFINED RADIO”** done by us under the guidance of **Dr.M.SUGADEV, M.Tech ,Ph.D**, Assistant professor, Dept. of Electronics and Communication Engineering, SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI, is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Electronics and Communication Engineering.

DATE :

SIGNATURE OF THE CANDIDATES

PLACE :

ACKNOWLEDGEMENT

We are pleased to acknowledge our sincere thanks to the Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey our thanks to **Dr. N. M. NANDHITHA, M.E., Ph.D. Dean, School of Electrical and Electronics Engineering** and **Dr. T. RAVI, M.E., Ph.D. Head of the Department, Department of Electronics and Communication Engineering** for providing us necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to our Project Guide **Dr.M.SUGADEV. M.Tech., Ph.D**, Assistant professor, Dept. of Electronics and Communication Engineering, for his valuable guidance, suggestions and constant encouragement for the successful completion of our project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Electronics and Communication Engineering who were helpful in many ways for the completion of the project.

ABSTRACT

A Software Defined Radio (SDR) is a flexible technology that enables the design of adaptive communications systems. A generic hardware design can be used to address different communication needs, such as changing frequencies, modulation schemes and data rates. Applied to small satellites, some of the implications are increased data throughput when down-linking or up-linking by varying communications parameters and making use of one hardware design and implementation for communicating for many missions, just by updating the software. Therefore, development time for small satellite communication systems can be reduced in the future. The SDR technology can support different kinds of applications, such as Earth observation and communication services. In this project, we are going to develop a weather station using a software defined radio (SDR), which can be able to receive real-time satellite weather forecast grey scale images. This system can be used in a disastrous situation where there is no access to internet or cellphone signals. Hack-RF SDR is utilized in this work to implement the satellite receiver. Custom designed antenna is connected to the SDR to pickup satellite signal. Raspberry Pi single-board computer is used to process the data received by SDR and reproduce the satellite weather images on a display device or conventional monitor.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	i
	LIST OF FIGURES	iv
1	INTRODUCTION	1
1.1	SOFTWARE DEFINED RADIO	1
1.2	RTL-SDR RECEIVER	3
1.3	NOOELEC RTL-SDR	5
1.4	SDR COMPUTER SYSTEM REQUIREMENTS	6
1.4.1	RTL-SDR softwares	7
1.4.2	RTL-SDR Hardware Support Package	7
1.5	AIM AND OBJECTIVES OF THE WORK	8
2	EXISTING SYSTEM	9
2.1	GENERAL SDR ARCHITECTURE	9
2.2	QUADRATURE SDR ARCHITECTURE	11
2.3	SDR HARDWARE COMPONENTS	12
2.3.1	Signal Processing in R820T tuner	13
2.4	NESDR SDR RECEIVERS	15
2.5	INTERFACING WITH THE SDR FROM MATLAB AND SIMULINK	17
2.6	CHALLENGES OF LOW COST DESKTOP SDR	18
2.7	PROBLEM STATEMENT	19
2.8	OBJECTIVE OF THE WORK	19
3	PROPOSED SYSTEM	21
3.1	SETTING UP THE HACKRF ONE DEVICE	21
3.1.1	What is HackRF One	21
3.1.2	Setting up the work environment	21
3.1.3	Installing GNU Radio companion	22

3.2	RECEIVING FM RADIO SIGNALS	24
3.2.1	Customization of Receiving FM Radio Signals	29
3.3	TRANSMITTING FM RADIO SIGNALS	32
3.4	RETRANSMITTING FM RADIO SIGNALS	33
3.5	LIVE BROADCASTING FM RADIO SIGNALS	34
3.6	SATELLITE RECEIVER USING HACKRF ONE	36
4	RESULTS AND DISCUSSION	37
4.1	WEATHER IMAGES OUTPUT	37
5	CONCLUSION AND FUTURE SCOPE	42
5.1	CONCLUSION	42
5.2	FUTURE SCOPE	42
	REFERENCES	43

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
1.1	Radio frequency spectrum and its applications.	4
1.2	RTL SDR receiver block diagram	5
1.3	Three types of NooElec RTL-SDR : (a) nano (b) RF shielded (c) mini and the antenna	6
1.4	Operational Frequency bands for Nooelec SDR.	7
2.1	General SDR Architecture	11
2.2	Evolution of SDR receiver architecture: a) the baseband digital radio (b) the IF digital radio (c) the zero-IF digital radio.	12
2.3	Internal architecture of R820 Tuner	14
2.4	NooElec NESDR RTL-SDR	18
2.5	Configuring the RTL-SDR Receiver Simulink block (top), and the RTL-SDR MATLAB System Object (bottom) with identical parameter	20
3.1	Block of SDR based Satellite receiver	31
4.1	Setup used for reception of satellite signals	36
4.3	Antenna mast for satellite signal reception	37
4.4	Test Satellite images received in the laptop screen	38
4.5	Images of clouds received in the laptop	38
4.6	Images of water vapor received from satellite	39
4.7	Images received under visible light condition	40
4.7	Images received under Infrared light setup	41

CHAPTER 1

INTRODUCTION

1.1 SOFTWARE DEFINED RADIO

Software Defined Radio (SDR) is a generic term which refers to radio systems in which almost all of the functionality associated with the Physical Layer (PHY) is implemented in software using Digital Signal Processing (DSP) algorithms. An ideal SDR receiver would have a very small hardware front-end; only an antenna and a high speed GHz sampler that was capable of capturing and digitizing a wide band of radio frequencies. Any demodulation, synchronization, decoding or decryption required to recover information contained within a received signal would be performed in software that is executed on a superfast, dedicated processing device.

A number of definitions can be found to describe Software Defined Radio, also known as Software Radio or SDR. The SDR Forum, working in collaboration with the Institute of Electrical and Electronic Engineers (IEEE) P1900.1 group, has worked to establish a definition of SDR that provides consistency and a clear overview of the technology and its associated benefits. Simply put Software Defined Radio is defined as¹ : "Radio in which some or all of the physical layer functions are software defined" A radio is any kind of device that wirelessly transmits or receives signals in the radio frequency (RF) part of the electromagnetic spectrum to facilitate the transfer of information. In today's world, radios exist in a multitude of items such as cell phones, computers, car door openers, vehicles, and televisions. Traditional hardware based radio devices limit cross-functionality and can only be modified through physical intervention. This results in higher production costs and minimal flexibility in supporting multiple waveform standards. By contrast, software defined radio technology provides an efficient and comparatively inexpensive solution to this problem, allowing multimode, multi-band and/or multi-functional wireless devices that can be enhanced using software upgrades.

It's scarcely twenty years since the first second generation (2G) digital mobile phones appeared on the market. By the mid-1990s texting was widely adopted, and further, GSM and GPRS based modems became available—albeit only providing slow connectivity no higher than a few kbps. This was the start of the wireless digital data revolution. By the end of the 1990s and into the 2000s, third generation (3G) mobile connections became available at speeds of a few 100kbps and Wi-Fi emerged as a means of connecting devices to wireless access points over distances below 20metres or so. From early Wi-Fi speeds of just a few Mbps, by 2005 we were running at 54Mbps in most implementations and soon to speeds of 300Mbps and higher when MIMO methods emerged in the early 2010s. Smartphones also just keep getting smarter.

Wi-Fi and Bluetooth are now considered standard wireless connectivity requirements on smartphones alongside 2G, 3G, and likely fourth generation (4G) LTE-Advanced connections. Homes and offices are now well served by superfast Wi-Fi, and there are mobile base stations throughout cities, towns and the countryside, all around the world. In many ways though, the 'wireless revolution' is still just beginning. With more Short Range Devices (SRDs) being produced, and the so-called Internet of Things (IOT) continuing to evolve, we should expect in the very near future that there will be more than one wireless phone, laptop or tablet per person; and perhaps up to 10 other devices, ranging from key fobs to sensors, GPS tracked objects, and so on. It will be wireless everything very soon.

Many smartphones and similar devices currently have up to around 8 different radios optimized for receiving various signals from different frequency bands, such as those for Wi-Fi (2.4GHz), LTE (Long Term Evolution, 800MHz), GSM (Global System for Mobile Communications, 900MHz), UMTS (Universal Mobile Telecommunications System, 2.1MHz) GPS (Global Positioning System, 1.5GHz). Figure 1.1 shows radio spectrum allocation for various applications.

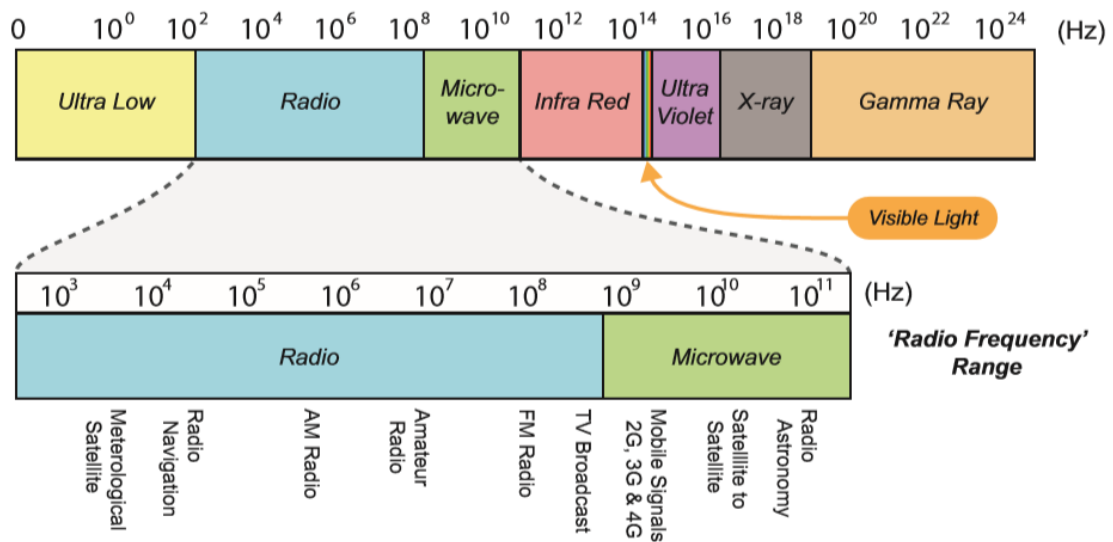


Fig: 1.1. Radio frequency spectrum and its applications.

The availability of devices that can sample up to a rate of say $f_s = 5\text{GHz}$, the sampling theorem dictates we can digitize all signals from baseband up to 2.5GHz. In practice, it's more likely that different bands would be filtered, and digitized at a lower sampling frequency, but in theory we could work with the entire band. With the availability of devices that can sample up to a rate of say $f_s = 5\text{GHz}$, the sampling theorem dictates we can digitize all signals from baseband up to 2.5GHz such as Bluetooth (2.4GHz), NFC (Near Field Communications, 13.56MHz), and FM Radio (100MHz). Soon they may even include radios to receive IOT and TV White Space UHF (Ultra High Frequency at 400MHz+) signals too. The ultimate solution here would be to utilize a single SDR that samples the spectrum at GHz rates to digitize and capture all signals from baseband to 2.5 or even 3GHz, and to implement all of these receivers in software code. Illustrates the approximate frequency band positions for many of the broadcast radio signals that most of us receive and use on a daily basis.

1.2 RTL-SDR RECEIVER

The RTL-SDR is a very low cost (sub-\$20), easy to USB device that receiver RF radio signals. Originally. These device were designed to be used as DVB-T (Digital Video Broadcast-Terrestrial) receiver, but it was discovered that they could be used as generic (receiver only) SDRs by simply putting them into a different mode. In this mode, they are capable of receiving any signal in the digital

television (DTV) signal they were designed to receiver. This range varies from device to device depending on what components have been used, but is most commonly from 25MHz to 1.75GHz. The front end of the RTL-SDR receives RF signal live off the air, down converts them to baseband, digitizes them, and the device outputs samples of the baseband signal across its USB interface.

