Implementation of Wireless Communication based on Software Defined Radio

Author: Lei Zhang

Supervisor: César Briso Rodríguez

programa oficial de postgrado en ingeniería de sistemas y servicios para la sociedad de la información

departamento de ingeniería audiovisual y comunicaciones.

E.U.I.T.TELECOMUNICACIÓN

Julio de 2013

Trabajo Fin de Máster	
Título	Implementation of Wireless Communication based on Software Defined Radio
Autor	Lei Zhang
Programa de Postgrado Oficial	MÁSTER EN INGENIERÍA DE SISTEMAS Y SERVICIOS PARA LA SOCIEDAD DE LA INFORMACIÓN
Tutor	César Briso Rodríguez
Tribunal	
Presidente	Rafael Herradón Díez
Secretario	Antonio Mínguez Olivares
Vocal	Florentino Jiménez Muñoz
Fecha de lectura	Madrid, a 18 de Julio de 2013

路漫漫其修遠兮，吾將上下而求索。

———屈原<離騷>

The road ahead is hard and endless,

but my climbing never stop.

—— Qu Yuan <Li Sao>

ACKNOWLEDGEMENTS

My deepest gratitude goes first and foremost to my supervisor, Professor César Briso Rodríguez, for his constant encouragement and guidance which offered me valuable suggestions in the academic studies. Without his patient instruction, insightful criticism and expert guidance, the completion of this thesis would not have been possible. Also, I would like to thank my lab-mates and every friend in Spain, especially Mr. Jean Raphaël Fernández Fernández and Mr. Sergio Pérez Jiménez who gave me valuable experience and selfless help.

I also owe a special debt of gratitude to all the professors in E.U.I.T de Telecomunicación, from whose devoted teaching and enlightening lectures I have benefited a lot and academically prepared for the thesis.

Apart from all above, I would like to extend my heartfelt gratitude to my beloved families, for their permanent help and support through every step not only in this time but also in my whole life.

I love this beautiful country and this excellent university, *¡Gracias a todos!*

The author.

Index

Index of figures

Index of tables

Index of equations

Resumen

Software Defined Radio (SDR) es una tecnología emergente que está creando un impacto revolucionario en la tecnología de radio convencional. Un buen ejemplo de radio software son los sistemas de código abierto llamados GNU Radio que emplean un kit de herramientas de desarrollo de software libre. En este trabajo se ha empleado un kit de desarrollo comercial (Ettus Research) que consiste en un módulo de procesado de señal y un hardaware sencillo. El módulo emplea un software de desarrollo basado en Linux sobre el que se pueden implementar aplicaciones de radio software muy variadas. El hardware de desarrollo consta de un un microprocesador de propósito general, un dispositivo programable (FPGA) y un interfaz de radiofrecuencia que cubre de 50 a 2200MHz. Este hardware se conecta al PC por medio de un interfaz USB de 8Mb/s de velocidad. Sobre la plataforma de Ettus se pueden ejecutar aplicaciones GNU radio que utilizan principalmente lenguaje de programación Python para implementarse. Sin embargo, su módulo de procesado de señal está construido en C + + y emplea un microprocesador con aritmética de coma flotante. Por lo tanto, los desarrolladores pueden rápida y fácilmente construir aplicaciones en tiempo real sistemas de comunicación inalámbrica de alta capacidad. Aunque su función principal no es ser un simulador, si no puesto que hay componentes de hardware RF, Radio GNU sirve de apoyo a la investigación del algoritmo de procesado de señales basado en pre-almacenados y generados por los datos del generador de señal.

En este trabajo fin de master se ha evaluado la plataforma de hardware de DEG (USRP) y el software (GNU Radio). Para ello se han empleado algunas técnicas de modulación básicas en el sistema de comunicación inalámbrica. A partir de los ejemplos proporcionados por GNU Radio, hemos realizado algunos experimentos relacionados, por ejemplo, escaneado del espectro, demodulación de señales de FM empleando siempre el hardware de USRP. Una vez evaluadas aplicaciones sencillas se ha pasado a realizar un cierto grado de mejora y optimización de aplicaciones complejas descritas en la literatura. Se han empleado aplicaciones como la que consiste en la generación de un espectro de OFDM y la simulación y transmisión de de señales de vídeo en tiempo real. Con estos resultados se está ahora en disposición de abordar la elaboración de aplicaciones complejas.

Summary

In current communication systems, there are many new challenges like various competitive standards, the scarcity of frequency resource, etc., especially the development of personal wireless communication systems result the new system update faster than ever before, the conventional hardware-based wireless communication system is difficult to adapt to this situation. The emergence of SDR enabled the third revolution of wireless communication which from hardware to software and build a flexible, reliable, upgradable, reusable, reconfigurable and low cost platform.

The Universal Software Radio Peripheral (USRP) products are commonly used with the GNU Radio software suite to create complex SDR systems. GNU Radio is a toolkit where digital signal processing blocks are written in C++, and connected to each other with Python. This makes it easy to develop more sophisticated signal processing systems, because many blocks already written by others and you can quickly put them together to create a complete system. Although the main function of GNU Radio is not be a simulator, but if there is no RF hardware components,it supports to researching the signal processing algorithm based on pre-stored and generated data by signal generator.

This thesis introduced SDR platform from hardware (USRP) and software(GNU Radio), as well as some basic modulation techniques in wireless communication system. Based on the examples provided by GNU Radio, carried out some related experiments, for example GSM scanning and FM radio station receiving on USRP. And make a certain degree of improvement based on the experience of some investigators to observe OFDM spectrum and simulate real-time video transmission. GNU Radio combine with USRP hardware proved to be a valuable lab platform for implementing complex radio system prototypes in a short time.

1 Introduction

Radio communication in the modern communication system occupies an extremely important position, which is widely used in commercial, meteorology military and civilian fields. Communication system constantly transit from analog systems to digital systems, a number of digital IF receiver appeared in this trend. Despite these receivers can cover multiple bands, but they only work on a single frequency band and mode, function is relatively small, lack of flexibility and scalability. It is still not fully interoperable between different types of stations, unable to meet the modern communications.

Conventional communication system which hardware-based, for the specific purpose, urgently needs to be replaced by a multiband, multimode, programmable, versatile radio system. The Software Defined Radio(SDR) concept put forward in a timely manner to solve these problems.

Software Definition Radio, suggests that it is a wireless communications which use modern software to manipulate and control the traditional "pure hardware circuit". Breaking the development pattern that communication device implementation always depends on hardware[1]. The central idea is: constructs a open, standardized, modular common hardware platform, and the various functions, such as working frequency, modulation and demodulation types, data formats, encryption mode communication protocol software to accomplish, and to **A/D** and **D/A** converter as close to the antenna, in order to develop a high degree of flexibility, openness, a new generation of wireless communication systems[2].

USRP (Universal Software Radio Peripheral) designed to enable ordinary computer can work like high bandwidth software defined radios. Essentially, it acts as digital baseband and IF section in radio communications system[3]. At the same time, there is an open source software named GNU Radio which is one of free software development tool kit and provides signal operation and processing module, it can be implemented software defined radio on a low cost radio frequency (RF) hardware which is easily produced and a general purpose microprocessor. GNU Radio applications are mainly using Python programming language to write. But its core signal processing module is built in C++ on a microprocessor with floating-point arithmetic. Thus, developers can quickly and easily build a real-time, high-capacity wireless communication systems.

1.1. Objectives

1.1.1. General Objective

Learning and understanding the basics of the wireless communication, combined with the applications of software defined radio. Search relevant literature, learning work principle and usage of USRP and GNU Radio by the examples provide by Ettu Research. Carried out relevant test to obtain results for analysis.

1.1.2. Specific Objectives

- Broadly understanding the application of SDR platform.
- Installing correctly and learning the working principle of USRP.
- Setting up GNU Radio to implement the adapted project.
- Scanning the GSM base station nearby and analyze the frequency spectrum.
- Changing the center frequency by GNU Radio to receive different FM radio stations.
- Transmitting the OFDM signal and observe the spectrum, on this basis, designing a simulation to observe and transmit the OFDM signal.
- Creating a simulation for video transmission, add UDP sink block to achieve real-time video broadcasting.

1.2. Structure

This master work is conducted into two phases. The first phase is a literature study about Software Defined Radio, the working principle of GNU Radio and the USRP. The details of the literature study is described in chapter 2 combine with chapter 3. These study are essential to fully understand how to use the approach, set the parameters and finish a application and implemented in chapter 4.

The second phase describes the result of the practical work. In this phase the setting up for GNU Radio and USRP will described in details. an FFT spectrum analysis experiment is proposed to perform GSM scanning system. Then a stereo wideband FM receiver is implemented to receive different FM radio stations. The approach to transmitting OFDM signal described in the 3rd section of this phase. And base on the python code, designed a OFDM signal observing simulation. The last implementation is a design for real-time digital video broadcasting, the explanation and test result will described step by step.

Finally, The thesis finishes with the considerations, conclusions and future work. USRP B100 datasheet and the created source codes can be found in the appendix.

1.3. Related Work

Chapman E, El Choueiry R, Jackson J, et al presented the design, development, and results of two major aspects of a Software Defined Radio. They are the RF unit, which is the means of transmitting wireless data, and the Forward Error Correction (FEC) coding which supplies a coding gain to the system, and a simulation of the entire SDR in Multi-Disciplinary Engineering Design Conference[4].

David A. Scaperoth defined cognitive radio merges artificial intelligence and software defined radios. This research create a method for communicating between these two levels and showed a genetic algorithm approach to perform intelligent radio adaptation, using the GNU radio platform as an example[5].

Although conventional cryptographic security mechanisms are essential to the overall problem of securing wireless networks, these techniques do not directly leverage the unique properties of the wireless domain to address security threats[6]. Zang Li, Wenyuan Xu, Rob Miller and Wade established new forms of authentication and confidentiality that operate at the physical layer and can be used to facilitate cross-layer security paradigms. Their work showed that GNU Radio combine with USRP will be a good choice to perform prototyping of wireless protocols.

Kalen Watermeyer aimed to create an ADC-based system that could capture samples for processing in software on a host PC, while providing a framework for

functionality enhancements through system extensions in [7]. his demonstrates that GNU Radio software is not on only combine with USRP, but also with other different kind of hardware peripherals.

In[8], Mate A, Lee K H, Lu I T use Software Defined Radio implement spectrum sensing in real environment and verify two present algorithms based on the time-covariance matrix. In their work, GNU Radio and USRP were combined as a powerful tool to do spectrum sensing without any prior knowledge of primary signal and noise power. Spectrum sensing plays a paramount role in cognitive radio, Sarijari M A and his colleagues also made an analysis study on energy detection sensing based on GNU radio and USRP[9].

OFDM has developed into a popular scheme for wideband digital communication. [10] implemented in GNU Radio framework, enables interference-free coexistence of two OFDM-based systems within a common frequency band with optimally configured transmission parameters for given system constraints to meet the optimal utilization of radio resources in multi-carrier based systems. Braun M, Müller M and Fuhr M presented a measurement testbed for OFDM radar which uses USRP as a front-end to perform measurements for car-to-car or car-to-infrastructure applications. And showed how signals parametrized according to the IEEE 802.11a/p standards can be enhanced by radar functions. By using the constellation expansion technique for OFDM based systems in DSA networks. the available bandwidth can be used efficiently while at the same time keeping the interference power level below a certain threshold[12], Selim A and Doyle L also did some very valuable work.

2 Background

This chapter describes in detail about the relevant background knowledge of the master work. First of all, section 2.1 will explain the basic concepts and ideas behind Software Defined Radio platforms. Afterwards, As an excellent platform and the key equipment of this thesis, Universal Software Radio Peripheral (USRP) conducted in-depth introduction in section 2.2 and important applications in radiocommunication area expounded in section 2.3. The last section will explain the different basic background knowledge about GSMK, OFDM as well as UDP. And the importance of these techniques which cause to be widely implemented in the current communication system.

2.1. Software Defined Radio

J.Mitola in 1992 first proposed the concept of software defined radio[13], since the technology has been widespread concern in the industry and research. Its original purpose was to create a device capable of emulating multiple radios working at different frequencies. Nowadays, it has evolved and is still doing so into a tool which has a much broader use.

In SDR, signal will be processed in digital mode instead in analog mode as in the conventional radio. The digitization work will be done by a device called the Analog to Digital Converter (ADC). Figure 2.1.1 shows the concept of Software Defined Radio. It shows that the ADC processor is taking place after the RF Front-End circuit. RF Front-End is used to down convert the signal to the lower frequency called an Intermediate Frequency (IF); this is necessary due to the limitation of the speed of current Commercial of The Shelf (COTS) ADC. The ADC will digitize signal and pass it to the baseband processor for further processes; demodulation, channel coding, source coding and etc. Therefore compared with the conventional radio equipment, the SDR equipment is easier to reconfigure, which can flexibility for multi-format switch and adapt technology development and evolution.

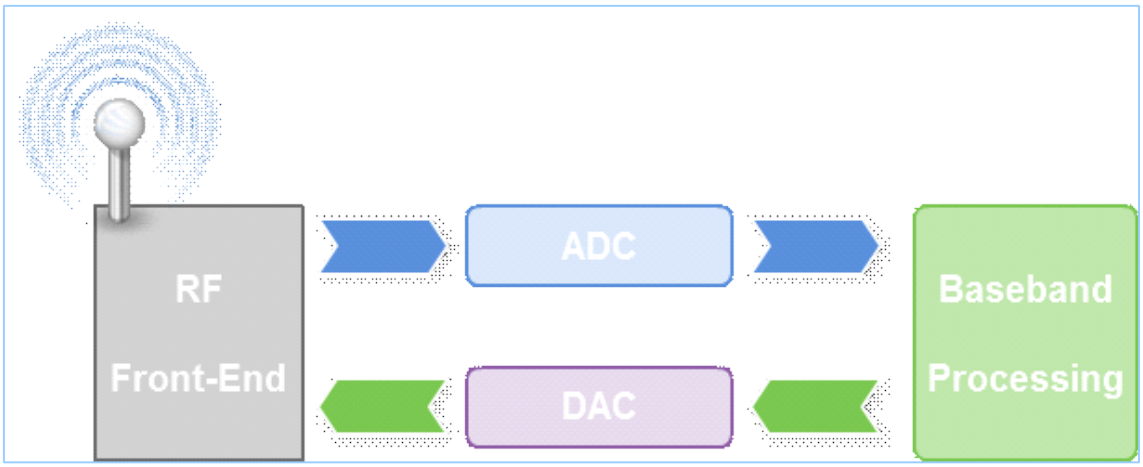


Figure2.1.1. Software Defined Radio System Block Diagram

Software Defined Radio broadly divided into three categories:

- The device integrated with a variety of different formats. For example the GSM-CDMA dual-mode cellphone on the market. Obviously, this approach can only be switched between several preset formats, to increase support for the new standard would mean more integrated circuit, reconfiguration capability is very limited.