MathWorks released a Hardware Support Package for the RTL-SDR in early 2014 which enables both MATLAB and Simulink to interface with, and control the RTL-SDR. With this add-on, samples output from the device can be captured and brought into the software, enabling users to implement any kind of DSP receiver or spectrum sensing system they desire in either a Simulink model or MATLAB code. As the entire demodulation process is carried out in software, this is a *Software Defined Radio* implementation.

The RTL-SDR set up of is represented in a block diagram form in figure 1.2. RF signals are received at the antenna, quadrature down converted by the RTL-SDR, and In Phase/ Quadrature Phase (IQ) samples are presented to the computer running MATLAB. The receiver design is implemented using the appropriate DSP algorithms to demodulate the signal to baseband and extract the information signal. This might be audio, video, images, or data.

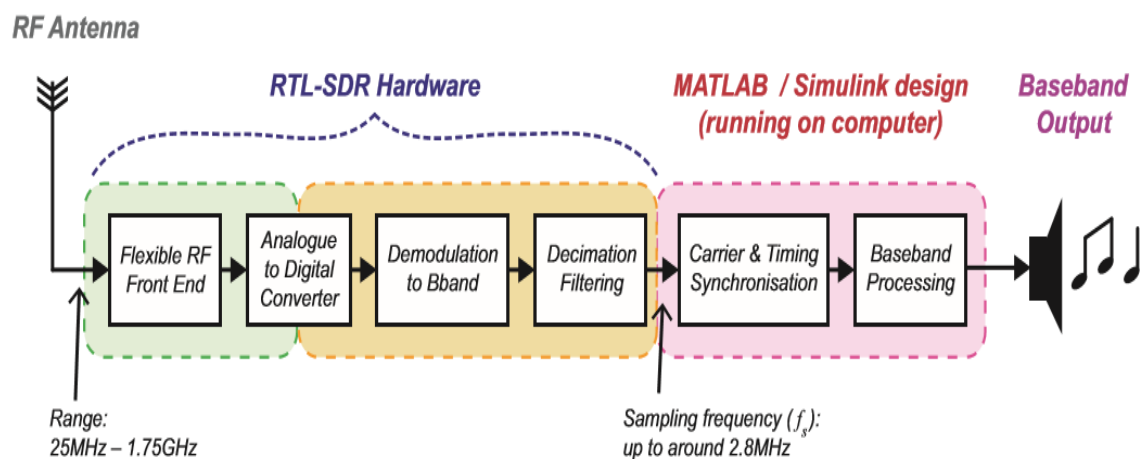


Fig:1.2: RTL SDR receiver block diagram

Subject to appropriate hardware configuration (i.e. using the right antenna!) and signals being broadcast in your vicinity, the RTL-SDR allows you to receive not only FM radio signals, but also UHF/ DTV signals, Digital Audio Broadcast (DAB) radio, GPS signals, 2G, 3G and 4G cellular signals, transmissions in the Industrial, Scientific and Medical (ISM) bands, etc. — in fact any signal that is transmitted within the operating range of the tuner. Shows some of the RF signals from the electromagnetic spectrum that can be received by the device.

RTL-SDR is a very cheap ~\$25 USB dongle that can be used as a computer based radio scanner for receiving live radio signals in your area (no internet required). Depending on the particular model it could receive frequencies from 500 kHz up to 1.75 GHz.

1.3 NOOELEC RTL - SDR

The **NooElec NESDR RTL-SDR receiver** cost around \$20 and there are a number of different form factors and sizes of these devices. All of these types, essentially have the same functionality and can be used in the same way. There are also some other RTL-SDRs that use different RF tuners notably the ElonicsE4000 rather than the more common Rafael Micro R820T.

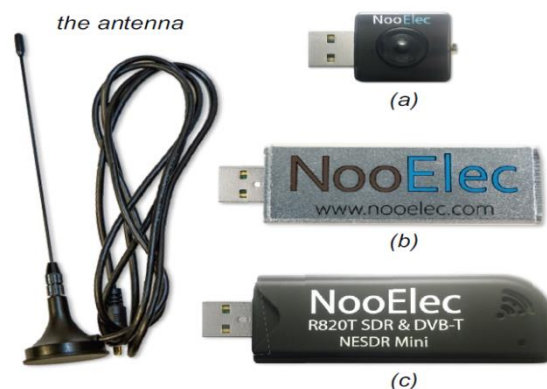


Fig:1.3: Three types of NooElec RTL-SDR : (a) nano (b) RF shielded (c) mini and the antenna

NooElec RTL-SDR devices feature a Micro Coaxial (MCX) antenna port, and ship with an omnidirectional MCX antenna. Although this antenna is tuned to

perform best in the UHF band, it will be suitable for receiving a range of signals for preliminary ‘out of the box’ spectrum viewing and baseband DSP processing in MATLAB and Simulink. Figure 1.3 shows the Nooelec RTL-SDR devices available in the market.

The Nooelec RTL-SDR can be configured to receive signals in the frequency bands illustrated in figure 1.4. It can work from the HF band to 1.5 GHz microwave band

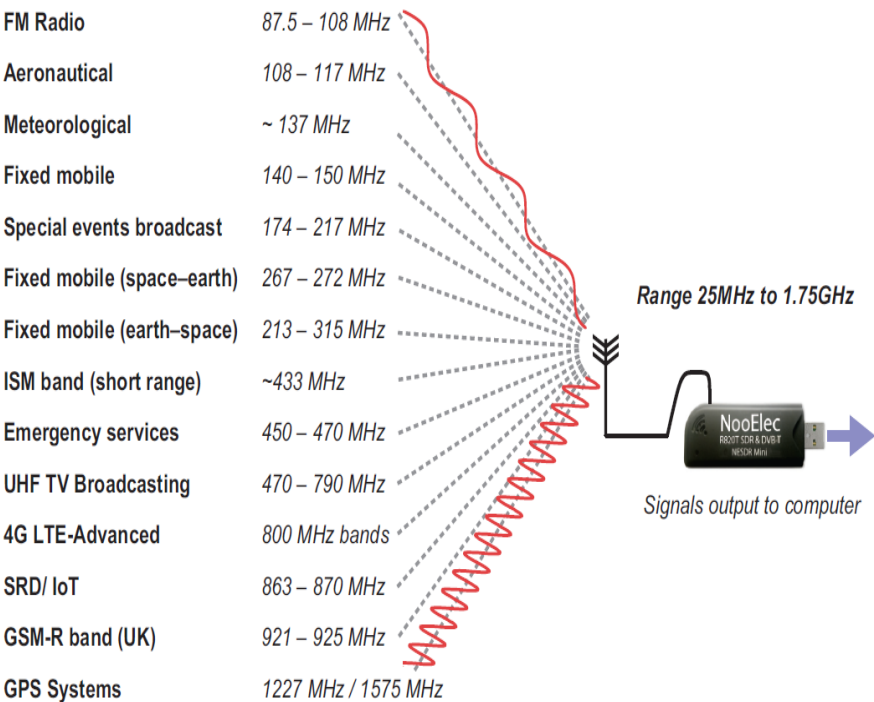


Fig: 1.4: Operational Frequency bands for Nooelec SDR.

1.4 SDR COMPUTER SYSTEM REQUIREMENTS

It is not appropriate to list detailed system requirements but it is recommended to have a modern computer with a ‘good’ processor such as a recent Intel i5, i7, AMD FX-8k series or similar core, at least 8GB of RAM, and 30GB of free hard drive space. You will also need at least one free USB2.0 (or higher) port, and we recommend a computer with a soundcard so that you can listen to any audio signals you demodulate. As long as your computer is fast enough, then for the most part, the SDR implementations with your RTL-SDR

should be able to run in real-time (i.e. the processor will be able to process data as fast as it arrives). If your PC is not fast enough to run real-time, you can of course just record the signals to disk or cloud locations, and then process them offline. The necessary software is available for other Operating Systems (OSs) (Windows 7, 8 and 10 32- and 64-bit, OS X 10.7.4+ and numerous Linux distributions).

1.4.1 SDR softwares

For SDR systems, you need to have MATLAB and Simulink installed on your computer and the RTL-SDR Hardware Support Package can be downloaded and installed by all users of MATLAB. . These are official software add-ons that provide extra features (such as filter designing tools and frequency domain scopes) in the development environment:

- DSP System Toolbox
- Communications System Toolbox
- Signal Processing Toolbox

MATLAB and Simulink provide you an environment where you can conveniently code and build the receivers, and these toolboxes will provide you the necessary means to implement any SDR receiver algorithm you desire using the Nooelec hardware. Not only do they give you the facilities to design things like the digital filters, decimators and synchronisers your system requires, but also provide tools that allow you to visualize signals in the time and frequency domains as they undergo the demodulation process.

1.4.2 RTL-SDR Hardware Support Package

MathWorks Hardware Support Packages can be considered ‘add-ons’ that provide specific MATLAB and Simulink interfacing support for third-party hardware, including devices such as the Raspberry Pi, the Arduino, various Field Programmable Gate Arrays (FPGAs) and the Universal Software Radio Peripheral (USRP®). Most relevant to this book is the RTL-SDR Hardware

Support Package, which enables interfacing with your RTL-SDR, although you may well use some of the others later on, which will allow you to generate your own low power RF signals from the desktop. This means that you can transmit (Tx) signals to your RTL-SDR, which will receive (Rx) them. Transmitters can either be purchased; built using instructions that we provide in; or you may find some wireless sensors around the home (for example 433MHz or 868 MHz RF low temperature sensors) that transmit signals suitable for the tasks too.

1.5 AIM AND OBJECTIVES OF THE WORK

To learn about, and implement SDR designs at very low cost using the RTL-SDR receiver, with communications algorithms and receivers implemented in real-time using the MATLAB & Simulink software tools.

1. To understand the fundamental design and implementation of SDR systems.
2. Gain an appreciation of the many suitable applications of SDR in radio, mobile, wireless, instrumentation, and general RF receiving and 'listening'.
3. To learn about the core components of an SDR system, and how DSP and PHY layer algorithms allow signals to be extracted and received information presented.
4. Be able to tune across the spectrum to capture and view different signals, and plot live RF spectra on screen.
5. To implement a Software receiver for VHF band using RTL-SDR module.

CHAPTER 2

EXISTING SYSTEM

Over recent years, and probably since the late 1990s, SDR has often been presented — even heralded — as the future solution and design for all RF receivers. It is probably true to say that SDR has been promising solutions for quite a few years, but only now (from 2014) has this become very low cost and widely available at the desktop level. In the past, SDR was more commonly associated with military and research applications, due to its (historically) high cost of implementation. It is useful to review the ‘ultimate’ SDR architecture, along with more common practical implementations, and to understand the technology factors involved.

2.1 GENERAL SDR ARCHITECTURE

At its very simplest conceptual level, SDR comprises of an RF section (antenna, amplifiers and filters) and a very high speed Analogue-to-Digital Converter (ADC) and Digital-to-Analogue Converter (DAC) pair, interfaced with a powerful DSP processor and/ or computing system, as illustrated in. Samples are passed into and out of the DSP section via the ADC and DAC, respectively. Recalling the Nyquist theorem (i.e. that you need to sample a signal at greater than twice the signal bandwidth to retain All information), sampling at, say, $f_s = 4\text{GHz}$, would produce a baseband spectrum from 0 to 2GHz, or $f_s / 2$. Thus, the SDR would be capable of transmitting and receiving all frequencies up to 2GHz, with the modulation and demodulation undertaken in the digital domain.

The components of the simple, conceptual ultimate software defined radio, which in theory will digitize the spectrum from 0Hz to 2GHz. SDRs that can receive from 200KHz to 10GHz or even higher is also available in the market. Figure 2.1 shows a general architecture of a SDR receiver. To develop a complete radio receiver system, we do need to rely on our RF and broadband

antenna colleagues to educate us on some of the engineering issues associated with operating over such a wide frequency range. Ensuring that we use appropriate analogue components are therefore also important considerations. Assuming this is all in order, then from a DSP theory point of view, if we can digitize the RF signal at these sampling frequencies, the rest of the receiver can be implemented purely in software.

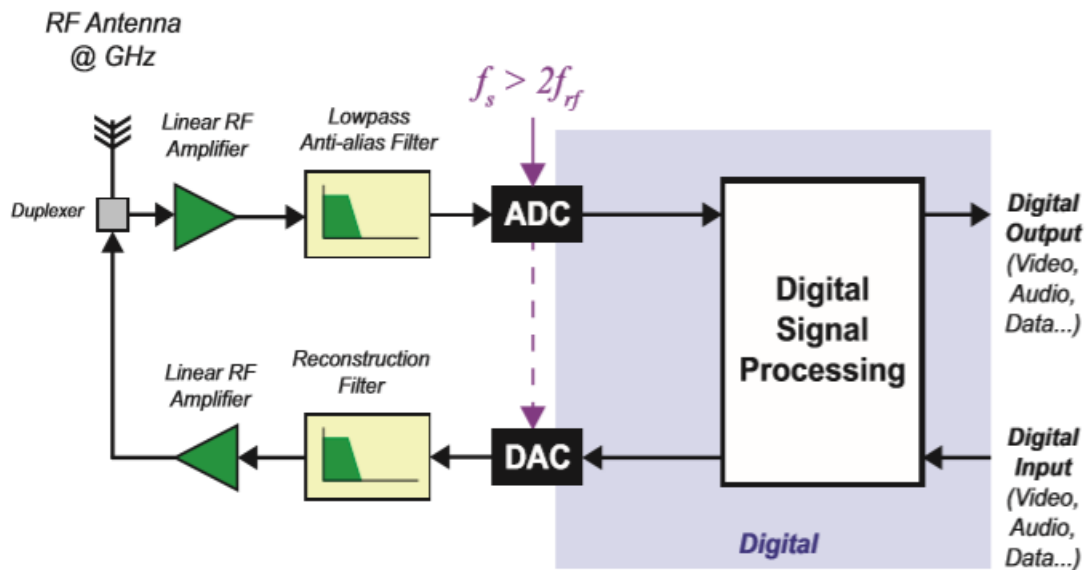


Fig: 2.1 General SDR Architecture

The software would of course be required to run on very high speed processors, or perhaps even on powerful FPGAs (e.g. for extensive, complex standards-based signals such as LTE). These devices perform all of the DSP algorithmic arithmetic required, which are primarily multiplications and additions, as per usual for DSP.

The super-high speed ADCs and DACs needed to interface at multi-GHz sampling rates will become available, and thus the system indicated in will be readily realizable. To remind ourselves of technology timelines and evolutions, back in the late 1990s, a DAC/ADC running at $f_s = 100$ kHz with 16-bit resolution was the latest (and rather expensive) technology. Now, in 2015, devices running at $f_s = 100$ MHz with almost 14-bit resolution are widely available at reasonable cost from companies such as Analog Devices Inc. Samplers operating at GHz

frequencies are available now with 8-bit (or so) resolution, but they are expensive and not yet consumer grade. Despite these current limitations, one thing we can be sure of with technology is that it will advance — so get ready, GHz samplers in our consumer devices are just around the corner!

2.2 QUADRATURE SDR ARCHITECTURE

Figure 2.2 shows the improvements in receiver architecture of software-defined radios. The popular receiver architectures used in contemporary devices are baseband , IF receiver and Zero IF receiver models.

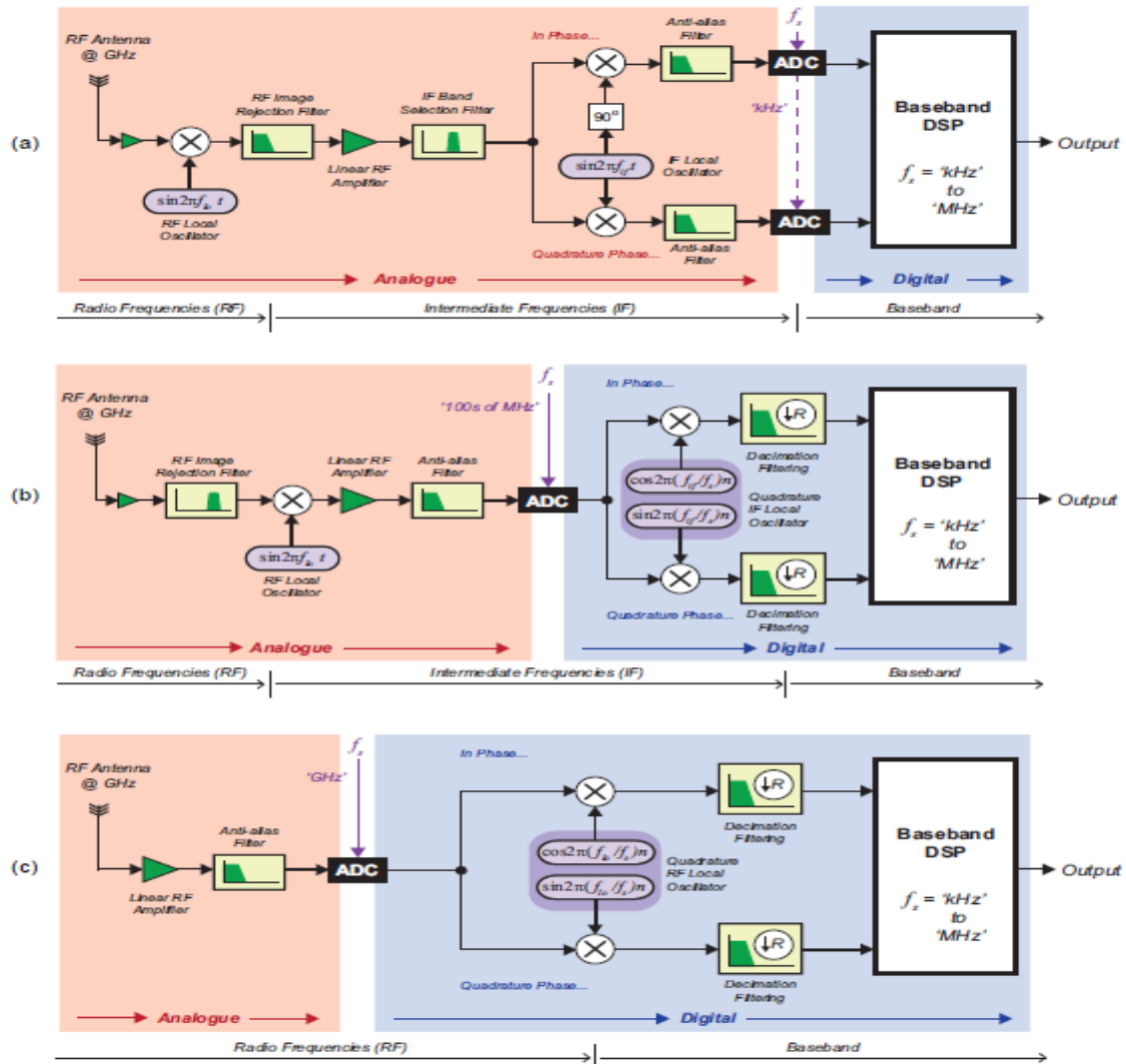


Fig: 2.2: Evolution of SDR receiver architecture: a) the baseband digital radio (b) the IF digital radio (c) the zero-IF digital radio.

2.3 RTL-SDR HARDWARE COMPONENTS

The RTL-SDR USB hardware originated from consumer grade DVB-T receivers, designed to enable users to watch DTV on a computer is *not* originally designed or conceived to be used as generic programmable SDRs, and we owe our thanks to a number of independent engineers and developers in the SDR community, who discovered their potential for use as SDRs. Specifically, they found that the devices could be placed in a ‘test mode’, which essentially bypassed the DVB decoding stage, thus allowing the device to tune over the range 25MHz to 1.75GHz, producing raw, 8-bit IQ data samples at a programmable baseband sampling rate.

The two main components used in the RTL-SDR DVB receivers are a DTV tuner (the Rafael Micro *R820T* is most commonly used, although some RTL-SDRs utilize the Elonics *E4000*) and the Realtek *RTL2832U* DVB-T Coded Orthogonal Frequency Division Multiplex (COFDM) demodulator.

Something we should be clear on is that the RTL-SDR is ***only a receiver***, and cannot be used to generate or transmit signals. Fortunately, there are plenty of RF signals broadcast in the air around you (such as the ones shown in that you can receive with your RTL-SDR with little effort. You may also find that there are plenty of RF devices transmitting around your home and workplace, in the form of temperature and alarm sensors, car key fobs, and even wireless doorbells!

As mentioned earlier, those with a larger budget, or who are working in the classroom/ lab environment, may consider acquiring a SDR transmitter. There are a number of different SDR transmitters available (such as the USRP® hardware) which, when paired with an RTL-SDR, will allow you to implement complete Tx/ Rx communication systems. Examples of this kind will be featured throughout the book.

When the original RTL2832U based DVB devices were shipped, they were

supplied with drivers and software that was only compatible with the Windows OS. Realtek, who designed and produced the demodulator, released a stripped-down operations manual to Video-4-Linux (V4L) developers in early 2010, with a view to encouraging the development of equivalent software and drivers for Linux OSs. A V4L developer named Eric Fry spent time investigating exactly what data was being transmitted over the device's USB interface, and he created some low level drivers that facilitated communication between Linux OSs and the device.

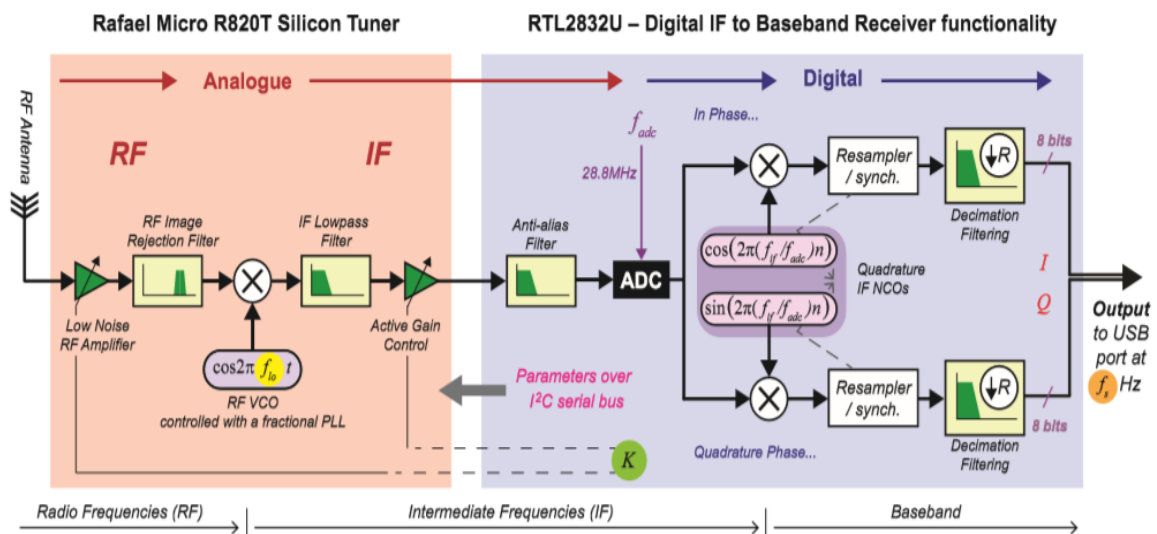


Fig.2.3: Internal architecture of R820 Tuner

The commands transmitted when tuning to a radio station were captured, and used to force the device to stay in this special mode continuously. It turned out to be a test mode, and when the RTL2832U was in this mode, it output 8-bit unsigned samples of baseband IQ data, rather than decoded DVB signals in MPEG2-TS format (as it was designed to operate). Some developers from Osmocom, who had produced an independent SDR device called 'OSMO-SDR', then became involved. These developers had experience in writing software that was able to program the DTV tuners used with the RTL2832U, and after spending some time examining the Windows drivers provided by Realtek, figured out how to program the tuner via the demodulator.