- Based on field-programmable gate array (FPGA) and digital signal processor (DSP). Such programmable hardware reconfiguration capability has been greatly improved. But for FPGA VHDL, Verilog and other languages as well as 418 assembly language are for vendor-specific products, making the software in this way too dependent on specific hardware, portability is poor. In addition, for the majority of technical people, FPGA and DSP development threshold is still high and the development process is relatively cumbersome.

- For the above two types of defects, the third category of software radio equipment using common hardware, for example commercial servers, ordinary PC and embedded systems as a signal processing software platform. It has the following advantages: pure software signal processing with great flexibility; adopt a common high-level languages (such as C / C++) for software development, scalability and portability, short development cycle; based on a common hardware platform, lower

cost, and enjoy the advancement of computer technology brings various advantages for example CPU processing power continues to improve as well as software technology, etc.

Although based on a common hardware platform, software-defined radio has many advantages, but for the efficiency in processing speed, size and power consumption as well as real-time aspect, the common hardware platform is still worse than that dedicated FPGA and DSP hardware at the present stage. So now the second category of software radio is still the mainstream, but because of microelectronics technology and the rapid development of computer technology, software radio will increasingly favor a common hardware platform.

Most recently, the GNU Radio using primarily the Universal Software Radio Peripheral (USRP) through a USB 2.0 interface, an FPGA, and a RF front-end high-speed set of analog to digital and digital to analog converters, combined with reconfigurable free software. Its sampling and synthesis bandwidth is a thousand times that of PC sound cards, which enables wideband operation.

The HPSDR (High Performance Software Defined Radio) project uses a 16-bit 135 MSPS analog-to-digital converter that provides performance over the range 0 to55 MHz comparable to that of a conventional analogue HF radio. The receiver will also operate in the VHF and UHF range using either mixer image or alias responses. Interface to a PC is provided by a USB 2.0 interface though Ethernet could be used as well. The project is modular and comprises a back plane onto which other boards plug in. This allows experimentation with new techniques and devices without the need to replace the entire set of boards. An exciter provides 1/2 W of RF over the same range or into the VHF and UHF range using image or alias outputs[14].

WebSDR[15] is a project initiated by Pieter-Tjerk de Boer providing access via browser to multiple SDR receivers worldwide covering the complete shortwave spectrum. Recently he has analyzed Chirp Transmitter signals using the coupled system of receivers[16].

2.2. Universal Software Radio Peripheral

The current wireless communication systems typically use high frequencies to

communicate, Down converting must use to sample and transfer those high frequencies the SDR implementation. Universal Software Radio Peripheral (USRP) is such a family of hardware by computer hosted. Simultaneously, a flexible and low-cost platform for SDR developed by Matt Ettus [17] used to create the connection between the RF-world (radio frequency) and the PC. USRP composed by USRP motherboard, along with a variety of daughterboard and the corresponding antenna. Figure 2.2.1 shows a USRP mother board combined with four daughter boards which means the individual blocks of a typical USRP product consists of two parts: one motherboard with a high-speed signal processing FPGA, and one or more daughterboards which cover different frequency ranges and can be swapped. Combine them to achieve the bit stream data from the antenna to the host computer as a receiver, or from the host computer to the antenna as a transmitter. In a variety of daughterboards, USRP series covers the entire range from DC to 5.9GHz, which include all frequencies from AM radio to over IEEE802.11 standard. The USRP is constructed out of the different components, which are described in detailed below:

- USB2.0 Controller
- ADC (Analog to Digital Converter)
- DAC (Digital to Analog Converter)
- PGA(Programmable Gain Amplifier)
- Daughterboards
- FPGA (Field Programmable Gate Array)

The specific configuration of modules above and their workflow is described in Figure 2.2.2. And the specifications of the USRP in first generation were listed in table 2.1.

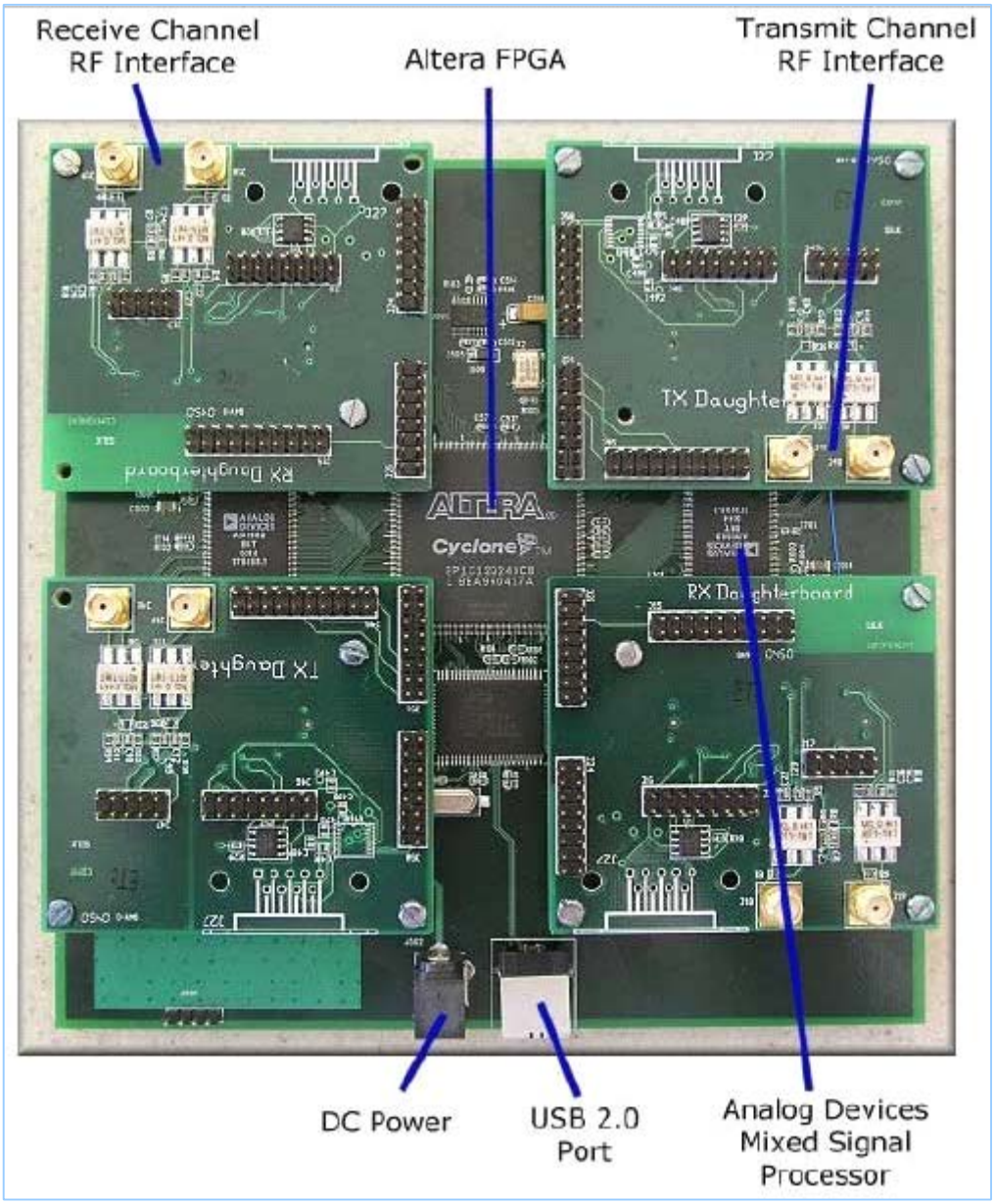


Figure 2.2.1:USRP motherboard and four daughter boards [18]

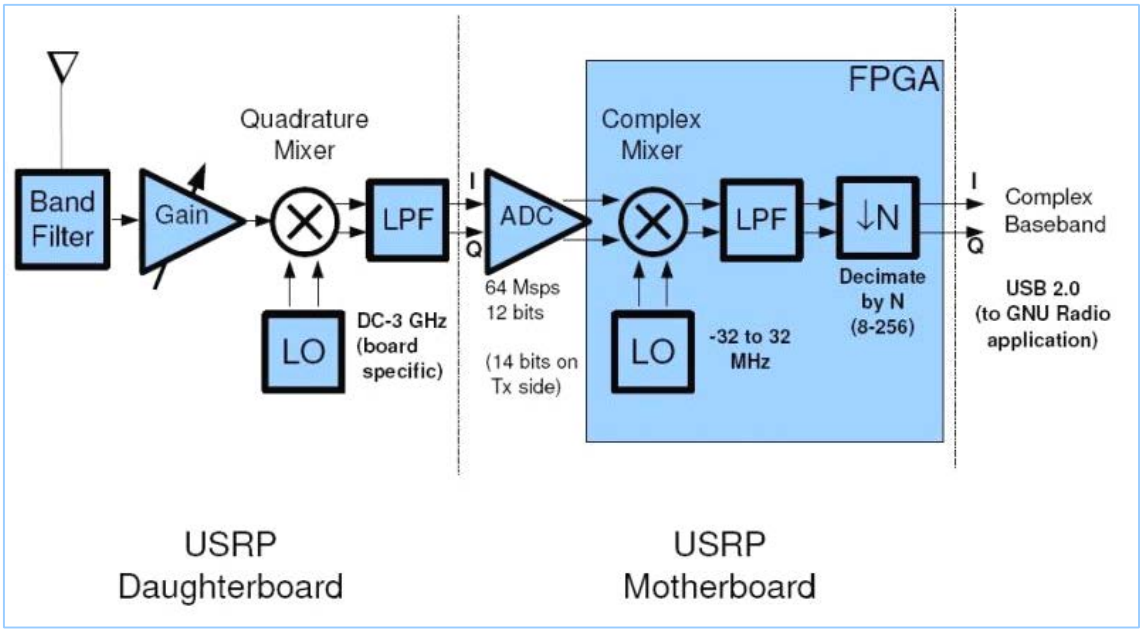


Figure 2.2.2: Universal Software Radio Peripheral block diagram[19]

Supported OS	Input	Output	Auxiliary I/Q
Linux Mac OS X Windows XP, Windows 2000, FreeBS D, NetBSD	**Number of input channels:** 4 (or 2 I/Q pairs) **Sample rate**: 64 Ms/s **Resolution:** 12 bits **SFDR:** 85 dB	**Number of output channels:** 4 (or 2 I/Q pairs) **Sample rate:** 128 Ms/s **Resolution:** 14 bits **SFDR:** 83 dB	**High -speed digital I/O:** 64 bits **Analog input:** 8 channels **Analog output:** 8 channels

Table 2.1 USRP Specifications [20]

2.2.1. The motherboard

The main function of the motherboard are IF sampling and the conversion between IF signal and baseband signal.

As figure 2.2.2 shows there are four slots on the motherboard which are used to connect the daughter boards with the mother board. Two of the four slots, labeled TXA and TXB, are meant for the transmitter daughterboard while another two, RXA and

RXB, are for the receiver daughterboard.

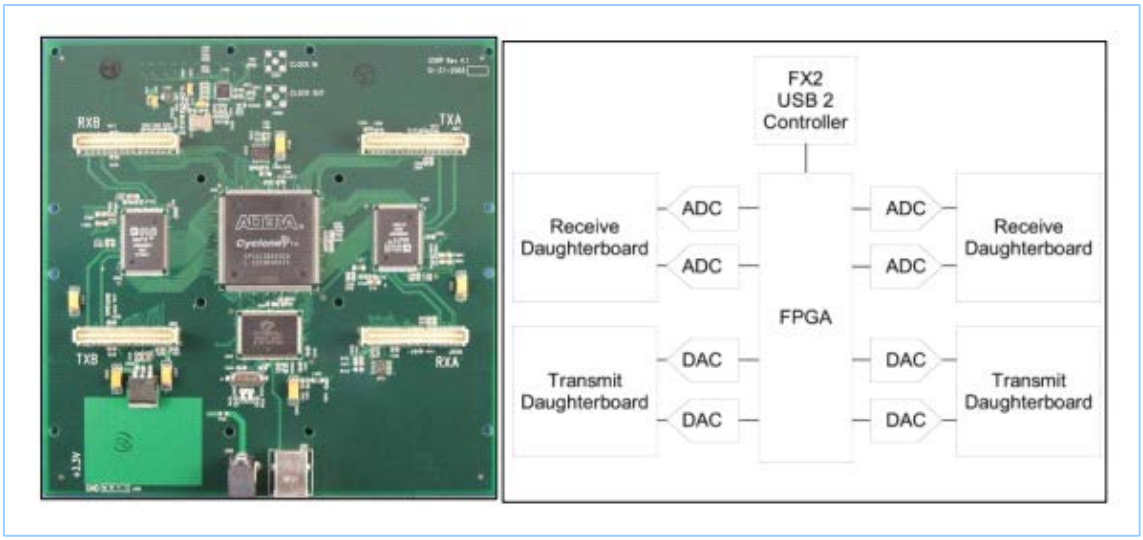


Figure 2.2.3: USRP motherboard Architecture [20]

The motherboard consists of four 12-bit Analog to Digital Converter (ADC) with sampling rate up to 64Msps(samples per second), four 14-bit Digital to Analog Converter (DAC) with speed up to 128Msps, two Digital up Converter (DUC) to up convert the baseband signal to 128Msps before translating them to the selected output frequency, a programmable USB 2.0 controller for communication between USRP and GNU Radio or other software which supported USRP and an FPGA for implementing four Digital Down Converter (DDC) which described by the figure below and high rate signal processing.

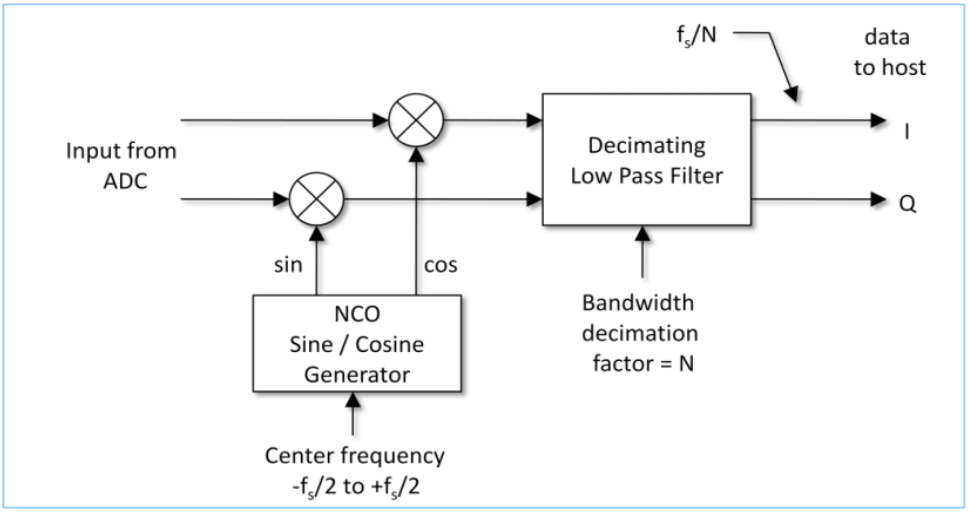


Figure 2.2.4: Digital Down Converter Block Diagram [21]

The digitized samples from ADC are mixed down to the desired IF by being multiplied with a sine and cosine function respectively resulting in the I and Q path. The frequency is generated with a numerically-controlled oscillator (NCO) which synthesizes a discrete-time, discrete amplitude waveform within the FPGA. Via the used NCO, very rapid frequency hopping is feasible. Afterwards a decimation of the sampling rate is performed by an arbitrary decimation factor N. The sampling rate (fs) divided by N results in the output sample rate, sent to host. In transmit path, the same procedure is done by using digital up converters (DUC) and digital analog converters (DAC).