2.3.1 Signal Processing in R820T tuner

Figure 2.2 shows a signal processing flow diagram of the main stages that are carried out on the RTL-SDR. RF signals entering the tuner are down converted to a low-IF using a Voltage Controlled Oscillator (VCO). The VCO is programmable, and is controlled by the RTL2832U over an Inter-Integrated Circuit (I²C) interface. After an Active Gain Control (AGC) stage, which dynamically adjusts the amplitude of the input signal to suit the operating range of the device, the IF signal then requires to be brought down to baseband. The classical method of doing this is to pass the IF signal through an anti-alias filter, sample the output with an ADC, and then downconvert it to baseband using quadrature. Numerically Controlled Oscillators (NCOs) (i.e. a sine and a cosine oscillating at the IF frequency).

When the RTL2832U is operating as it is designed to, this baseband IQ data would be DVB-T demodulated, and an MPEG2-TS (Moving Picture Experts Group Transmission Stream) video stream would be output over the device's USB interface. When it is in 'test mode' however, the final demodulation stage is skipped, and the 8-bit baseband IQ data is output instead. Although no formal documentation for the RTL-SDR or the RTL2832U demodulator has been made public, data sheets for the tuners (the Rafael R820T and the Elonics E4000 can be obtained with a quick online search. A blogger from Japan has also taken the time to create schematics of the device, showing how all of the components interface.

When the original RTL2832U based DVB devices were shipped, they were supplied with drivers and software that was only compatible with the Windows OS. Realtek, who designed and produced the demodulator, released a stripped-down operations manual to Video-4-Linux (V4L) developers in early 2010, with a view to encouraging the development of equivalent software and drivers for Linux Oss. A V4L developer named Eric Fry spent time investigating exactly what data was being transmitted over the device's USB interface, and he created some low level drivers that facilitated communication between Linux OSs and the device.

The Rafael Micro R820T tuner (and its upgrade, the R820T2) use a *low-IF* of 3.57MHz, and downconvert a band of RF signals (roughly 6MHz wide) to this IF. The resulting signal is input to an ‘IF sampling’ connection on the RTL2832U, which tunes to the IF centre frequency and down converts the IF signal to baseband. After this, the RTL2832U samples the signal with its ADC at a rate of 28.8MHz, and performs quadrature demodulation to produce IQ samples. A decimation process is performed to reduce the sampling rate to a lower value, e.g. to 2.8MHz, and the samples are then output over its USB interface.

NooElec is one company with worldwide distribution of these devices. Based on their use of the R820T tuner, the NooElec NESDR RTL-SDR devices are capable of receiving signals from the RF frequency range 25MHz to 1.75GHz, and reliably sampling the frequency spectrum at a rate of up to 2.8MHz. The rate at which samples are output by the RTL-SDR can also be configured, and reducing the rate often improves the performance of baseband processing on less powerful computers. Setting the ‘ f_s ’ value to 2.4MHz, for instance, will configure the RTL2832U to decimate by a factor of 12 (noting the ADC samples at 28.8MHz), meaning that only 1 in every 12 IQ samples is output to the computer. With this reduced sampling rate, the signal the computer receives has a bandwidth of it is a complex signal), and spans from -1.2MHz to 1.2MHz.

2.4 NESDR RTL-SDR RECEIVERS

The first generation NooElec NESDR RTL-SDR itself is about the size of an ordinary USB memory drive, and comes packaged with an antenna and remote (which is included for anyone that wants to use the device for its intended purpose), but with no software or any form of documentation. This is because the user is expected to source their own, depending on what they want to use the device for. More recently (in 2014), NooElec released some new RTL-SDRs that have a Printed Circuit Board (PCB) footprint smaller than 2cm^2 (the *Nano*), and also versions with higher quality components to improve tolerance issues (the *RF Shielded*—set. The latter is more thoroughly shielded to reduce noise arising from its surroundings. Highlights of some of the main components used on two of the more popular variations of the NooElec RTL-SDRs: the *NESDR Mini* (shown in

figure 2.4) and the NESDR Nano are:

- **MCX connector** — for attaching an antenna to the device (an omnidirectional antenna is supplied with it);
- **Electrostatic Discharge (ESD) Diode** — protects the tuner from electrostatic discharge originating from the MCX antenna;
- **R820T** — the tuner chip, which selects a portion of the RF spectrum and down converts to an IF;
- **RTL2832U** — the demodulator chip, which down converts from IF to baseband, digitizes the signal and reduces the sampling rate.
- **28.8MHz clock crystal** — provides a reference for frequency synthesis, and used for the generation of the local oscillator and clock (this component is common to both the R820T and RTL2832U);
- **USB 2.0 Interface** — part of the RTL2832U, used to transfer the baseband IQ data to the host PC;
- **The Infra Red (IR) Sensor** — used to interface with the remote control that is supplied with the device (neither of these are supported when the stick is working as an IF SDR);
- **Serial EEPROM (Electrically Erasable Programmable Read Only Memory)** — holds USB configuration information for the device, and is connected to the RTL2832U via an I²C bus .

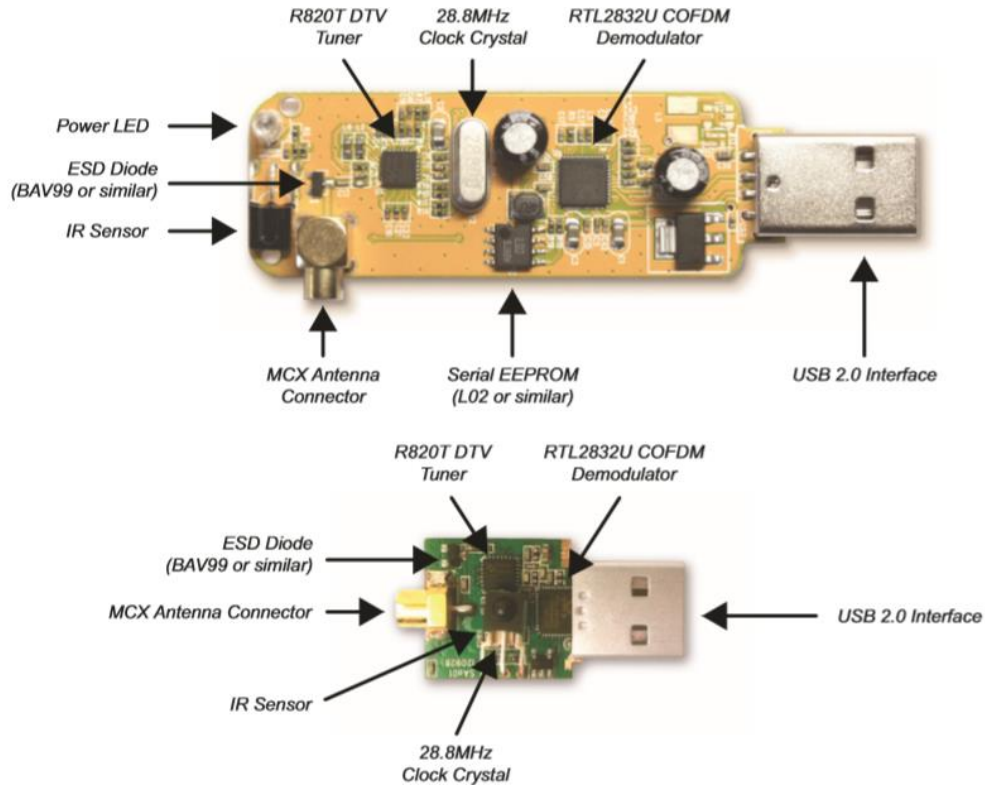


Fig: 2.4 NooElec NESDR RTL-SDR

2.5 INTERFACING WITH THE RTL-SDR FROM MATLAB AND SIMULINK

After installing the RTL-SDR Hardware Support Package, the RTL-SDR Receiver Simulink block illustrated in figure 2.5 will become available for use in your Simulink Library Browser. Similarly, the Hardware Support Package adds support to MATLAB in the form of the comm. SDRRTL Receiver System Object, also shown in the same figure. Both of these facilitate direct communication with any RTL-SDRs attached to your computer, and allow key parameters such as the RF centre frequency f_c , the sampling rate f_s , and the tuner gain K to be set. Additionally, a frequency correction value can be entered through both interfaces to compensate for any hardware tolerance issues associated with the R820T. Looking at figure 2.4, which shows the internal structure of the RTL-SDR, you may notice how these parameters relate to physical values in the hardware. What may not be obvious however, is that the VCO used by the R820T is not set with the value of parameter f_c , but is actually set to a value f_{lo} , where

$$f_{lo} = f_c - f_{if} \quad (2.1)$$

and f_{if} is the tuner's IF frequency. As an example, let's consider that we would like to receive a signal that is being transmitted on an RF carrier with a frequency of 400MHz. This signal must be demodulated down to baseband by the RTL-SDR, but because the R820T has an IF value of $f_{if} = 3.57\text{MHz}$, its VCO is configured to oscillate at a value of:

$$f_{lo} = 400 - 3.57 = 396.43 \text{ MHz} \quad (2.2)$$

When the RF signal (centred at f_c) is mixed with the VCO's sinusoid oscillating at f_{lo} , we get:

$$[\cos(2 \cdot 400 \times 10^6 t)] [\cos(2 \cdot 396.43 \times 10^6 t) + \cos(2 \cdot 3.57 \times 10^6 t)] \quad (2.3)$$

The component at 796.43MHz is attenuated by the R820T's I Flowpass filter and only the IF component (the energy around f_{if}) is propagated forward to the next stage.

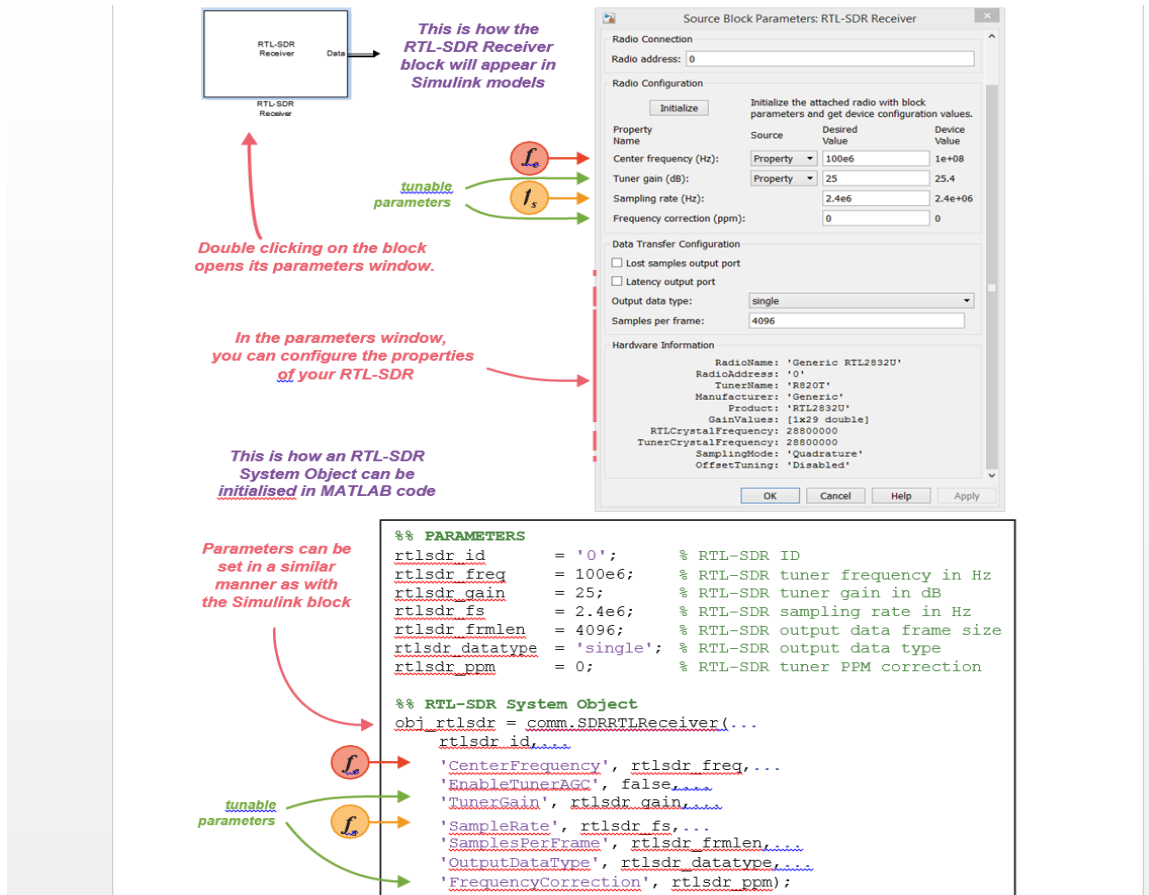


Fig: 2.5: Configuring the RTL-SDR Receiver Simulink block (top), and the RTL-SDR MATLAB System Object (bottom) with identical parameter

2.6 CHALLENGES OF LOW COST DESKTOP SDR

There are a few potential challenges and pitfalls to be aware of in desktop SDRs. The challenges you may encounter could vary from RF and antenna problems, hardware or software issues, not having the right components and tools, or software installation or run-time problems but of course this is engineering, and solving and addressing any or all of the preceding problems is what engineering is all about. The challenges for you to implement real SDR systems, and engineering issues you might encounter are:

1. The omnidirectional antenna not working optimally in the particular band you are using; i.e. not enough gain, or a sensitivity/ polarisation issue that needs to be addressed.
2. The tuner hardware in the RTL-SDR having a frequency offset or error, and further, that the error drifts with time, or as the device heats up.
3. Challenges with the front end gain and AGC of the RTL-SDR, leading to saturation or similar problems.
4. Your computer not being powerful enough to run the DSP algorithms in MATLAB or Simulink in real-time, perhaps meaning you require the use of a faster machine.
5. Complexity of implementing synchronisation and timing circuits.
6. Issues arising from being in a very busy RF environment, or, at the other end of the scale, perhaps in an environment where there are very few RF signals to receive (such as in rural areas or inside shielded buildings or rooms).

2.7 PROBLEM STATEMENT

Though various implementation of RTL SDR receivers are found in the literature, there are drawbacks and challenges in these desktop systems. Some of the problem statements identified are :

- Requires sound programming skills
- Most SDR implementations are based on Linux
- Difficult implement signal processing algorithms in C and C++.
- Cost of the Dongle are high.

2.8 OBJECTIVE OF THE WORK

The main objectives of this project is to address these aforementioned issues and implement a low cost SDR based weather satellite receiver operating in the VHF band at 137MHz. HackRF SDR is choosen for this work as it is more

powerful than conventional dongle based SDRs. A turnstile antenna that is circularly polarised is used for the reception of signals from the satellite. Raspberry Pi Board is used along with SDR for conversion of satellite data into images which can be streamed to a display device.

CHAPTER 3

PROPOSED SYSTEM

A basic understanding of the HackRF One device and the GNU Radio companion software through written instructions, pictures, and video tutorials in order to demonstrate the process of receiving, broadcasting, retransmitting, and live transmitting signals. HackRF One is a Software Defined Radio (SDR) peripheral capable of transmission or reception of radio signals. The Radio signals are ranging from 1 MHz to 6 GHz, designed to enable test and development of modern and next-generation radio technologies. HackRF One covers many licensed and unlicensed ham radio bands. HackRF One is an open-source hardware platform that can be used as a USB peripheral or programmed for stand-alone operation.

3.1 SETTING UP THE HACKRF ONE DEVICE

1. Attach the antenna to the antenna port after removing the red safety cap. **Attaching the antenna first is vital because running the HackRF One without an antenna can damage the device's hardware.**
2. Ubuntu Linux is recommended as the operating system alongside GNU Radio.
3. Plug the HackRF One into the computer with the Micro-USB to USB cable. Confirm that the first three LED lights are illuminated to ensure the device is working.

3.1.1 Setting up the work environment

1. Before getting started, the HackRF One libraries must be installed on the computer to start using the device.
 - a. The libraries can be installed with the Terminal command *"sudo apt-get install hackrf"*
2. After installation, type the command *"hackrf_info"* to verify that the HRF1 device is connected.

- a. After a successful installation, the terminal should respond with *“Found HackRF board”*.
- b. If the terminal shows the HRF1 as not connected, troubleshoot by verifying wires are plugged in and power is being supplied to the device.

3.1.2 Installing GNU Radio Companion

1. In order to install the GNU Radio Companion software, open a terminal window.
2. Type in the command: *“apt-get install gnuradio”*.
 - a. If the root user is not logged in, type *“sudo”* in front of this command.
3. Now, open GNU Radio Companion using the command: *“gnuradio-companion”*.
 - a. Once initialized, GNU Radio Companion displays its user interface, which automatically opens an **untitled project**. This will show an empty white space at the center, which will be used to build a **flowgraph**. The play button on the top toolbar is used to execute the flowgraph. The red ‘X’ next to it can be used to kill the program, but it is recommended to use the red ‘X’ in the popup window when the program is running. Using the kill button in the top toolbar will cause issues with file sinks.
 - b. The flow graph itself is divided into 4 components:

3.2 RECEIVING FM RADIO SIGNALS

This section outlines how to receive FM radio signals from the HackRF One using GNU Radio Companion in order to visualize the signal in a flowgraph. This section also contains a video tutorial.

1. Upon starting the GNU Radio Companion software, there will be two boxes already present on the flowgraph area: *Variable* and *Options*. These default to a set value, which can be altered, as mentioned above. In the *Options* block, the *Generate Options* should be set to *QT GUI*.

2. In the *Variable* block, change the sample rate (*samp_rate*) to the correct value. The lowest recommended sample rate is 2e6 (2 million) Hz, and the highest sample rate used should be 20e6 (20 million) Hz. It is recommended to begin at 4e6 Hz, and decrease as needed. A computer with less processing speed should not have a high sample rate frequency.
3. To begin creating the flowgraph to receive FM radio signals, add an *osmocom source* function via (no module specified) > Source > *osmocom Source* (or search for *osmocom source* via the magnifying glass, which is found on the right side of the top menu). This enables the ability to receive data captured by the HackRF One device.
 - a. The sample rate should already be set to *samp_rate*, which was assigned in the *Variable* box. Changing the *samp_rate* in the *Variable* box will reflect throughout the rest of the flowgraph.
 - b. Change the *Ch0 Frequency (Hz)* to the radio station that is strongest in your area. For example, this should be formatted as "98.9e6". Setting the *osmocom source* > *Ch0 frequency* tells the HackRF One device which radio channel to tune to.
 - c. It is recommended to turn off the RF amplifier by setting the *Ch0: RF Gain (dB)* value to 0. The *RF Gain* amplifies the frequency source at its default setting of 10 dB, but it is not necessary or recommended to have this setting turned on.
4. Now, add the *DC Blocker*, which filters out the direct current from the power supply to the HRF1, which can interfere with the device. If there is no *DC Blocker* present in the flowgraph, there will be a peak in the center of the flowgraph when being executed, which can be misinterpreted as a signal.
 - a. Set the length value to 1024.
 - b. Connect the *osmocom Source* block to the *DC Blocker* with an arrow.
5. In order to visualize the signals being received by the HackRF One device, the frequency sink function needs to be added to the flowgraph. This function can be found via Instrumentation > QT > *QT GUI Sink* (or by searching for *QT GUI Sink*).

- a. Set the *Center Frequency (Hz)* to the frequency from *Ch0 Frequency (Hz)* in the *osmocom Source* function. For example, 98.9e6.
 - b. In order to visualize the whole range of FM radio signals, change the *bandwidth (Hz)* to 20e6. This is normally set to your sample rate by default, but changing the value will not affect your *samp_rate* in the *Variable* function. Also, the focus of the flowgraph will still be on the frequency set as the *Ch0 Frequency* while also showing the full range of radio channels in the area.
 - c. The *Update Rate* can be changed from 10 to 5. This indicates the amount of times per second the graph updates. 5 will cause less strain on your system if you do not have a high end computer.
 - d. Change *Show RF Frequency* to “yes”. This will display the frequencies on the x-axis of the frequency sink correctly.
 - e. Lastly, connect the *QT GUI Sink* (in) function to the *osmocom Source* (out) function with an arrow. The flowgraph should look like this:
6. Press the green play button at the top in order to run the flowgraph. Upon pressing it, a prompt to save the flowgraph will appear in a new window. The file will autosave every time afterwards. After the file saves, the python script will be generated in the bottom left of the screen and the visualization will show on the main area of the screen.
 - a. The first tab in the Top Block window will show the Frequency Display. There should be obvious differences between the strong signals and the rest of the frequencies. For example, channels 98.9, 91.3, and 102.9 are the strongest in the Burlington, VT area.
7. To stop the flowgraph, click the red X in the top left of the pop up window. Sometimes, the HackRF One may need to be reset (via the reset button

- on the device) because it will continue receiving signals even after the flowgraph is closed.
8. To listen to the signals, a few things need to be done in addition to the flowgraph that has already been built. The *Low Pass Filter* function (found via searching for *Low Pass Filter*) will focus on the center frequency and reduce noise from the surrounding stations.
 - a. The *Decimation* value should be set to 20, which will divide the sample rate by 20. This will get the sample rate to the same rate as the audio.
 - b. The *Cutoff Frequency* should be $75\text{e}3$. This is how wide the signal is.
 - c. The *Transition Width* should be $25\text{e}3$. This is how sharply the unneeded signals are filtered out, and applies a more focused center frequency. The lower the transition width, the sharper the transition width. $25\text{e}3$ is a good width value for FM radio.
 - d. Connect the *Low Pass Filter* function to the *osmocom Source* function.
 9. A *Rational Resampler* should now be added as a way to modify the sample rate by a ratio, as GNU radio companion will only allow the sample rate to be changed by integers. By setting this rate as 12 to 5, this will help move the sample rate closer to the targeted audio frequency. If the sample rate is faster than the actual audio rate, then it will affect the speed of the audio and distort the sound, so this can fix that problem.
 - a. Connect this function to the *Low Pass Filter* block.
 10. In addition, the Wide Band Frequency function is needed. This is found by searching for *WBFM Receive*. This block will have one blue input (like the rest of the inputs/outputs on the flowgraph) and one orange output. Blue outputs are complex numbers so there is a real and imaginary number being received. The *WBFM Receive* function is combining both complex numbers and creating one output via the orange real number output.
 - a. Set the *Quadrature Rate* to $480\text{e}3$, which is recommended for the FM radio audio receive task.

- b. Set the *Audio Decimation* to 10, which will reduce the audio rate to 48 KHz since you're dividing it by 10. This is the rate at which the audio will be heard.
 - c. Connect this function to the *Rational Resampler*.
- 11. The last function needed is the *Audio Sink* (found via searching for *Audio Sink*). This will allow you to actually hear the audio from the radio station set as the center frequency. This block has an orange input, which will be connected to the orange *WBFM Receive* output.
 - a. Change the sample rate to 48 KHz from the drop down menu.
- 12. Now, execute the flowgraph again by clicking the green play button in the menu. The audio should immediately begin playing and the radio signals for the center frequency should be visualized in the flowgraph. The audio should be clear, like listening to a normal radio, but it is dependent on the processing speed of the computer.