The FPGA also supports time dependent applications which e.g. use TDMA. A free running internal counter allows incoming samples to be sent in strictly definable time stamps.

2.2.2. The daughterboard

The daughterboard is acting as the RF front-end of the SDR. In most of daughterboards, the signal is already filtered, amplified and tuned to a baseband frequency dependent on the boards IF bandwidth and local oscillator frequency. There are also so called Basic Rx/Tx boards with no frequency conversion or filtering. They only provide a direct RF connection to the motherboard. The details for most of available daughterboards are listed in table 2.2.

Identifier	Frequency range	Area of application
Transceiver		
WBX	50-2200 MHz	Broadcast TV; GSM; WSN
SBX	400-4400 MHz	WiFi, WiMax
RFX900	750-1050 MHz	GSM (Low Band)
RFX1200	1150-1450 MHz	GPS
RFX1800	1.5-2.1 GHz	DECT, GSM (High Band)
RFX2400	2.3-2.9 GHz	WLAN, Bluetooth
XCVR 2450	2.4 GHz and 5 GHz	WLAN
Transmitter & Receiver		
Basic TX, Basic RX	1-250 MHz	Misc baseband operations
TVRX Receiver	50-860 MHz	VHF, DAB
DBSRX2 Receiver	800-2300 MHz	Cellular and PCS,DECT

Table 2.2 USRP daughterboard list

2.2.3. Relative Projects

In this section, Two practical and interesting projects will be introduced.

2.2.3.1. Open BTS Project

It's been a century since the growth of telecom industry, still telecom sector is in the middle of a communication revolution as wireless technologies radically transform the industry. The world of telecommunications has been characterized by a remarkable growth like many other sectors which experienced rapid growth and high technological development. Implementation of GSM (Global System for Mobile Communications) system was a big step towards improving communication, traditionally it has an expensive hardware.

The most common example of a cellular network is a mobile phone (cell phone) network. A mobile phone is a portabletelephone which receives or makes calls through a cell site (base station), or transmitting tower. Radio waves are used to transfer signals to and from the cell phone.

Modern mobile phone networks use cells because radio frequencies are a limited, shared resource. Cell-sites and handsets change frequency under computer control and use low power transmitters so that a limited number of radio frequencies can be simultaneously used by many callers with less interference.

A cellular network is used by the mobile phone operator to achieve both coverage and capacity for their subscribers. Large geographic areas are split into smaller cells to avoid line-of-sight signal loss and to support a large number of active phones in that area. All of the cell sites are connected to telephone exchanges (or switches), which in turn connect to the public telephone network.

In cities, each cell site may have a range of up to approximately ½ mile, while in rural areas, the range could be as much as 5 miles. It is possible that in clear open areas, a user may receive signals from a cell site 25 miles away.

Since almost all mobile phones use cellular technology, including GSM, CDMA, and AMPS (analog), the term "cell phone" is in some regions, notably the US, used interchangeably with "mobile phone". However, satellite phones are mobile phones that do not communicate directly with a ground-based cellular tower, but may do so indirectly by way of a satellite.

A simple view of the cellular mobile-radio network consists of the following:

- A network of radio base stations forming the base station subsystem.
- The core circuit switched network for handling voice calls and text.
- A packet switched network for handling mobile data.
- The public switched telephone network to connect subscribers to the wider telephony network.

This network is the foundation of the GSM system network. There are many functions that are performed by this network in order to make sure customers get the desired service including mobility management, registration, call set up, and handover[22].

A normal GSM network working is as follows.The end point of the system will be BTS (Base Transceiver Station) which send radio frequency signal to and from mobile devices or a modem.The BTS comes under BSC(Base station Controller) with makes the communication between there radio signals with MSC/VLR. The MSC/VLR is responsible to authenticate the user against the database (HLR – Home Location Register, AuC - Authentication Center), call setup and call routing. A typical GSM network diagram is shown below.

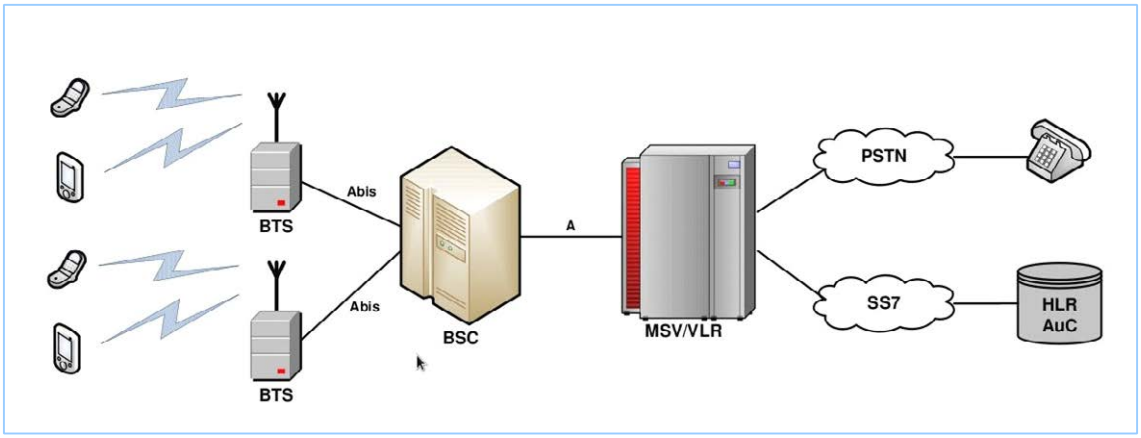


Figure 2.2.5 GSM network diagram

An upcoming technology named Open Base Transceiver Station (BTS) which is a new kind of cellular network that can be installed and operated at a very low cost compare to current GSM technology. Recent development in Software Defined Radio (SDR) in signal processing has made it much more economical to implement such a system where most of the back end hardware can be substituted with real-time software applications. With the development of GNU radio and Universal Software Radio Peripheral (USRP) this network can be accessed by the normal GSM handsets which are available in the market.

Figure 2.2.6 described the composition of the radio frequency part which Open BTS used. the key component is the Universal Software Radio Peripheral which is also the core of Open BTS.

Conventional radio signal processing is essentially complete by the pure hardware device, but the realization of USRP is send the complex signal to the PC software to processing, including modulation and demodulation of the signal and line switching. The most basic processing for radio frequency signals, such as a digital signal conversion, interpolation and sampling, to the FPGA on USRP accomplished through a USB to link PC.

Put such advanced features to the software, and put the underlying processing practices to the hardware ensure the performance of the system and also convenient for system expansion.

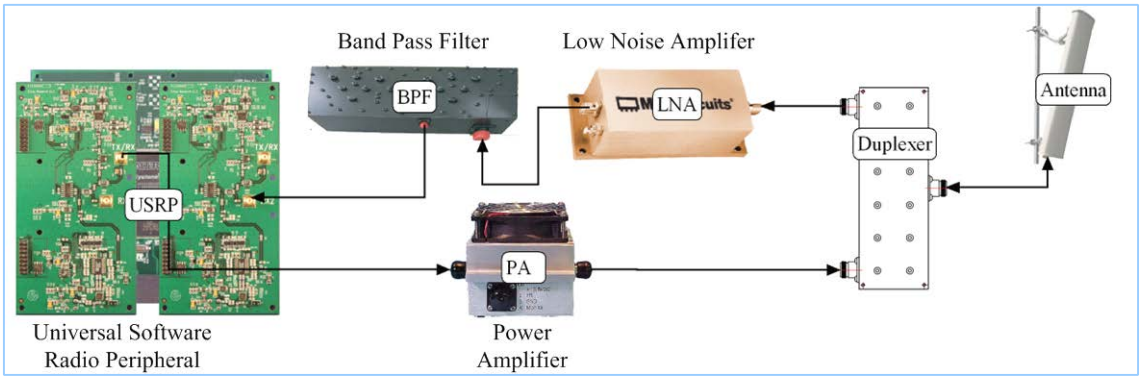


Figure 2.2.6 Open BTS RF component[23]

Open base transceiver station (BTS) is a Unix application that uses a software defined radio platform(like the USRP) to present a GSM (Global System for Mobile Communications) air interface ("UM") to standard GSM handsets and uses a SIP softs witch or PBX to connect calls. (It might even say that Open BTS is a simplified form of IMS that works with 2G feature-phone handsets). The combination of the global-standard GSM air interface with low-cost VoIP backhaul forms the basis of a new type of cellular network that could be deployed and operated at substantially lower cost than existing technologies (for example, commercial carrier BTS systems) in many applications, especially rural cellular deployments and private cellular networks in remote areas[23].

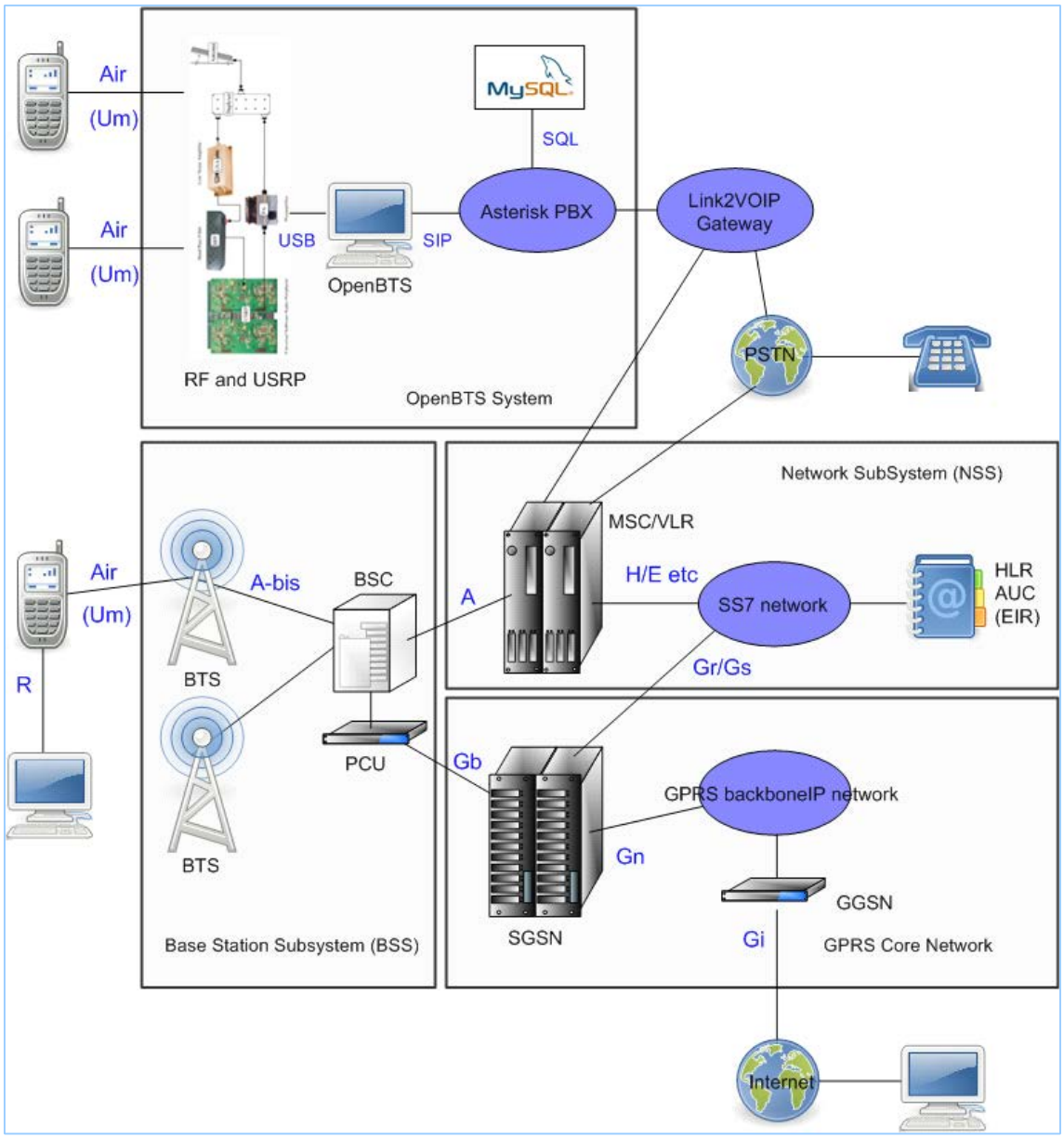


Figure 2.2.7 Mobile phone network system

An entire mobile phone network system which contained Open BTS project shows above:

Open BTS, acts as a mobile phone base station (BTS) and base station controllers (BSC), provide basic functions of modulation, demodulation and assigning channel frequency for each communication.

Asterisk, acts as mobile switching center (MSC) and telephone switching center. If there is a internal communication in the Open BTS users network, the Asterisk will

responsible for establishing communication links directly in the internal network; if there is a phone call to the external network, the Asterisk will connected PSTN networks via VoIP gateway.

MySQL, responsible for mobile phone users account management, recording call information and data. It is also responsible for the storage functions for HLR (Home Location Register) and VLR (Visitor Location Register) in the traditional mobile network.

Antenna, RF hardware and USRP, via a USB port to connect to a PC, running free software, access to Internet, can form a complete mobile phone networks.

2.2.3.2. Gqrx SDR Receiver

Gqrx SDR receiver is designed and implemented by Alexandru Csete OZ9AEC which is a software defined radio receiver for Funcube Dongle (FCD), RTL2832U-based DVB-T devices (RTL-SDR), and Universal Software Radio Peripheral (USRP) and Osmo SDR devices. It is powered by GNU Radio and the Qt GUI toolkit[24].

OsmoSDR can be thought of something in between a FunCube Dongle (only 96kHz bandwidth) and a USRP (much more expensive). For a very cheap (but inaccurate) SDR, you can use the DVB-T USB stick using the RTL2832U chip, as documented in rtl-sdr[25].

The RTL2832U outputs 8-bit I/Q-samples, and the highest theoretically possible sample-rate is 3.2 MS/s, however, the highest sample-rate without lost samples that has been tested so far is 2.4 MS/s. The frequency range is highly dependent of the used tuner, dongles that use the Elonics E4000 offer the widest possible range as the table shows below.

Tuner	Frequency Range
Elonics E4000	52 - 2200 MHz with a gap from 1100 MHz to 1250 MHz (varies)
Rafael Micro R820T	24 - 1766 MHz
Fitipower FC0013	22 - 1100 MHz (FC0013B/C, FC0013G has a separate L-band input, which is unconnected on most sticks)
Fitipower	22 - 948.6 MHz

FC0012	
FCI FC2580	146 - 308 MHz and 438 - 924 MHz (gap in between)

Table 2.3 Frequency range in tuners

Currently, Gqrx offers the following features:

- Automatically detect supported devices attached to the computer.
- Process I/Q data from Funcube Dongle, RTL2832U SDR, USRP and OSmo SDR.
- Change frequency, gain and apply various correction (freqeuncy, I/Q blanace).
- AM, SSB, FM-N and FM-W modulations.
- Variable band pass filter.
- Squelch, noise blankers and AGC.
- FFT plot and waterfall.
- Record audio to file (playback is planned).
- Spectrum analyzer mode where all signal processing is disabled.

For example, in this project Csete uses the Funcube Dongle with an arrow antenna and the hardware(for example USRP) supported by Gqrx received automatic picture transmissions (APT) from NOAA weather satellites using Gqrx SDR, record them to a WAV file, and finally decode the images using the free and open source Atpdec decoder there is a very nice results show below[24].

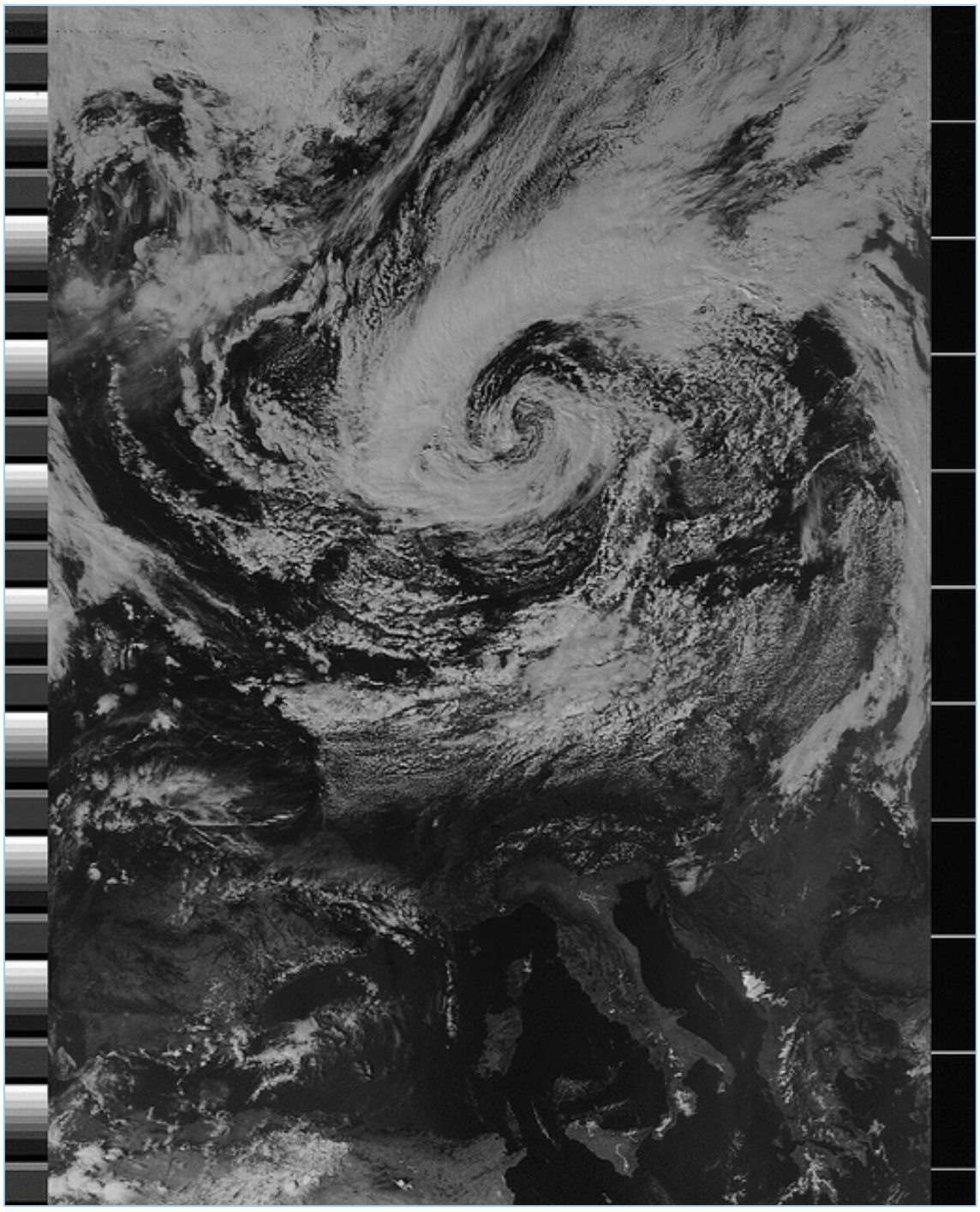


Figure 2.2.8 NOAA-18 APT image received with Gqrx and Funcube Dongle[24]

2.3. Radio Basic

Two widely applied modulation techniques and a useful internet protocol which associated with this thesis will be introduced in this section as background knowledge and aim to learn to develop complex wireless applications.

2.3.1. GMSK

GMSK (Gaussian Filtered Minimum Shift Keying) is a digital modulation method that developed on the basis of MSK (Minimum Shift Keying) modulation, which is a form of modulation used in a variety of digital radio communications systems. It has advantages of being able to carry digital modulation while still using the spectrum efficiently. One of the problems with other forms of phase shift keying is that the sidebands extend outwards from the main carrier and these can cause interference to other radio communications systems using nearby channels. In GMSK, because the Gauss pre-modulation filtering for digital signal before the modulation, the modulated signal in the zero crossing is not only continuous phase, but also smoothing filter, so the spectrum of the GSMK modulated signal compact and has good error characteristics, has been used in a number of radio communications applications. Possibly the most widely used is the GSM (Global System for Mobile communication) cellular technology which is used worldwide and has well over 3 billion subscribers.

MSK and also GMSK modulation are what is known as a continuous phase scheme. Here there are no phase discontinuities because the frequency changes occur at the carrier zero crossing points. This arises as a result of the unique factor of MSK that the frequency difference between the logical one and logical zero states is always equal to half the data rate. This can be expressed in terms of the modulation index, and it is always equal to 0.5.

A plot of the spectrum of an MSK signal shows sidebands extending well beyond a bandwidth equal to the data rate. This can be reduced by passing the modulating signal through a low pass filter prior to applying it to the carrier. The requirements for the filter are that it should have a sharp cut-off, narrow bandwidth and its impulse response should show no overshoot. The ideal filter is known as a Gaussian filter which has a Gaussian shaped response to an impulse and no ringing. In this way the basic MSK signal is converted to GMSK modulation.

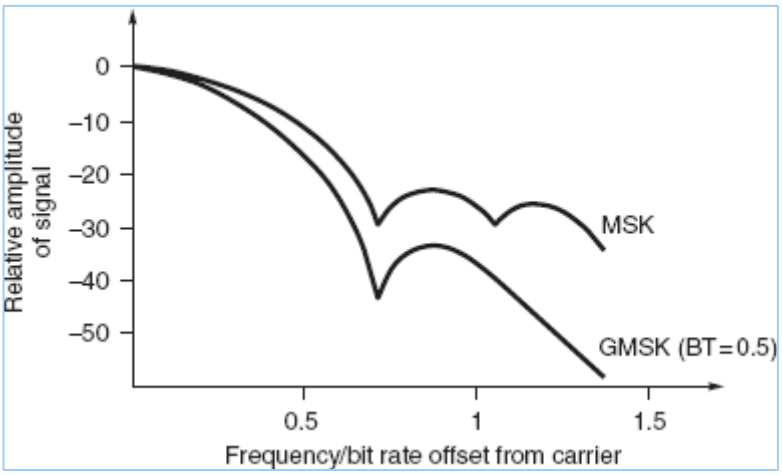


Figure 2.3.1 Spectral density of MSK and GMSK signals

There are two main ways in which GMSK modulation can be generated. The most obvious way is to filter the modulating signal using a Gaussian filter and then apply this to a frequency modulator where the modulation index is set to 0.5. This method is very simple and straightforward but it has the drawback that the modulation index must exactly equal 0.5. In practice this analogue method is not suitable because component tolerances drift and cannot be set exactly.

The second method is more widely used which described in figure 2.3.1. Here what is known as a quadrature modulator is used. The term quadrature means that the phase of a signal is in quadrature or 90 degrees to another one. The quadrature modulator uses one signal that is said to be in-phase and another that is in quadrature to this. In view of the in-phase and quadrature elements this type of modulator is often said to be an I-Q modulator. Using this type of modulator the modulation index can be maintained at exactly 0.5 without the need for any settings or adjustments. This makes it much easier to use, and capable of providing the required level of performance without the need for adjustments. For demodulation the technique can be used in reverse.

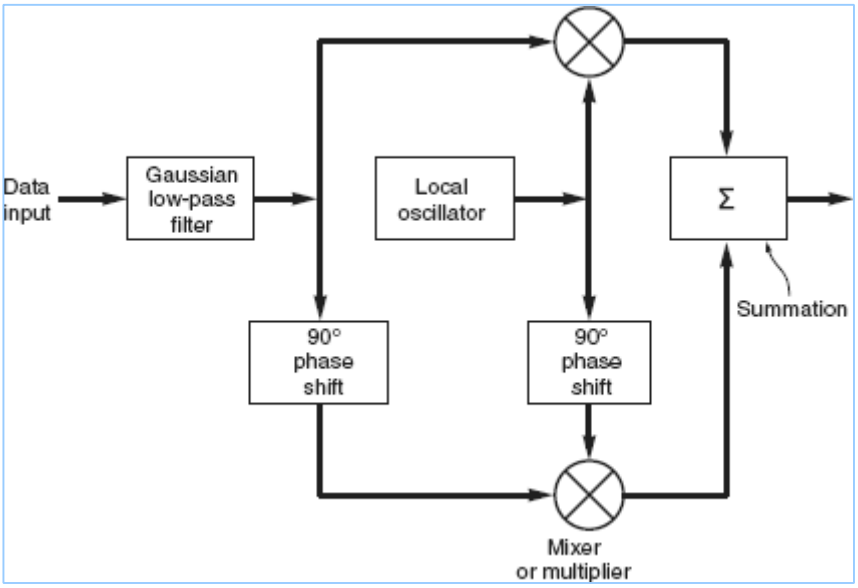


Figure 2.3.2 Block diagram of I-Q modulator for GMSK

There are several advantages to the use of GMSK modulation for a radio communications system. One is obviously the improved spectral efficiency when compared to other phase shift keyed modes.

A further advantage of GMSK is that it can be amplified by a non-linear amplifier and remain undistorted This is because there are no elements of the signal that are carried as amplitude variations. This advantage is of particular importance when using small portable transmitters, such as those required by cellular technology. Non-linear amplifiers are more efficient in terms of the DC power input from the power rails that they convert into a radio frequency signal. This means that the power consumption for a given output is much less, and this results in lower levels of battery consumption; a very important factor for cell phones.

A further advantage of GMSK modulation again arises from the fact that none of the information is carried as amplitude variations. This means that is immune to amplitude variations and therefore more resilient to noise, than some other forms of modulation, because most noise is mainly amplitude based[26].

2.3.2. OFDM

In recent years, OFDM (Orthogonal Frequency Division Multiplexing) technique

has been replaced the single-carrier spread spectrum technique (such as CDMA) in the new generation broadband wireless communication system.

OFDM is a multi-carrier modulation scheme, by reducing and eliminating the influence of inter-symbol interference to overcome the frequency selective fading in channel. It divides total available bandwidth into a large number of closely-spaced orthogonal sub-carriers and simultaneously transfers signals on these sub-carriers with a low data rate, achieving a total data rate approaching ideal Nyquist data rate. The data is divided into several parallel data streams or channels, each sub-carrier is modulated with a conventional scheme,such as quadrature modulation(QAM) or phase shift keying (PSK) at low symbol rate. maintaining total data rates similar to conventional single-carrier modulation schemes in the same bandwidth. Figure 2.3.3 shows a single sub-channel and 5 sub-carriers OFDM spectrum, at the central frequency of each sub-channel, there is no crosstalk from other sub-channels.

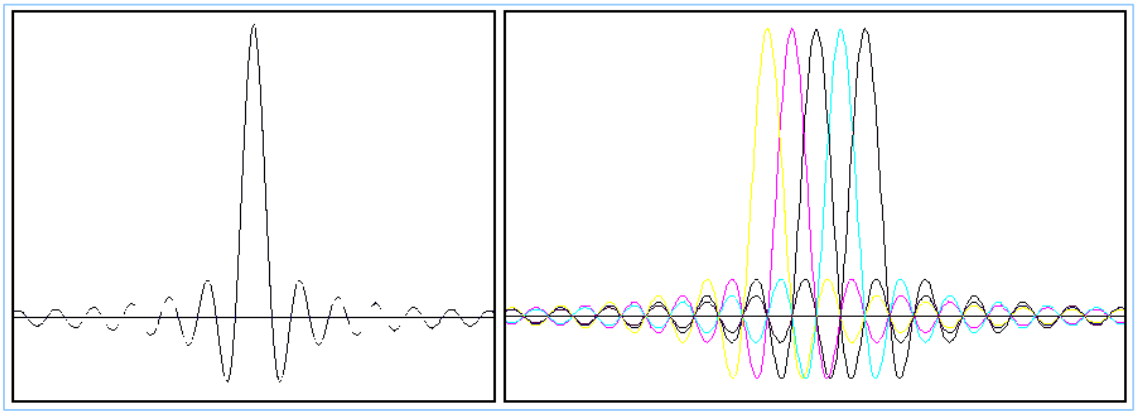


Figure2.3.3 A sub-channel(left) and 5 sub-carriers OFDM spectrum(right)

- **Orthogonality**

Conceptually, OFDM is a specialized FDM, the additional constraint being: all the carrier signals are orthogonal to each other. Orthogonality simplifies recovery of the N data streams, Orthogonal sub-carriers means no inter-carrier-interference (ICI). An orthogonal set of functions is a set with the property that a particular operation performed between any two distinct members of the set yields zero. Vectors are orthogonal if they are at right angles to each other. The dot product of any two distinct vectors is zero. Figure 2.3.4 shows the time domain and frequency domain orthogonality of OFDM.