3.2.1 Customization of Receiving FM Radio Signals

This section outlines some of the additional functionality that GRC presents, like changing the FM radio station in real time and changing volume from the flowgraph view. This section also contains a video tutorial.

1. From the GRC flowgraph already created from the first part of this tutorial, further customizable options are available in order to listen to a wider range of frequencies and change the volume.
2. First, copy the *Variable* function block and paste it twice so there are three *Variable* blocks present in the flowgraph. These will aid in changing the radio station being listened to, and with updating the sample rate. The ID tags in each of the three blocks will, by default, be named *samp_rate*, *samp_rate0*, and *samp_rate1*.
 - a. In the second *Variable* block, change the ID of *samp_rate0* to *channel_width*, with the value of 200e3.
 - b. These are variables and can be set in any blocks to universally change the variables in the rest of the flowgraph. After setting the

channel width to 200e3, change the decimation in the *Low Pass Filter* function to $\text{int}(\text{samp_rate}/\text{channel_width})$. Since radio waves are transmitted at 480 KHz, this should be the final value of the flowgraph. By dividing the sample rate by the channel width in the Low Pass filter, you will not have to change the decimation values if you change the sample rate.

- c. In the third *Variable* block, change the ID of *samp_rate1* to *center_freq*, with the value of the target radio station. For example, 98.9e6 in the Burlington, VT area.
3. In order to be able to change radio stations within the flowgraph without reentering values throughout the flowgraph, add a *QT GUI Range*. This block has no input or output values.
 - a. Change the *ID* of the block to *channel_freq*.
 - b. Change the *Default Value* to the targeted radio station. For example, 98.9e6.
 - c. Change the *Start* value to 87.9e5 and the *Stop* value to 107.9e6. These values represent the entire range at which FM radio signals broadcast.
 - d. Change the *Step* to 200e3. This represents the distance between each tick on the slider being added. Because FM radio stations are 200 MHz wide, set this to 200e3 so each tick on the slider represents a different radio station.
4. In order to control volume within the flowgraph via a slider, add a second *QT GUI Range* to the flowgraph. This block has no input or output values.
 - a. Change the *ID* of the block to *audio_gain*.
 - b. Change the *Default Value* to 1.
 - c. Change the *Start* value to 0 and the *Stop* value to 10. These values represent the highest and lowest audio volume.
 - d. Change the *Step* to 0.1.
5. Now that all of the values needed are within the *Variable* and *QT GUI Range* blocks, they need to be implemented into the flowgraph. In order to change the channel frequency, start by adding a *Signal Source* block.

On its own, this would just generate the one targeted frequency, with a spike at the default 1000 Hz frequency. The *Signal Source* function can aid in changing the radio station by creating the variables in this block and multiplying those signals by the values in the *osmocom Source*.

- a. Change the *Frequency* to "*center_freq - channel_freq*" to subtract the channel frequency from the center frequency.
 - b. Do not connect this block to any other function yet.
6. Now, add a *Multiply* function. This block has two input sources and one output source.
 - a. Delete the arrow between *osmocom Source* and the *Low Pass Filter* blocks. Do this by clicking on the connecting arrow and dragging away.
 - b. Position the *Multiply* function and connect its output to the *Low Pass Filter* input. Connect the *Signal Source* output to one of the *Multiply* inputs. Connect the *osmocom Source* output to the *Multiply* function input.
 - i. Notice that the *Signal Source* block has a grey input area. This is because it is not necessary to have an input connection with the *Signal Source* function.
7. After adding these functions, execute the flowgraph. The radio station can be changed by moving the *channel_freq* slider.
 - a. If no peak shows up initially, start slowly moving the slider to search for the signal. Once found, set the center frequency to this value so it doesn't have to be searched for again.
8. To add an additional slider to control volume, delete the arrow connecting the *WBFM Receive* function to the *Audio Sink* function.
9. Add a *Multiply Const* block, which, instead of multiplying two values from two different blocks, it will multiply the input value by the value set within this block.
 - a. Notice that the *Multiply Const* block has blue inputs and outputs, while the *WBFM Receive* block has an orange output. The easiest way to change from *Complex* to *Float* inputs and outputs is to click

on the block and press the keyboard's down arrow. The input will change from blue to orange to green to yellow. Another way to change the input and output in order to match is to alter the properties within the *Multiply Const* block. Double click the block and change the *ID type* from *Complex* to *Float* from the drop down menu.

- b. Connect the *WBFM Receive* output to the *Multiply Const* input, and the *Multiply Const* output to the *Audio Sink* input.
- c. Set the Constant in the *Multiply Const* block to *audio_gain*.
- d. Now, while executing the flowgraph, there will be a slider at the top of the screen labeled *audio_gain* to control the volume, along with the *channel_freq* slider to change the radio station.

3.3 TRANSMITTING FM RADIO SIGNALS

This section outlines how to transmit radio signals to an open radio station through the use of the HackRF One device and the GNU Radio Companion software. This section also contains a video tutorial.

1. Upon starting the GNU Radio Companion software, there will be two boxes already present on the flowgraph area: *Variable* and *Options*. These default to a set rate and value, which can be altered, as mentioned above. In the *Options* block, the *Generate Options* should be set to *QT GUI*.
2. In the *Variable* block, change the sample rate (*samp_rate*) to the desired value. The lowest recommended sample rate is 2e6 (2 million), and the highest recommended sample rate is 20e6 (20 million). A good starting sample rate for transmitting radio signals is 2e6 (2 million), and increase from there. A computer with less processing speed should not have a high sample rate frequency.
3. To even begin the process of transmitting audio to an FM radio station via the HRF1 device, obtain a file source to transmit, such as any available '.wav' audio file. Please note that every '.wav' file may have its own audio rate (for example, 22050 Hz). This can be confirmed by

- checking the file's properties (by right-clicking the file in the folder and checking its Properties > Audio > Sample rate).
4. In GRC, add the *Wav File Source* function and place it into the flowgraph.
 - a. From the *file* option within the block, add the desired '.wav' file.
 5. Add a *WBFM Transmit* function to the flowgraph.
 - a. Set both the *Audio Rate* and *Quadrature Rate* to an integer four times the file's audio rate (for example, 88200, which needs to be an integer, not in scientific notation). This will generate a clearer sound from the audio file transmitted from the HRF1 without lowering the volume.
 - b. Set *Tau* to 25e-6. This will increase volume.
 - c. Connect the WBFM Transmit input to the *Wav File Source* output with an arrow.
 6. Next, add a *Rational Resampler* block to the flowgraph.
 - a. Set the *Interpolation* value to 90 in order to decimate the sample rate to the correct value. If the audio rate of the .wav file is 44100 Hz, set the interpolation to 45.
 7. Obtain a *QT GUI Range* to determine the broadcasting station.
 - a. Rename the ID as *freq*.
 - b. Set the *Default Value* and *Start* value inputs of the desired radio station to broadcast (for example, 87.7e6). Set the *Stop* frequency to 107.7e6, which is the highest FM radio value. Additionally, the *Step* value represents the width of each FM Radio channel in *Step*. Set this to 200e3.
 8. Add an *Osmocom Sink* function, which will allow the Hack RF to transmit the signals.
 - a. Set the *Ch0: Frequency* to *freq*, which will enable the utilization of a slider within the pop up window in order to change the broadcast radio station.
 9. Click the play button and save the flowgraph as a file. Upon executing the flowgraph, a new window will appear with just the slider. The .wav file

uploaded to the *Wav File Source* will begin playing. Depending on the quality of the antennae used, the sound may be slightly distorted.

3.4 RETRANSMITTING FM RADIO SIGNALS

This section outlines the procedure to record an audio sample from a radio station and retransmit it to a different radio station. First, follow the tutorials above or open the previously saved files for receiving (which will be referred to as *tutorial 1.grc*) and transmitting (which will be referred to as *tutorial 2.grc*) FM radio signals. Open these files in separate tabs of the same window.

1. In the *tutorial 1* tab, add a *Wav File Sink* function to allow the signals received to be saved as a '.wav' file.
 - a. Change the file path in the *File* prompt to somewhere on the desktop (for example, an empty folder on the desktop).
 - b. Change the number of channels (*N Channels*) to 2. This value needs to be 2 in order to double the number of samples to successfully save the file. This will enable the *Wav File Sink* function to have 2 inputs.
 - c. Set the *Sample Rate* to 48000.
 - d. Connect one of the inputs of the *Wav File Sink* to the *Multiply Const.*
3. Obtain a *Constant Source* function, and set the value to 0. This will keep the values within the .wav as real numbers.
 - a. Connect the second *Wav File Sink* input to the *Constant Source* output.
4. Execute the flowgraph for about 10 seconds. GRC will immediately begin recording the audio sample. It is recommended to only play the flowgraph for a few seconds because .wav files can become very large, very fast.
 - a. Again, a reminder to close the flowgraph via the red x in the upper left-hand corner of the flowgraph window, not the red x by the play button. Otherwise, the .wav file will not save properly.

5. Now, in the tab containing *tutorial 2*, from the Wav File Source function, select the recorded filename from *Tutorial 1* as the desired file to transmit.
6. Change the *Interpolation* value in the *Rational Resampler* function to 45. This is half of the previous *Interpolation* value of 90. This will prevent the file from sounding sped up or slowed down, since the sample rate of the recorded .wav file is different from the *WBFM Transmit Audio Rate*.
7. Now, by executing the flowgraph, the received file recorded from *tutorial 1* should play to any radio near the HRF1 antenna. That concludes the tutorial to retransmit FM radio files via the HRF1.

3.5 LIVE BROADCASTING FM RADIO SIGNALS

1. In order to live broadcast, just delete the *Wav File Source* function. Replace it with the *Audio Source* function. This enables the computer's sound card, so instead of transmitting a recorded file, the audio played will be any sound picked up by the computer's microphone.
 - a. In the *Audio Source* block, change the *Sample Rate* to 44.1 KHz. This value aligns better with the other values in the flowgraph.
 - b. Connect the *Audio Source* output to the *WBFM Transmit* input.
2. By changing the *Audio Source > Sample Rate*, the *Variable > Sample Rate* should also be changed to 44100.
3. Set the *osmocom Sink > Sample Rate* back to 2000000 (2M). Doing this will unlink the default *samp_rate* within the function to the *Variable* block. These two values need to be different because the signal will not transmit correctly should the sample rate still be linked.
4. Execute the flowgraph and speak into the computer's microphone. Your voice should be heard by receiving radios near the HRF1 antenna.

3.6 SATELLITE RECEIVER USING HACKRF ONE

The NOAA satellites are LEO satellites. They are Low Earth Orbiting satellites. As a result of this, they orbit the Earth around 4 to 5 times a day. This necessitates the fact that they need to be tracked. Only when the satellite and receiver system are in line of sight is the APT signal available. There should also be some minimum elevation level. The task of locating the satellite traversal may be complex, but there are several software that can be used for the prediction.

Hack-RF SDR is utilized in this work to implement the satellite receiver. Custom designed antenna is connected to the SDR to pickup satellite signal. Raspberry Pi single-board computer is used to process the data received by SDR and reproduce the satellite weather images on a display device or conventional monitor.