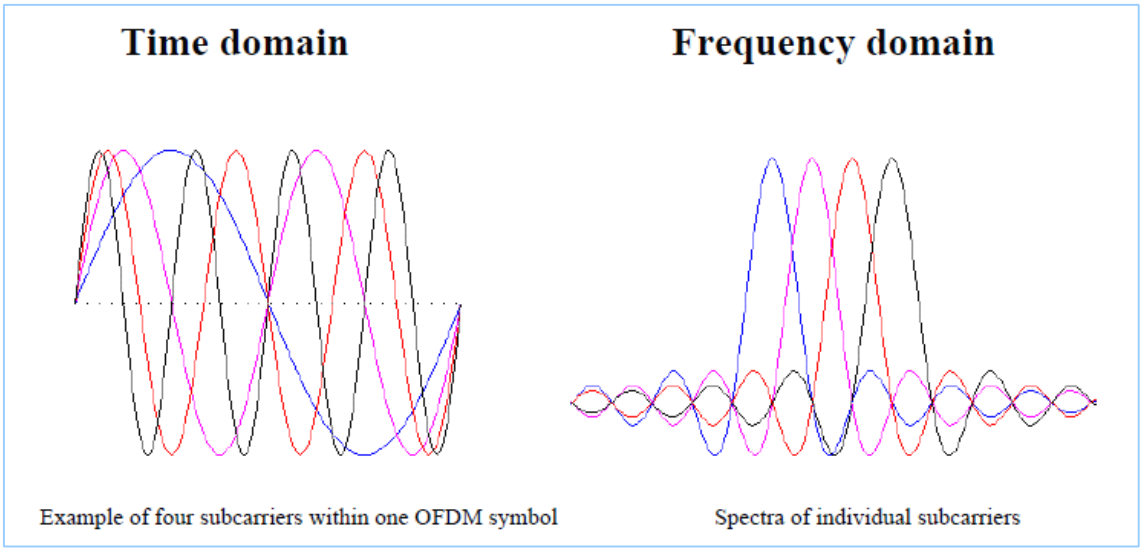


Figure2.3.4 OFDM Orthogonality

Time Domain Orthogonality:

Every sub-carrier has an integer number of cycles within T_{OFDM} . Satisfies precise mathematical definition of orthogonality for complex exponential (and sinusoidal) functions over the interval $[0, T_{OFDM}]$.

Frequency Domain Orthogonality:

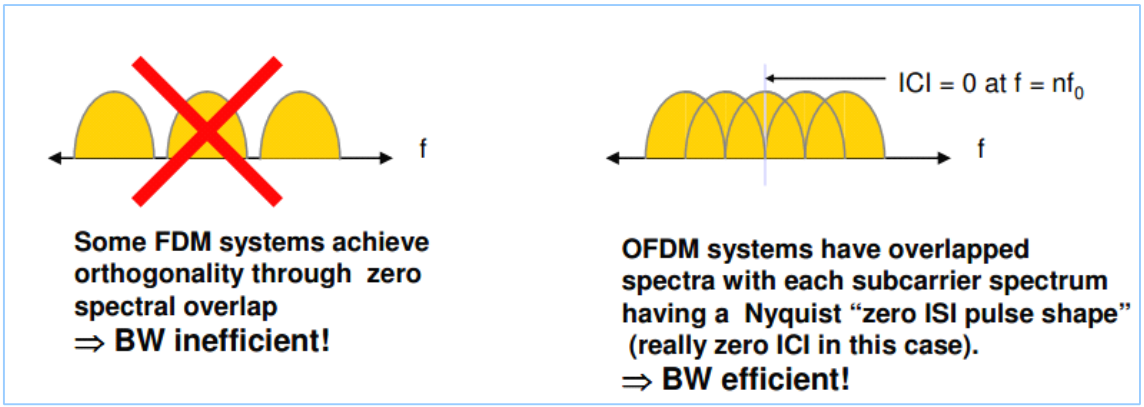


Figure2.3.5 Frequency Domain Orthogonality

If FDM system had been able to use a set of sub-carriers that were orthogonal to each other, and as long as orthogonality is maintained, it is still possible to recover the individual sub-carriers signals, because if the dot product of two deterministic signals is equal to zero, these signals are said to be orthogonal to each other. Orthogonality can also be viewed from the standpoint of stochastic processes. If two

random processes are uncorrelated, then they are orthogonal. Given the random nature of signals in a communications system, this probabilistic view of orthogonality provides an intuitive understanding of the implications of orthogonality in OFDM [27].

- **Modulation**

The idea behind the analog implementation of OFDM can be extended to the digital domain by using the discrete Fourier Transform (DFT) and its counterpart, the inverse discrete Fourier Transform (IDFT). These mathematical operations are widely used for transforming data between the time-domain and frequency-domain. These transforms are interesting from the OFDM perspective because they can be viewed as mapping data onto orthogonal sub-carriers. For example, the IDFT is used to take in frequency-domain data and convert it to time-domain data. In order to perform that operation, the IDFT correlates the frequency-domain input data with its orthogonal basis functions, which are sinusoids at certain frequencies. This correlation is equivalent to mapping the input data onto the sinusoidal basis functions.

In practice, OFDM systems are implemented using a combination of fast Fourier Transform (FFT) and inverse fast Fourier Transform (IFFT) blocks that are mathematically equivalent versions of the DFT and IDFT, respectively, but more efficient to implement.

For example, n denotes the frequency component index, $S(n)$ denotes the original signal on the transmitter side. And the IFFT of the signal $S(n)$ is:

$$s(k) = \frac{1}{N}\sum_{n=0}^{N-1} S(n) e^{j2\pi nk/N}, k = 0,...N-1$$

(1)

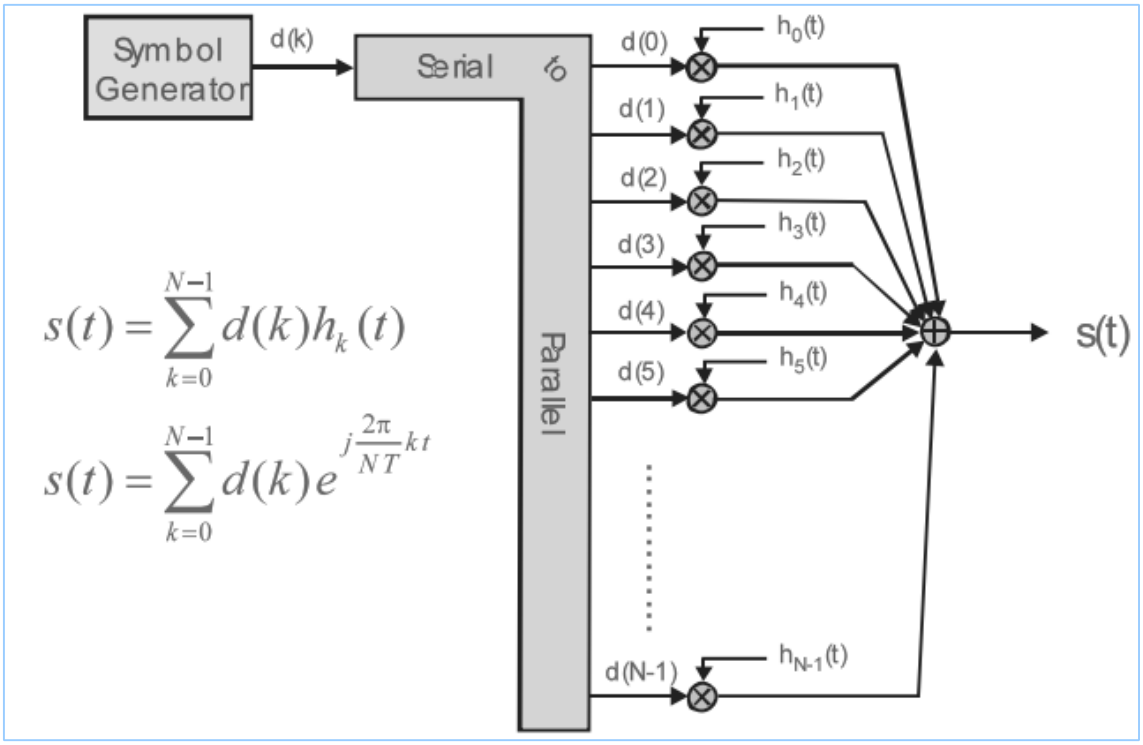


Figure 2.4.3 OFDM modulator[28]

Where N designates the number of frequency components, and s(k) is the resulting sampled signal, which is formed by the sum of the modulated frequency components S(n). To retrieve again the digital frequency components, the inverse equation is:

$$S(n) = \sum_{k=0}^{N-1} s(k) e^{-j2\pi nk/N}, k = 0,...N-1$$

(2)

Which corresponds to the N-point FFT of S(n).

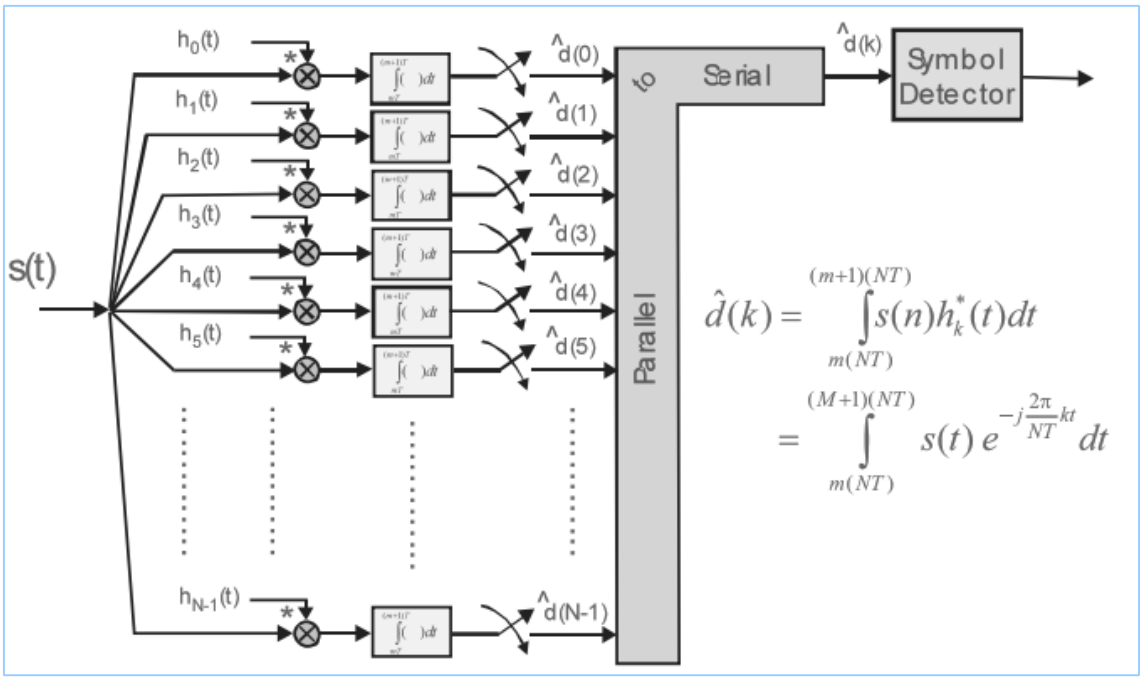


Figure 2.44 OFDM demodulator[27]

- **Guard Interval**

Assume the delay spread of the channel is T_m, then instead of a single carrier with a data rate of R (symbols/second), an OFDM system has N sub-carriers, each with a data rate of R/N (symbols/second) can be used. And because the data rate is reduced by factor of N, the OFDM symbol period is increased by a factor of N, so by choosing an appropriate value for N, the length of OFDM symbol becomes longer than the delay spread of channel. And because of this configuration, the effect of inter-symbol interference will be reduced but no completely eliminated. Guard interval is the technique that OFDM use to cancel the effect of inter-symbol interference. Guard Interval is samples inserted at the beginning of each symbol, and it could be a section of all zero-zero padding. Since it does not contain any useful information, the guard interval would be discarded at the receiver. If the length of the guard interval is properly chosen such that it is longer than the time span of the channel, the OFDM symbol itself will not be distorted. Thus, by discarding the guard interval, the effects of inter-symbol interference are thrown away[27]. The guard interval also eliminates the need for a pulse-shaping filter, and it reduces the sensitivity to time synchronization problems.

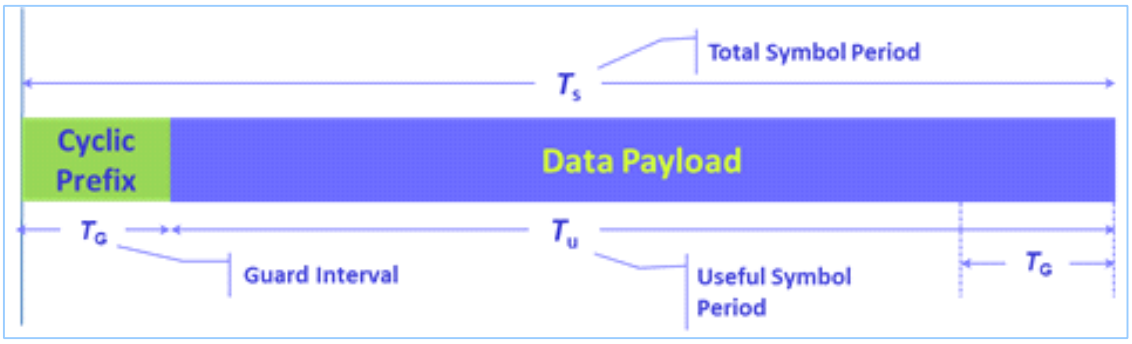


Figure 2.4.5 OFDM Symbol with Cyclic Prefix

The cyclic prefix, which is transmitted during the guard interval, consists of the end of the OFDM symbol copied into the guard interval, and the guard interval is transmitted followed by the OFDM symbol. Figure 2.4.5 shows the structure of the cyclic prefix OFDM. The reason that the guard interval consists of a copy of the end of the OFDM symbol is so that the receiver will integrate over an integer number of sinusoid cycles for each of the multi-paths when it performs OFDM demodulation with the FFT. In some standards such as Ultra wideband, in the interest of transmitted power, cyclic prefix is skipped and nothing is sent during the guard interval. The receiver will then have to mimic the cyclic prefix functionality by copying the end part of the OFDM symbol and adding it to the beginning portion.

Advantages and Disadvantages

OFDM modulation techniques have been used both in wired and wireless systems due to its advantages. Among them, the following features must be mentioned:

- Efficiently Deals With Multi-paths Fading
- Efficiently Deals With Channel Delay Spread
- Enhanced Channel Capacity
- Adaptively Modifies Modulation Density
- Robustness to Narrowband Interference

On the other hand, there are some disadvantages below can not be ignored:

- Complexity

– FFT for modulation, demodulation must be compared to complexity of equalizer.

– Synchronization.

- Overhead

– Cyclic extension increases the length of the symbol for no increase in capacity.

– Pilot tones simplify equalization and tracking for no increase in capacity.

- PAPR

– Depending on the configuration, the PAPR can be ~3dB-6dB worse than a single carrier system.

- Phase noise sensitivity

– The sub-carriers are N-times narrower than a comparable single carrier system.

- Doppler Spread sensitivity

– Synchronization and EQ tracking can be problematic in high doppler environments.

2.3.3. UDP

The User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite (the set of network protocols used for the Internet). With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768.

UDP uses a simple transmission model with a minimum of protocol mechanism[29]. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program. As this is normally IP over unreliable media, there is no guarantee of delivery, ordering or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.

UDP is suitable for purposes where error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system[30]. If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

A number of UDP's attributes make it especially suited for certain applications.

- It is transaction-oriented, suitable for simple query-response protocols such as the Domain Name System or the Network Time Protocol.

- It provides datagrams, suitable for modeling other protocols such as in IP tunneling or Remote Procedure Call and the Network File System.

- It is simple, suitable for bootstrapping or other purposes without a full protocol stack, such as the DHCP and Trivial File Transfer Protocol.

- It is stateless, suitable for very large numbers of clients, such as in streaming media applications for example IPTV.

- The lack of retransmission delays makes it suitable for real-time applications such as Voice over IP, online games, and many protocols built on top of the Real Time Streaming Protocol.

- Works well in unidirectional communication, suitable for broadcast information such as in many kinds of service discovery and shared information such as broadcast time or Routing Information Protocol.

UDP provides application multiplexing (via port numbers) and integrity verification (via checksum) of the header and payload[31]. If transmission reliability is desired, it must be implemented in the user's application.

The figure below is a typical UDP packet structure. A UDP packet consists of an Ethernet Header, an IP Header, a UDP Header, the packet data and an Ethernet Trailer. The size of the packet data can be up to 1500 bytes. Any data over 1500 bytes is typically broken up into multiple packets.

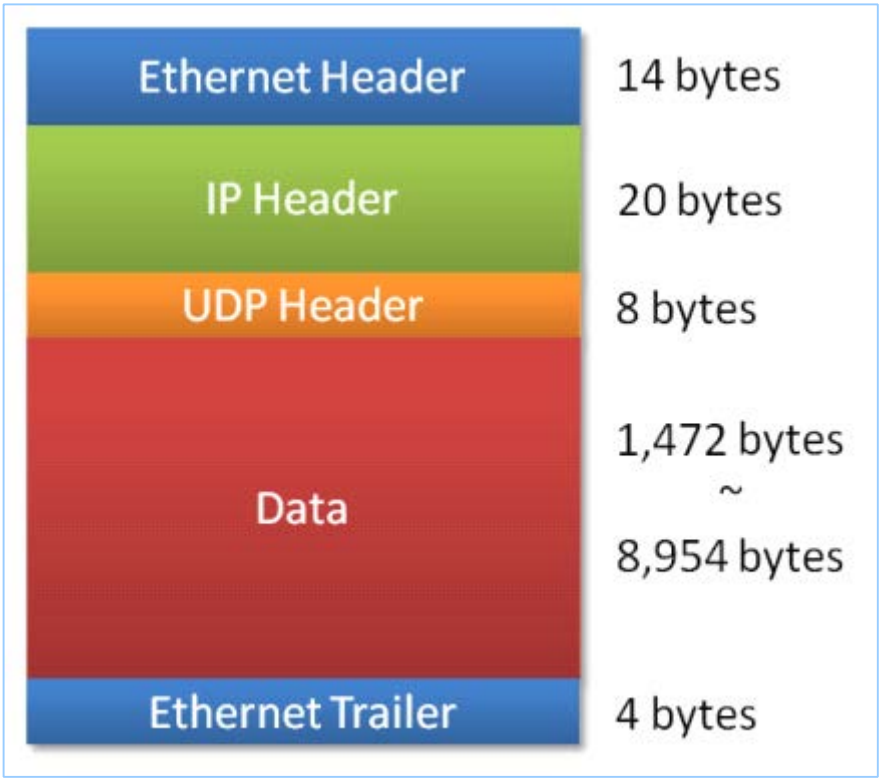


Figure 2.4.6 UDP packet structure

Compare with TCP (Transmission Control Protocol):

UDP	TCP
Unreliable - no concept for acknowledgment, retransmission, or timeout	Reliable - monitors message transmission, tracks data transfer to ensure receipt of all packets
Not ordered - data arrives in order of receipt	Ordered - buffering provisions to ensure correct

	order of data packets
Lightweight - no dedicated end-to-end connection, no congestion control	Heavyweight - dedicated connection, provisions for speed and congestion control
Datagram oriented	Streaming
Light overhead	Heavy overhead
Higher speed	Lower speed

Table 2.4 A comparison of TCP and UDP

UDP is a simpler message-based connectionless protocol, with no dedicated end-to-end connection. Communication is achieved by transmitting information in one direction from source to destination without verifying the readiness or state of the receiver. Because of the lack of reliability, applications using UDP must be tolerant of data loss, errors, or duplication, or be able to assume correct transmission. Such applications generally do not include reliability mechanisms and may even be hindered by them. In these cases, UDP—a much simpler protocol than TCP—can transfer the same amount of data with far less overhead, and can achieve much greater throughput.

UDP is often preferable for real-time systems, since data delay might be more detrimental than occasional packet loss. Streaming media, real-time multiplayer games and voice-over-IP (VoIP) services are examples of applications that often use UDP. In these particular applications, loss of packets is not usually a fatal problem, since the human eye and ear cannot detect most occasional imperfections in a continuous stream of images or sounds. To achieve higher performance, the protocol allows individual packets to be dropped with no retries and UDP packets to be received in a different order than they were sent as dictated by the application. Real-time video and audio streaming protocols are designed to handle occasional lost packets, so only slight degradation in quality occurs, rather than large delays, which would occur if lost packets were retransmitted.

Another environment in which UDP might be preferred over TCP is within a closed network, where there is little chance of data loss or delay. For example, on a board or within an SoC, data transfers from one component to another can be tightly controlled within the application, obviating the need for the reliability features of TCP.

UDP might be a more efficient and equally reliable protocol in such situations. UDP's stateless nature is also useful for servers answering small queries from huge numbers of clients, such as DNS, SNMP and so on.

Both TCP and UDP are widely used IP transfer layer protocols. For applications requiring reliable transfers, TCP is generally preferred, while applications that value throughput more than reliability are best served using UDP. Most TCP/IP stacks provide both protocols, so the application can use whichever transfer protocol is more appropriate, even changing from one to the other as desired. Rather than rely solely on TCP, the network system developer might want to investigate the trade-offs related to use of UDP. It might turn out to be beneficial to sacrifice some reliability in favor of greater throughput[32].

In summary, for applications like streaming video that offer real-time validation, user datagram protocol (UDP) can provide a fast, low-overhead option to TCP.

3 GNU Radio

GNU Radio is a free and open-source software development toolkit that provides signal processing blocks to implement software radios. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in hobbyist, academic and commercial environments to support both wireless communications research and real-world radio systems[33].

After the signal has been processed by the USRP FPGA, the stream of bits finally lows through the USB connection to the host cpu. It is here that the GNU radio framework comes into play. GNU Radio is a free software toolkit licensed under the GPL for implementing software-defined radios. Initially, it was mainly used by radio amateur enthusiasts, but it gained exponential interest from the research world, in an attempt to stay away from closed source firmwares/drivers, and low level of customizability of commercial chips. The GNU radio project was founded by Eric Blossom. It supports natively Linux, and packages are pre-compiled for the major Linux distributions. A port to Windows has been also developed, but it provides limited functionalities. GNU Radio includes a library of signal processing blocks like modulators, demodulators, filters etc. which are used to construct a radio. Essentially it needs the USRP to receive real radio waves or to transmit. You do not necessarily need a USRP. There is also the possibility to use a pre-recorded file as input. A universal SDR structure with the specific software (GNU Radio) and hardware (USRP) is given in figure 3.1.1.

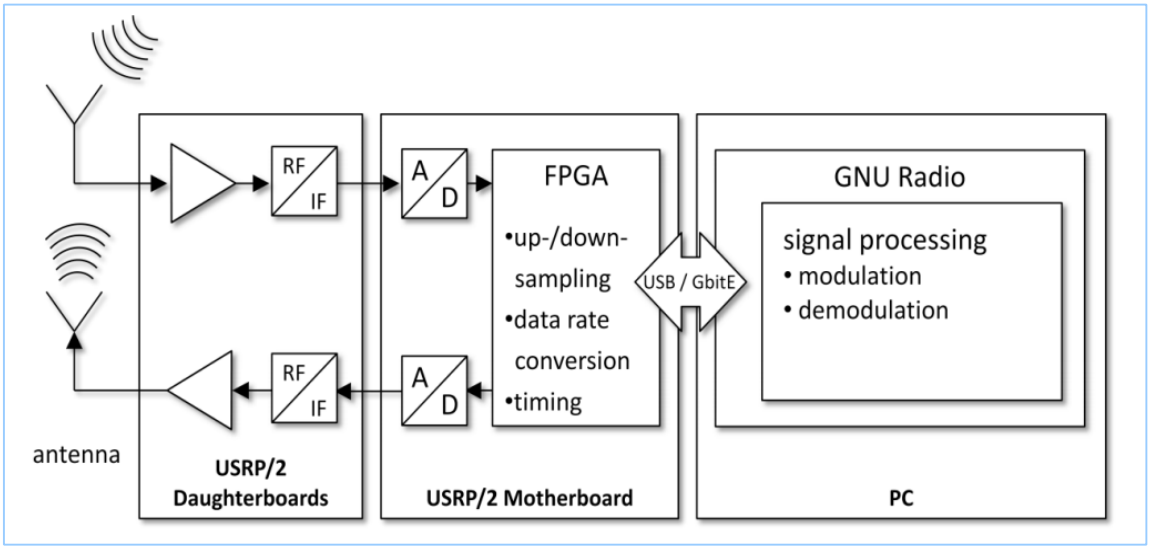


Figure 3.1.1 GNU Radio combined with USRP

GNU Radio's software is organized using a two-tier structure. All the performance-critical signal processing blocks are implemented in C++ [34], while the higher-level organizing, connecting and gluing the signal blocks together is done using Python [35]. There is also a graphical environment available to create a custom radio. This is called GNU Radio Companion (GRC).

3.1. GNU Radio Architecture

The baseline architecture of GNU Radio shows in figure 2.3.2 involves a complex flow-graph that consists of modules and low-level algorithms. Each module or low-level algorithm is structured in C++ and provides basic signal processing functions (ex: Filters, FFT, Channel Coding etc..). They are automatically generated into python modules with the use of python 'wrapper' or interface i.e., SWIG (Simplified Wrapper and Interface Generator) which is used as the interface compiler which allows the integration between C++ and Python language.

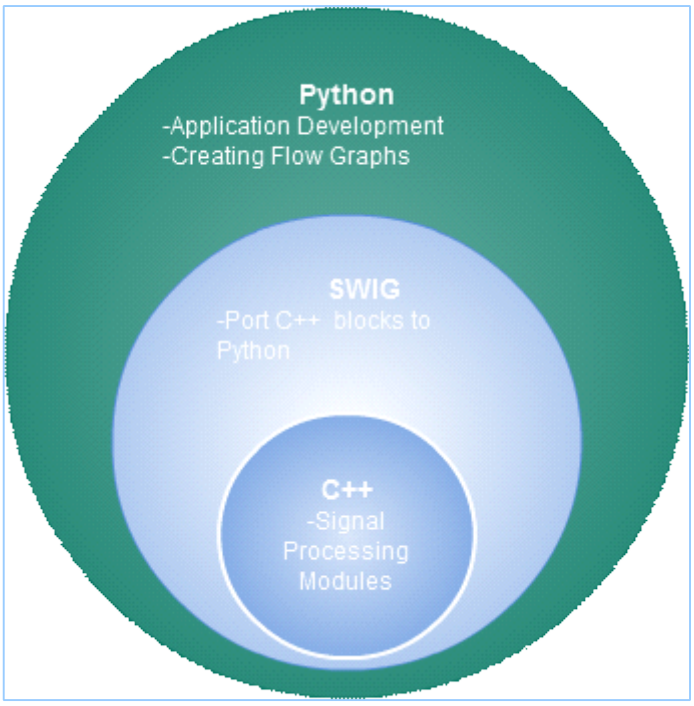


Figure 3.1.2 GNU Radio Software Architecture

So the signal processing blocks are written in C++ while python is used as a scripting language to tie the blocks together to form the flow graph. The generated blocks are used to construct a flow-graph model with the help of python.The application

is built on python program that provides python framework. The python framework is responsible for communication of data through module buffers and creates a simple scheduler that helps to run blocks in a sequential order for single iteration[36] .

The GNU Radio software typically consists of four elements[37]:

- Source: Each flow-graph has a single source. It is the head (start) of the flow-graph. For instance, USRP source or file source are common types of source blocks.

- Sink: Each flow-graph has a single sink. It is the tail (end) of the flow-graph. For instance, USRP sink or file sink are common types of sink blocks.

- Flow-graph: The application is based on a flow-graph. Each flow-graph consists of intermediate blocks along with single source and sink blocks. We can have multiple flow graphs within a single application.

- Scheduler: It is created for each active flow-graph, which is based on steady stream of data flow between the blocks. It is responsible for transferring data through the flow-graph. It monitors each block for sufficient data at I/p and O/p buffers so as to trigger processing function for those blocks.

GNU Radio runs under several operating systems like Linux, Mac OSX, NetBSD. Also a Cygwin porting for Windows exists, but due to the limited hardware control, the full functionality is not guaranteed. Python and C++ are used as main programming languages in GNU Radio as well as the GNU Radio Companion (GRC) which introduced as follow.

Since the GNU Radio framework is the central point of data streams sent towards and

received from USRP, its structure will be illustrated step by step during the whole thesis.

3.2. GNU Radio Companion

GNU Radio Companion (GRC) is a very useful extension which provides a graphical interface that allow sits users to easily create GNU Radio applications. GRC has a list of available modules that can be inserted in the application only need double clicking or dragging directly. These modules can also be configured, and GRC even point out if the configured parameters are incorrect. In addition, the modules can be connected together also very easily. After click the generation or execution, GRC will automatically generated the corresponding python code that will run the application.

Once the installation is complete, type in Terminal:

“gnuradio-companion”

And press “Enter” to run the graphical programming interface of GNU Radio Companion. An untitled GRC window similar to the one below should open.