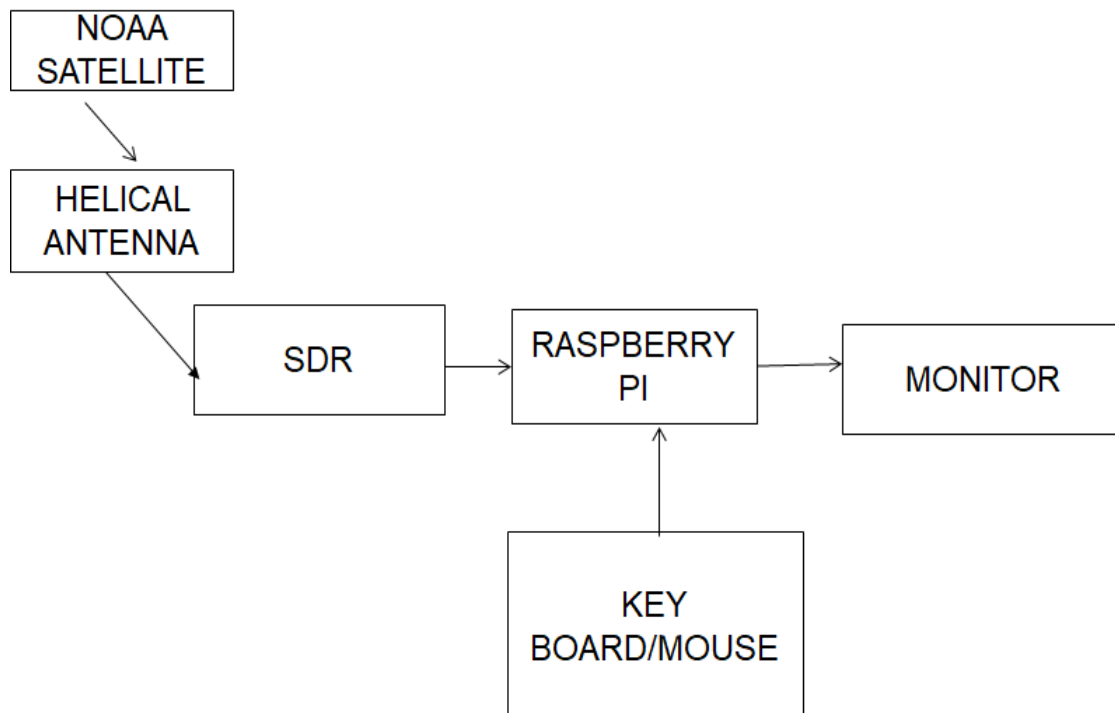


Fig: 3.1: Block diagram of SDR based Satellite receiver

CHAPTER 4

RESULTS AND DISCUSSION

4.1 SATELLITE RECEIVER SET-UP

The setup used to receive signals from NOAA satellite is shown in figure 4.1. It includes the HackRF SDR, RaspberryPi and Laptop.



Fig: 4.1: Setup used for reception of satellite signals

Figure 4.2 shows the antenna mast installed on roof top to pick up signals from the NOAA satellite. In this work, a helical shaped antenna is designed for the VHF band (137MHz) for satellite signal reception.



Fig: 4.3: Antenna mast for satellite signal reception



Fig: 4.4 Test Satellite images received in the laptop screen

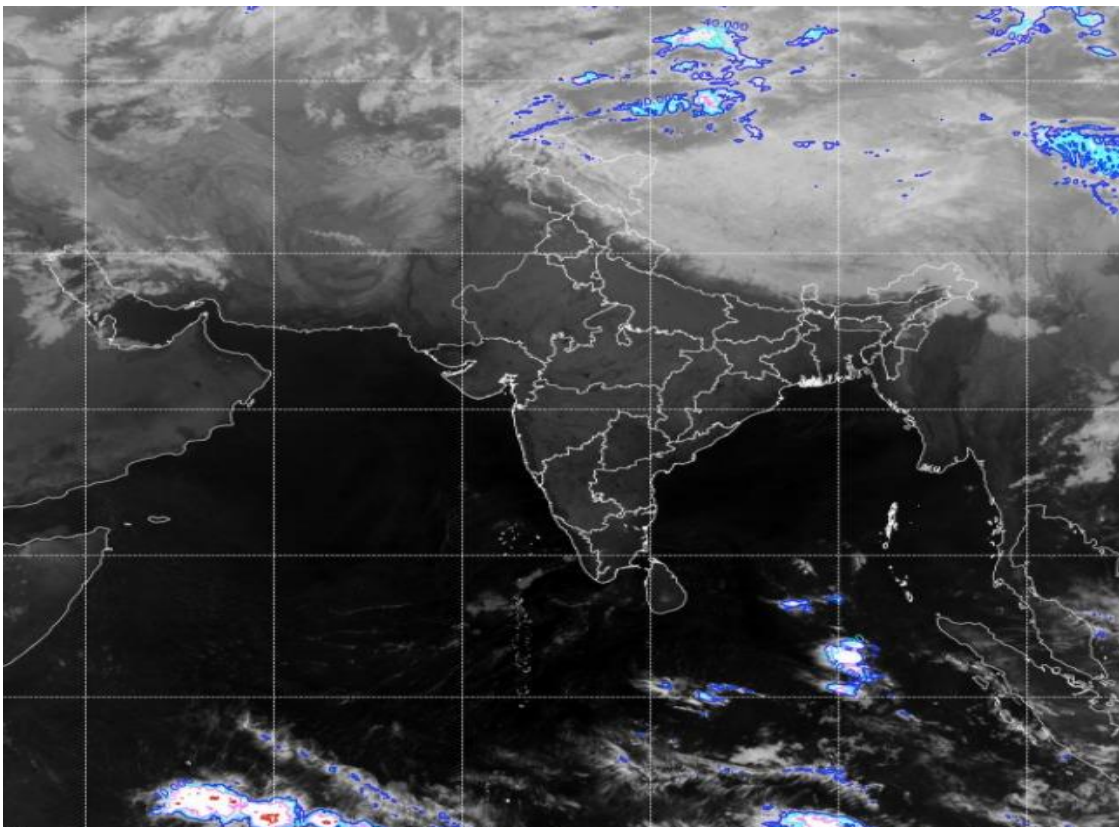


Fig: 4.5 Images of clouds received in the laptop

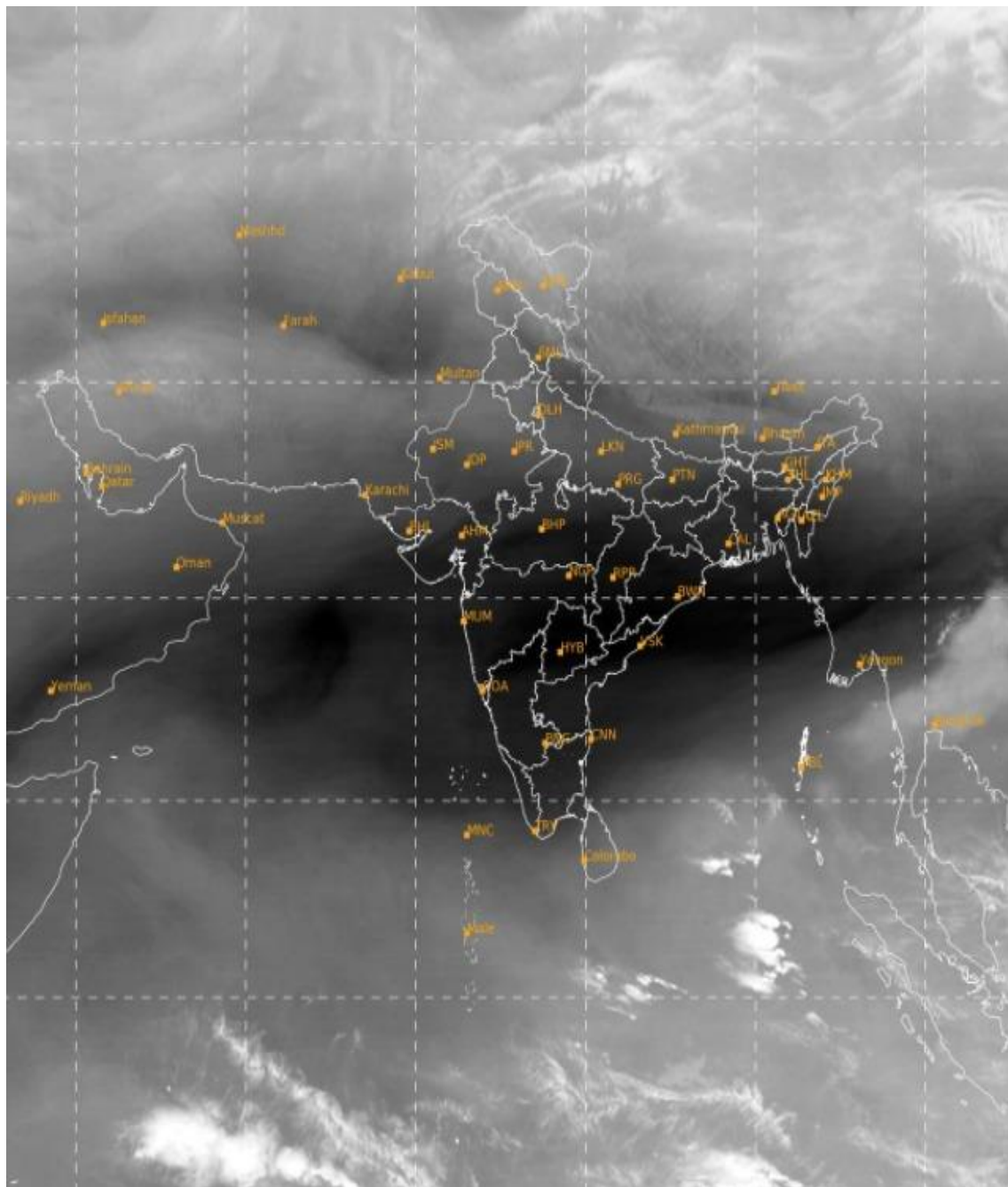


Fig: 4.6 Images of water vapor received from satellite.