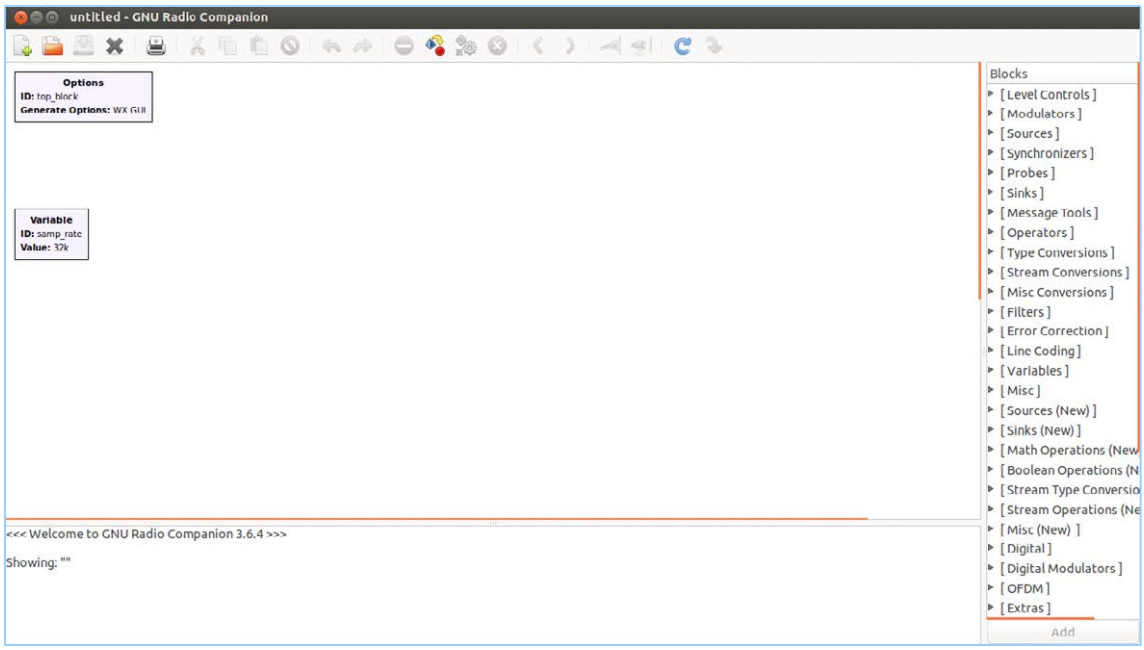


Figure 3.1.3 Untitled GNU Radio companion

The Options block sets some general parameters for the flow graph for example

project title, author and so on. The other block that is present is the Variable block. It is used to set the sample rate for the overall situation. Of course if you want to set more like frequency just add another variable block on the right side of the window which is a list of the blocks that are available. By expanding any of the categories (click on triangle to the left) you can see the blocks available. Explore each of the categories so that you have an idea of what is available.

Official GRC examples are in the folder named "**gnuradio/gnuradio-examples**". Some of the scripts that come with GNU Radio are generated from GRC flow graphs. Just run find in your GNU Radio checkout to get a list of all GRC files:

"find -name ".grc" -print"*

3.3. Basic Blocks

Some commonly used blocks will explain in this section.

3.3.1. UHD Blocks

The USRP Hardware Driver software (UHD) is the hardware driver for all USRP devices. It works on all major platforms (Linux, Windows, and Mac) and can be built with GCC, Clang, and MSVC compilers. The goal of the UHD software is to provide a host driver and API for current and future Ettus Research products. Users will be able to use UHD software standalone or with third-party applications, such as: GNU Radio, LabVIEW, Simulink, OpenBTS and Iris[33].

The bridge between GNU Radio and the USRP device is a set of blocks in the gr-uhd component, which includes:

- **UHD: USRP source block** - provides RX data to downstream processing blocks

  The USRP source block receives samples and writes to a stream. The source block also provides API calls for receiver settings.

- **UHD: USRP sink block** - accepts TX data from upstream processing blocks

The USRP sink block reads a stream and transmits the samples. The sink block also provides API calls for transmitter settings.

3.3.2. WX GUI Blocks

The most intuitive and straightforward way to analyze a signal is to display it graphically, both in time domain and frequency domain. For the applications in the real world, we have the spectrum analyzer and the oscillograph to facilitate us. Fortunately, in the software radio world, we also have such nice tools, thanks to wxPython, which provides a filexible way to construct GUI tools[38].

- **FFT sink block** - spectrum analyzer

In GNU Radio companion, the function of FFT sink block is a “soft spectrum analyzer”, based on fast Fourier transformation (FFT) of the digital sequence. This “soft spectrum analyzer” is used as the signal sink. That's why it is named as “fftsink”. It's defined in the module “**wxgui.fftsink.py**”. The function “**make_fft_sink_c()**” serves as the public interface to create an instance of the FFT sink:

gnuradio/wxgui/fftsink.py

```
...
def make_fft_sink_c(fg, parent, title, fft_size, input_rate, ymin=0, ymax=100):
    block = fft_sink_c(fg, parent, title=title, fft_size=fft_size,
                       sample_rate=input_rate, y_per_div=(ymax - ymin)/8, ref_level=ymax)
    return (block, block.win)
```

Notice that in Python, a function could return multiple values. “make_fft_sink_c()” returns two values: “block” is an instance of the class “fft_sink_c”, defined in the same module “**wxgui.fftsink.py**”. Another special feature of Python needs to be emphasized: Python supports multiple inheritance. “fft_sink_c” is derived from two classes: “gr.hier_block” and “fft_sink_base”. Being a subclass of "**gr.hier_block**" implies that "fft_sink_c" can be treated as a normal block, which can be placed and connected in a flow graph, as the next line shows:

```
self.connect (src, pre_demod)
```

“**block.win**” is obviously an attribute of “block”. In the definition of the class

"fft_sink_c", we can find its data type is the class "fft_window", a subclass of "wx.Window", also defined in the module "**wxgui.fftsink.py**". We can think of it as a window that is going to be hang up on your screen. This window "**block.win**" will be used as the argument of the method "vbox.Add".

- **Scope sink block** - Oscillograph

Another important WX GUI block in GNU Radio is the "Scope sink" which can be called "soft oscillograph". It would be very helpful if you wish to see the waveforms in the time domain. Its usage is quite similar to the "fft_sink":

```
if 1:
  scope_input, scope_win1 = |
            scopesink.make_scope_sink_f (self, panel, "Title", self.fs)
  self.connect (signal, scope_input)
  vbox.Add (scope_win1, 1, wx.EXPAND)
```

Note that here "signal" should be a real float signal. If you wish to display a complex signal with I/Q channels, "**make_scope_sink_c()**" is the right choice. Copy these lines wherever you think a scope should appear, then connect it to the signal as a block. Refer to "**gnuradio/wxgui/scopesink.py**" for more details.

3.4. Python codes explanation

Tables below are descriptions of the key blocks for the implementations in this thesis[39].

3.4.1. wfm_rcv_pll.py

Stereo demodulating a broadcast FM signal with a deemphasis.

wfm_rcv_pll ( )

Type	Function
Description	Hierarchical block for demodulating a broadcast FM signal. The input is the down converted complex baseband signal (gr_complex).The output is two streams of the demodulated audio (float) 0=Left, 1=Right.

Usage	blks.wfm_rcv_pll(fg, demod_rate, audio_decimation)
Parameters	fg: flow graph. type fg: flow graph demod_rate: input sample rate of complex baseband input. type demod_rate: float audio_decimation: how much to decimate demod_rate to get to audio. type audio_decimation: integer

Table 3.1 Explanation for “WBFM Receiver Pll” block

3.4.2. gmsk.py

differential QPSK modulation and demodulation

gmsk_mod ( )

Type	Function , GMSK modulator
Description	Hierarchical block for Gaussian Minimum Shift Key (GMSK) modulation. The input is a byte stream (unsigned char) and the output is the complex modulated signal at baseband.
Usage	blks.gmsk_mod(fg, samples_per_symbol =2, bt=.35, verbose=False, log=False)
Parameters	fg: flow graph type fg: flow graph samples_per_symbol: samples per baud >= 2 type samples_per_symbol: integer bt: Gaussian filter bandwidth * symbol time type bt: float verbose: Print information about modulator? type verbose: bool debug: Print modulation data to files? type debug: bool

Table 3.2 Explanation for “GMSK Mod” block

gmsk_demod ( )

Type	Function , GMSK demodulator
Description	Hierarchical block for Gaussian Minimum Shift Key (GMSK) demodulation. The input is the complex modulated signal at baseband. The output is a stream of bits packed 1 bit per byte (the LSB)
Usage	blks.gmsk_demod(fg, samples_per_symbol=2, gain_mu=None, mu=0.5, omega_relative_limit=0.005, freq_error=0.0, verbose=False, log=False)
Parameters	fg : flow graph type fg: flow graph samples_per_symbol: samples per baud type samples_per_symbol: integer Verbose: Print information about modulator? type verbose: bool log : Print modulation data to files? type log: bool Clock recovery parameters. These all have reasonable defaults. gain_mu: controls rate of mu adjustment type gain_mu: float mu: fractional delay [0.0, 1.0] type mu: float omega_relative_limit: sets max variation in omega type omega_relative_limit: float, typically 0.00020 0 (200 ppm) freq_error: bit rate error as a fraction type freq_error:float

Table 3.3 Explanation for “GMSK Demod” block

3.4.3. ofdm.py

OFDM mod/demod with packets as i/o.

ofdm_mod ( )

Type	Function
Description	Modulates an OFDM stream. Based on the options fft_length, occupied_tones, and cp_length, this block creates OFDM symbols using a specified modulation option. Send packets by calling send_pkt Hierarchical block for sending packets. Packets to be sent are enqueued by calling send_pkt. The output is the complex modulated signal at baseband.
Usage	blks.ofdm_mod (fg, options, msgq_limit=2, pad_for_usrp=True)
Parameters	fg : flow graph type fg: flow graph options : pass modulation options from higher layers (fft length, occupied tones, etc.) msgq_limit: maximum number of messages in message queue type msgq_limit: int pad_for_usrp: If true, packets are padded such that they end up a multiple of 128 samples
Sub Function	Blks.ofdm_mod.send_pkt(payload, eof=False)
Description	Send the payload
Parameters	payload: data to send type payload: string eof: To signal end of transmission Type eof: Bool True or False

Table 3.4 Explanation for “OFDM Mod” block

4 Implementation

The list below are the main hardware and software which are used in the follow four experiments.

Hardware:

- ASUS K43T laptop
- Ettus USRP B100
- WBX daughterboard (covers 50MHz – 2.2GHz)
- Arrow antenna
- Logitech ClickSmart 510 webcamera

Software:

- Linux OS: Ubuntu 12.10
- GNURADIO 3.6.4.1
- VLC media player 2.0.5 Two flower

4.1. GSM Scanning

Normally there are a number of GSM bands which a mobile phone can use and these bands can be different depending on the country. As figure4.1.1 shows, most of the world uses the GSM-900 and GSM-1800 bands, but the the United States, Canada and other parts of the Americas which use the GSM-850 and GSM-1900 bands.

Frequency Bands	Names	Channel Number	Uplink (MHz)	Downlink (MHz)	Description
GSM 850	GSM 850	128-251	824,0-849,0	869,0-894,0	USA, South American and part of Asian countries.
GSM 900	P-GSM 900	1-124	890,0 - 915,0	935,0 - 960,0	The GSM frequency band which is the first to achieved and most widely used.
	E-GSM 900	975 - 1023	880,0 - 890,0	925,0 - 935,0	900M Extension band.
	R-GSM 900	n/a	876,0 - 880,0	921,0 - 925,0	GSM-R, special version developed for Railway dispatch communication system.
GSM1800	GSM	512 - 885	1710,0 -	1805,0 -	Applies to the market which has a great demand

	1800		1785,0	1880,0	of channel capacity.
GSM1900	GSM 1900	512 - 810	1850,0 - 1910,0	1930,0 - 1990,0	Mainly for American countries, the system is not compatible with the 1800M because of frequency overlap.

Table 4.1 GSM Frequency bands

The purpose of this project is to detect the surrounding GSM Base transceiver station (BTS) mainly used the FFT function of USRP.

First, enter folder "**gnuradio/gr-uhd/examples/grc**".

Then double click "**uhd_fft.grc**" directly.or type in the command terminal:

./uhd_fft.py –s 8M –g 32 –f 940M

A plot window like figure 4.1.2 is shown with a constantly moving blue line which represents the amplitude of the signal detected at that pre-configured frequency. Here we observed a 8MHz band of spectrum which centered at 940MHz. There are several BTS and each one transmitting in a 200KHz wide channel.

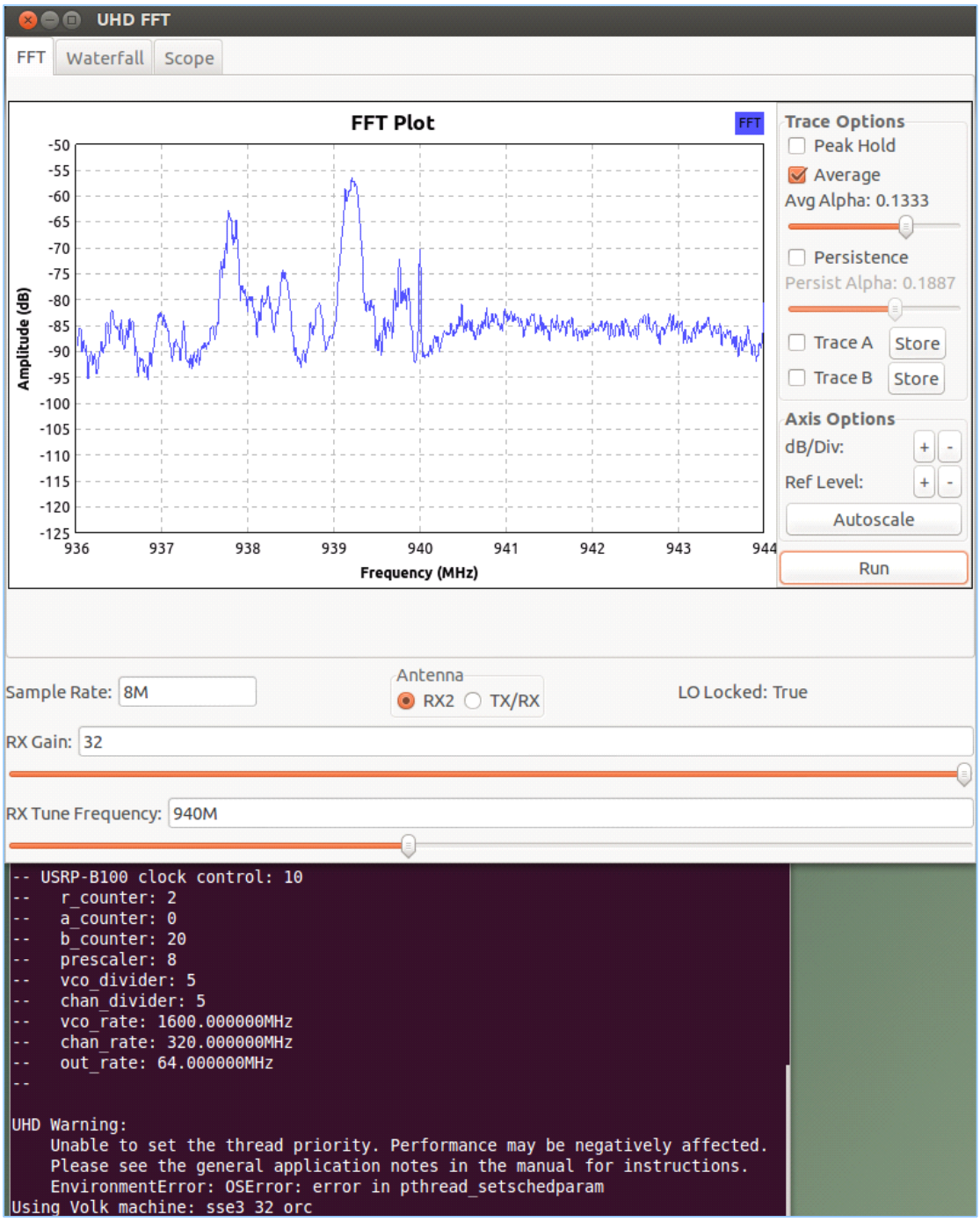


Figure 4.1.1 UHD FFT window

Next step is trying to find a active channel which should be visible in plot display as wide bump centered around a vertical division. Here it is very convenient to modify the center frequency by sliding the WX GUI Slider or typing a new value in the lattice. As the figure 4.1.4 shows a channel bump near center frequency 944.8 MHz.