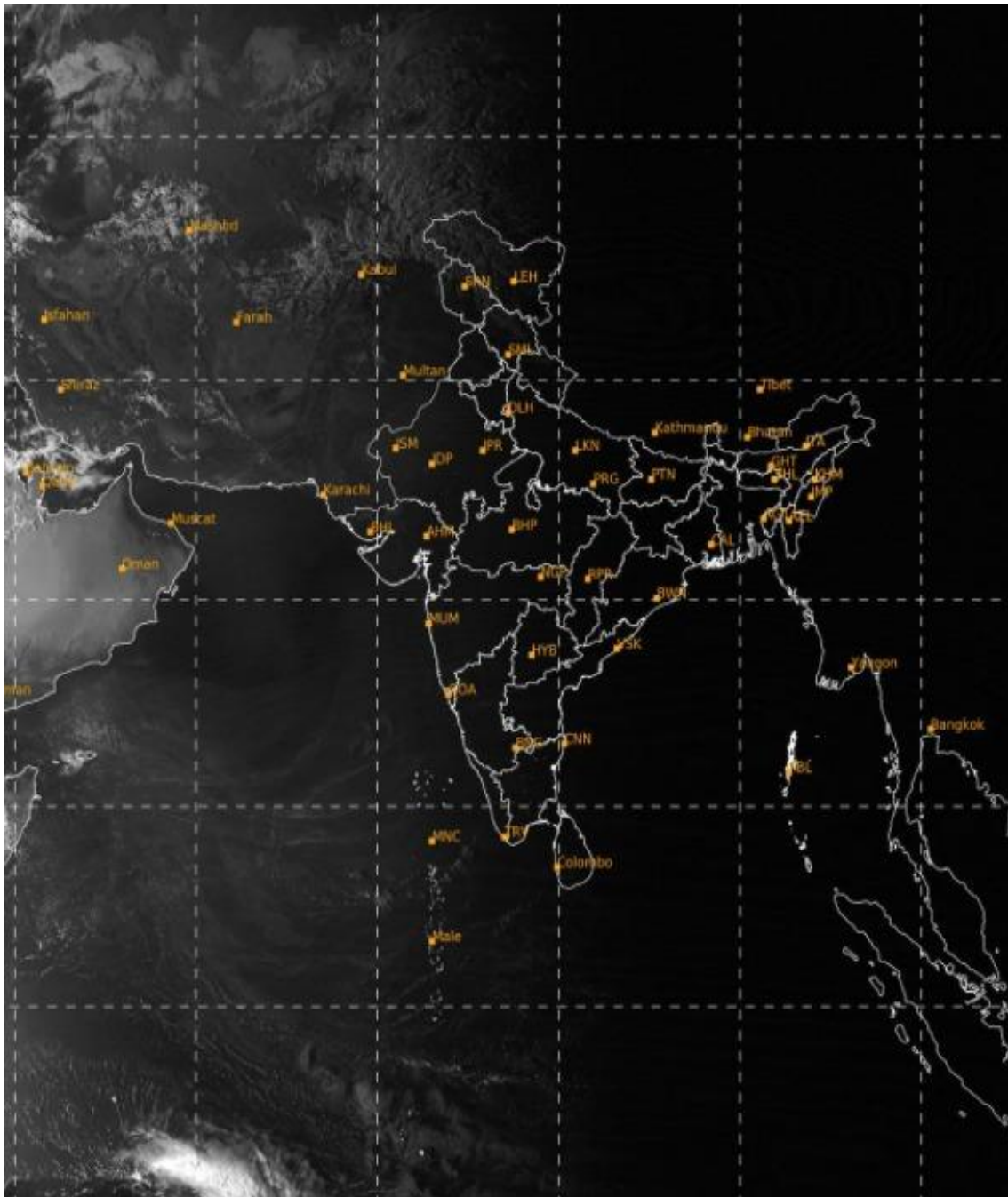


Fig: 4.7: Images received under visible light condition

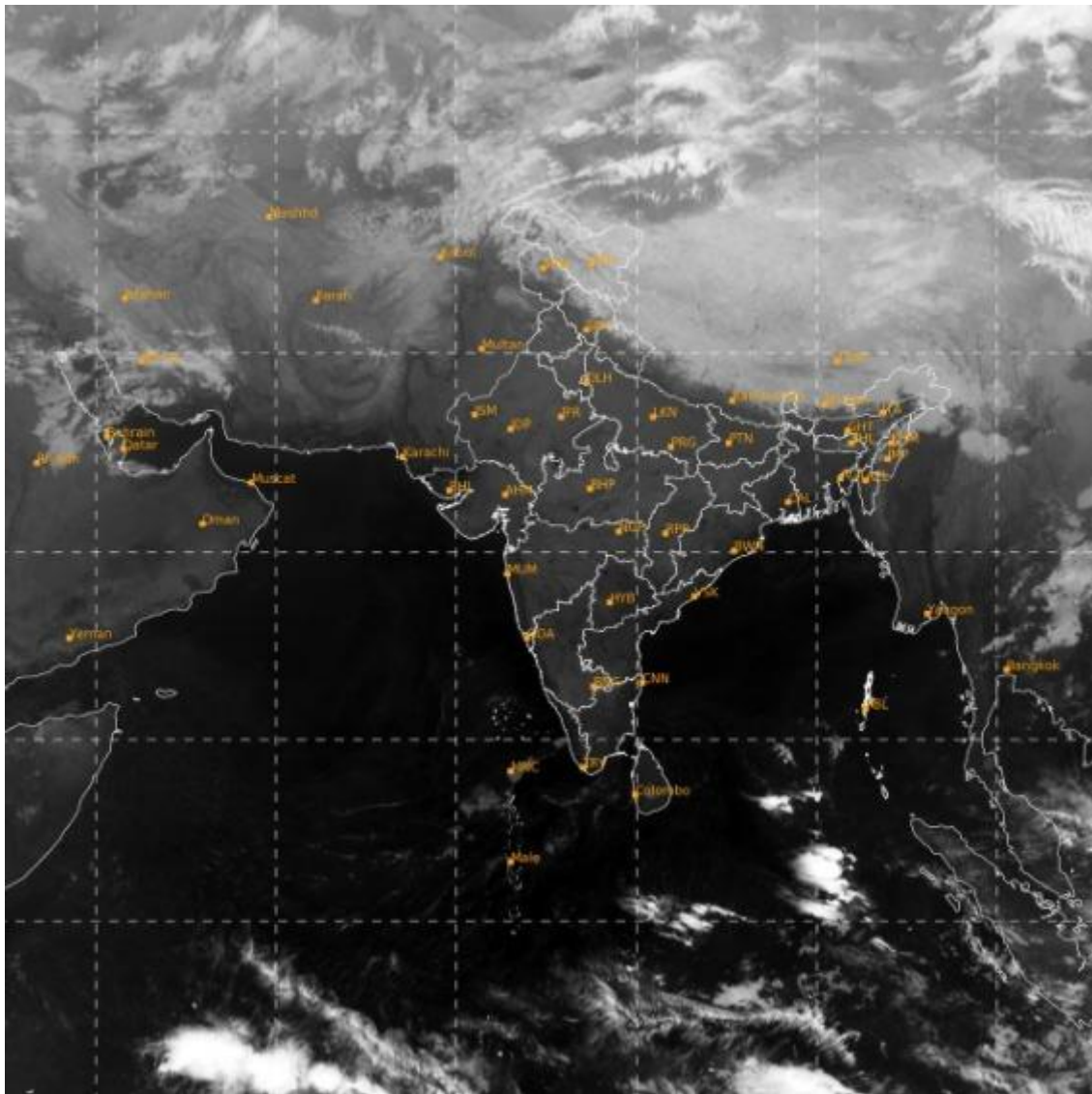


Fig: 4.7: Images received under Infrared light setup

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

In this project, we have implemented weather station using a software defined radio (SDR), which has received real-time satellite weather forecast grey scale images. This system can be used in a disastrous situation where there is no access to internet or cellphone signals.

5.2 FUTURE SCOPE

After documenting its setup, use, and functionality, our team proceeded to further research practical applications of the HackRF One. We found our research can be applied to a number of different radio spectrums and devices. For example, we explored modern technologies and consumer products that utilize radio frequency such as: Key Fobs from automobiles, NFC, and Bluetooth CR/LE.

With the rise of consumer products that utilize wireless technologies, the HackRF One is a handy tool that allows the examination of communicating devices. We hope that our research will help others further the practical application of the HackRF One in future projects.

REFERENCES

- [1] Abramson N., “Development of the ALOHANET”, IEEE Transactions on Information Theory, vol. 31, pp. 119-123, 1985.
- [2] Behnke, P., Soberal, D., Bredeweg, S., Dunne, B., Sterian, A., and Furton, D. (2013). Senior capstone: A software defined radio design for amateur astronomy. In Interdisciplinary Engineering Design Education Conference (IEDEC), 2013, pages 104–111.
- [3] Bondyopadhyay P. K. and G. Marconi, “The Father of Long Distance Radio Communication - An Engineer’s Tribute”, Proceedings of the 25th European Microwave Conference, vol. 2, pp. 879-885, 1995
- [4] Butler C.M. “The Equivalent Radius of a Narrow Conducting Strip”, IEEE Transaction on antennas and propagation, Vol. 30, No. 4, 755–758 pp, 1982.
- [5] Chowdhury. S.K, “Impedance of Multielement Dipoles”, IEEE Transaction on Antennas and Propagation, Vol.15, No. 5, pp. 682–684, September 1971.
- [6] Clapp, R. and Maxwell, J. (1967). Complex-correlation radiometer. Antennas and Propagation, IEEE Transactions on, 15(2):286–290.
- [7] Daniel F. Macedo, Dorgival Guedes, Luiz F.M. Vieira, Macros A.M Vieira, and Michele Nogueira., “ Programmable Network- From Software – Defined Radio to Software Defined Networking” ,IEEE comm. surveys & tutorials, vol.17,No.2,pp.1002-1008, 2015.
- [8] De Roo, R., Ruf, C., and Sabet, K. (2007). An I-band radio frequency interference (rfi) detection and mitigation testbed for microwave radiometry. In Geoscience and Remote Sensing Symposium, 2007. IGARSS 2007. IEEE International, pages 2718–2721.
- [9] Ellingson, S., Hampson, G., and Johnson, J. (2003). Design of an I-band microwave radiometer with active mitigation of interference. In Geoscience and Remote Sensing Symposium, 2003. IGARSS '03. Proceedings. 2003 IEEE International, volume 3, pages 1751–1753.
- [10] Erbas, C., Hornbuckle, B., and De Roo, R. (2006). Iowa state university/the university of michigan direct sampling digital radiometer. In

- Geoscience and Remote Sensing 78 Symposium, 2006. IGARSS 2006. IEEE International Conference on, pages 3074– 3077.
- [11] Evans, G. and McLeish, C. W. (1977). RF Radiometer Handbook. Artech House, Dedham, MA.
- [12] Forte, G., Tarongi Bauza, J., dePau, V., Vall Ilossera, M., and Camps, A. (2013). Experimental study on the performance of rfi detection algorithms in microwave radiometry: Toward an optimum combined test. Geoscience and Remote Sensing, IEEE Transactions on, 51(10):4936–4944.
- [13] Fujimoto, K. (1964), “On the correlation radiometer technique. Microwave Theory and Techniques”, IEEE Transactions on, 12(2):203–212.
- [14] Hardy, W. N., Gray, K., and Love, A. (1974). An s-band radiometer design with high absolute precision. Microwave Theory and Techniques, IEEE Transactions on, 22(4):382– 390.
- [15] Hazim Salah Abdulsatar K. J. T., Ali Hashim Jryan, “Low power Transceiver Structure for Wireless and Mobile Systems Based SDR Technology Using MATLAB and System Generator”, International Journal of Engineering and Advanced Technology (IJEAT), vol. 3, pp. 205-214, 2013.
- [16] Haziza N., M. Kassab, and R. Knopp, “Multitechnology vehicular cooperative system based on Software Defined Radio (SDR)”, Communication Technologies for Vehicles, pp 84-95, Springer, 2012
- [17] Jefferson L. Xu, MengChu Zhou, “Software-Defined Radio Equipped With Rapid Modulation Recognition”, IEEE Trans. On Vehicular Technology, vol. 59, No. 4, pp. 650-658, May 2010.
- [18] Jonard, F., Weihermuller, L., Jadoon, K., Schwank, M., Vereecken, H., and Lambot, S. (2011). Mapping field-scale soil moisture with I-band radiometer and groundpenetrating radar over bare soil. Geoscience and Remote Sensing, IEEE Transactions on, 49(8):2863–2875.
- [19] Jondral, F. K. (2005). Software-defined radio: basics and evolution to cognitive radio. EURASIP journal on wireless communications and networking, 2005(3):275–283
- [20] Kleinrock L., “An Early History of the Internet”, IEEE Communications Magazine, vol. 48, pp. 26-36, 2010

- [21] Liu. J, Xue. Q, "Microstrip magnetic dipole Yagi array antenna with end fire radiation and vertical polarization", IEEE Transactions on Antennas and Wave Propagation, Vol. 61, No. 3, pp. 1140-1147, March 2013.
- [22] Maguire J. G. Q., "Cognitive Radio: Making Software Radios More Personal", IEEE Personal Communications, vol. 6, pp. 13-18, 1999
- [23] Pooja Singh and Mrs. Lalley P.M., "Portable Monopulse Tracking Receiver", Proceedings of 2013 International Conference on Green High Performance Computing, March 14-15, 2013, India.
- [24] Proakis J. G., Digital Communications, 3rd edition. New York: McGraw Hill, 1995.
- [25] Robert W. Stewart, Louise Crockett, "A Low Cost Desktop Software Defined Radio Design Environment Using MATLAB, Simulink and the RTL-SDR.", IEEE Communication Magazine, pp. 64 – 71, 2015,
- [26] Sharma. P.C, Gupta. K, "Analysis and optimized design of single feed circularly polarized microstrip antennas", IEEE Transactions on Antennas and Propagation, Vol. 31, No. 6, pp. 949-955, Nov.1983.
- [27] Yagi. H, "Beam transmission of ultra short waves", Proc. Inst. Radio Engineering , Vol. 16, No. 6, pp. 715-740, Jun. 1928