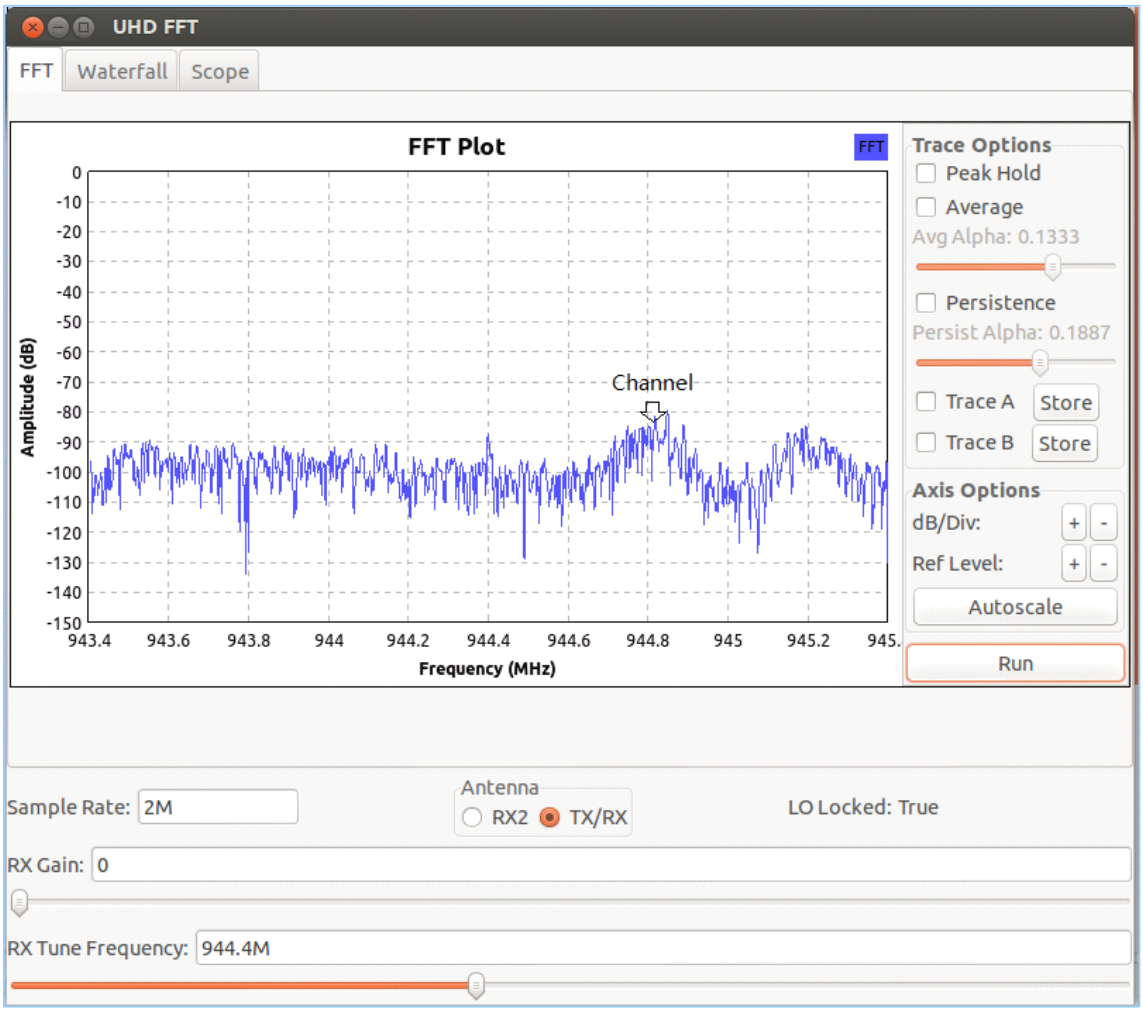


Figure 4.1.3 GSM Channel bump

To further understand the channel features, continue to slide the slider and we found a interesting channel which showed on figure 4.1.4.

According to the GSM standard, GSM is FDMA (Frequency Division Multiplexing Access) and TDMA (Time Division Multiplexing Access) systems, the user channel is represented by a time slot, Therefore, when a communication between mobile station and base station must be precisely synchronized, FCCH (Frequency Correction Channel), MS achieved the clock coarse synchronization through FFCH, and then achieve precise synchronization by using SCH. After the previous two steps to determine the timing of the base station transmitted information, and finally receive the slot data from BCCH (Broadcast Control Channel). So the FFCH is a mechanism for the mobile phone to find the base station.

The FCCH generates a Frequency correction burst (FB) which can be found on

the frequency spectrum plot as a peek frequency offset 1625/24 kHz above the carrier center. The green line which means peak hold in figure 4.1.4 highlight a FB which is clearly visible as a narrow peak in the plot. Notice that the spectrum on the right of the center channel has a similar amplitude, It means there are other channels may be using it for data traffic or USRP has captured more than one BTS.

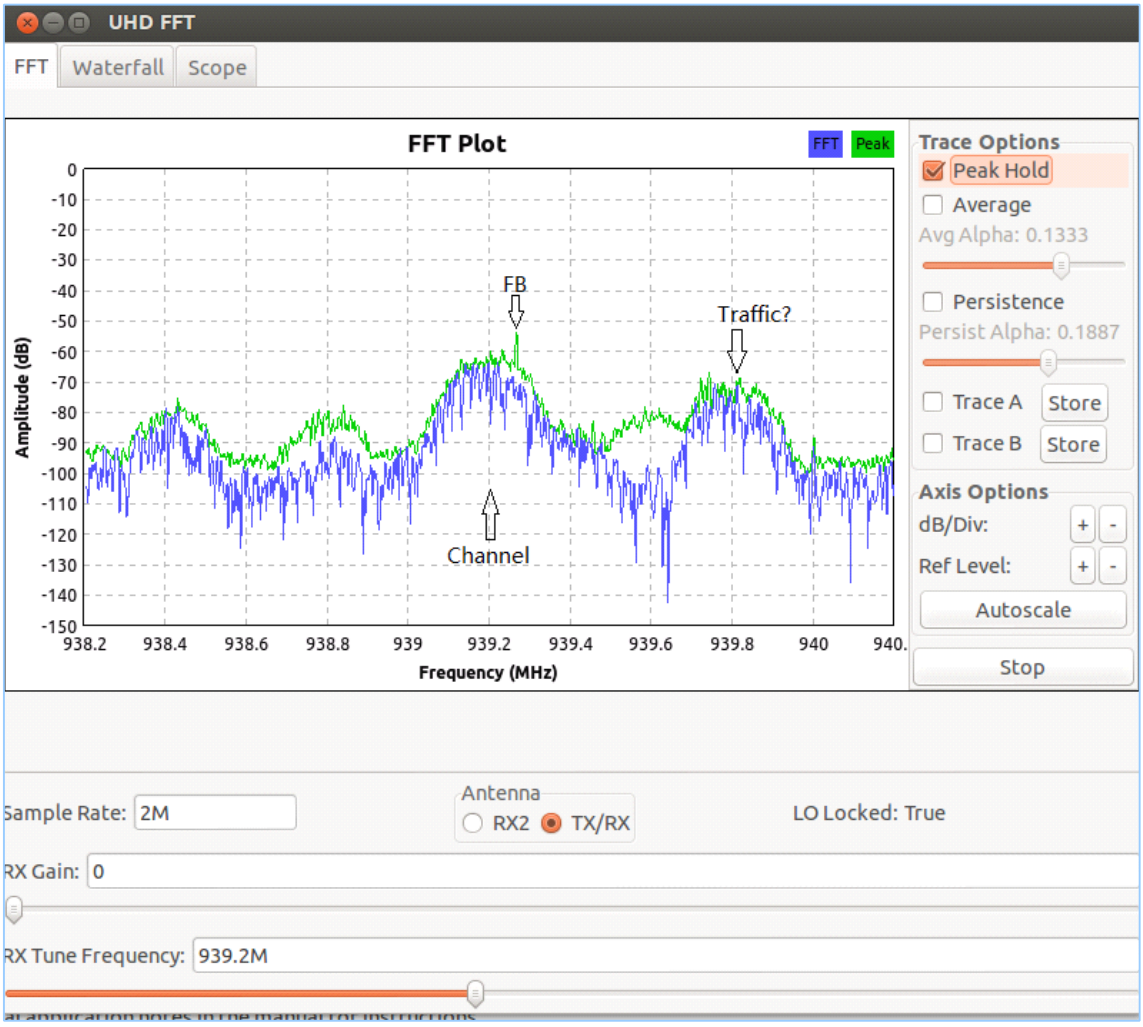


Figure 4.1.4 Frequency correction burst and possible traffic channel

This project provide a FFT method which is combined with USRP and GNU Radio for scanning the GSM BTS and analyzing the channel. And not all of the channels are used in every area, as we found that some lower strength signals which means some of channels are reserved for using in neighboring cells. Additionally, Frequency correction burst (FB) which enables the mobile to synchronize its frequency with the master frequency appears on frequency correction channel (FCCH).

4.2. FM Receiver

As a popular example, the implementation of an FM receiver with GUI, will be introduced and analyzed in this section. The FM signal from the air is received by the USRP board and then gets processed in the FPGA and in the computer. Finally, the demodulated signal is played using the sound card. You can hear a very high quality FM signal just by inserting a copper wire into the Basic RX daughter card.

Typical FM receivers are constructed entirely using hardware that must be fabricated in a plant. This procedure will demonstrate the power of software defined radio and how easy it is to use as figure 4.2.1 shows in GNU Radio companion.

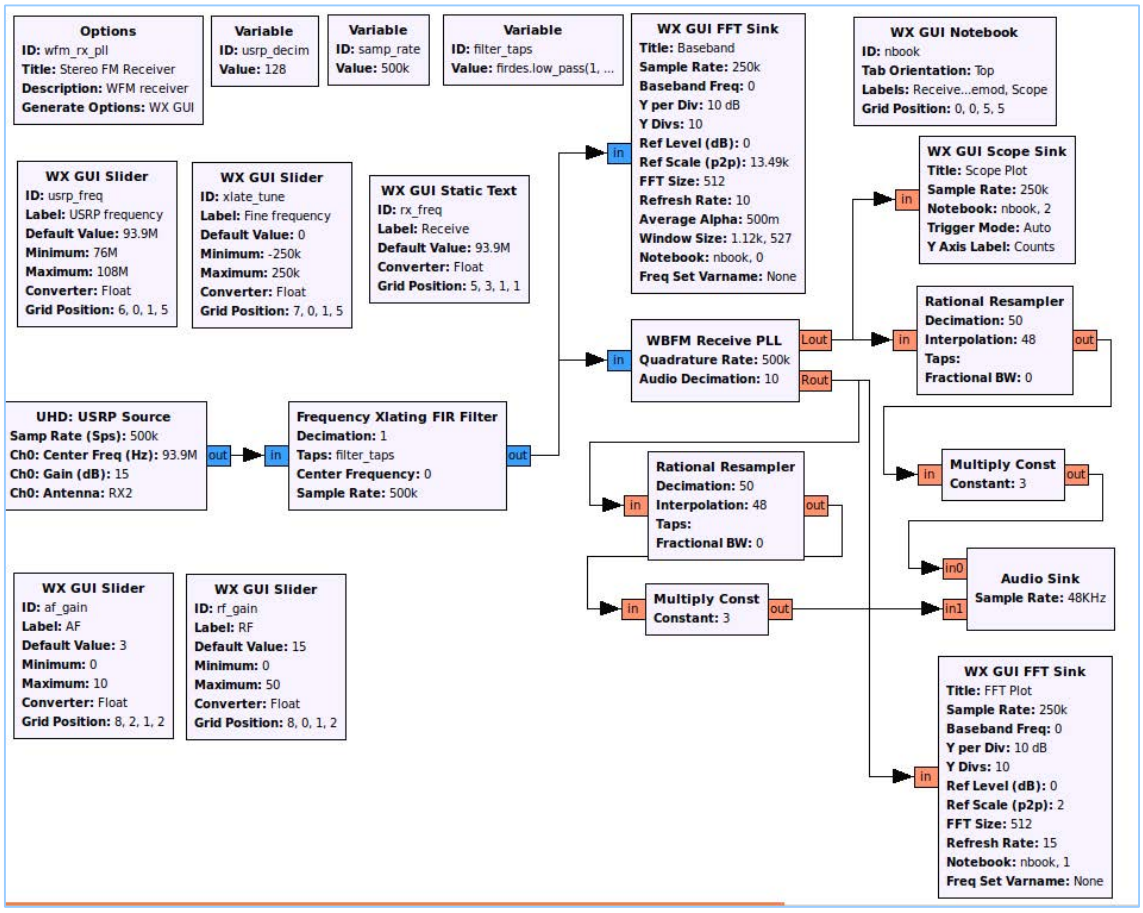


Figure 4.2.1 Stereo FM Receiver system

The “UHD: USRP Source” block is used to retrieve samples received on the USRP device connected to the computer as discussed in previous chapter. The sample rate internally at 64Msps in USRP B100. In the USRP block you can specify a lower

rate through set the decimation, and the USRP will try to match it quite closely if it's not evenly divisible by 64Msps by down conversion done inside the USRP. In this case the sample rate is set to 500ksps which can be evenly divisible by 64Msps. The reason to chose the 500ksps sampling rate is the nicely divisible with the supported 48000 Hz sampling rate of the sound card in the computer, and is enough to reproduce the full ~100 kHz wide FM radio station. The center frequency is the frequency that the USRP should tune into. So with a 500ksps sample rate, we will be able receive a 500kHz band around the center frequency[38].

The “Frequency Xlating FIR Filter” block following the USRP block efficiently combines a frequency translation (typically “down conversion”) with a FIR filter (typically low-pass) and decimation. It is ideally suited for a “channel selection filter” and can be efficiently used to select and decimate a narrow band signal out of wide bandwidth input. In this case it is by using a slider bar to change the center frequency of the Frequency Xlating FIR Filter to achieve fine tuning.

The WBFM block does the actual heavy work of decoding the FM signal and converting it into an audio signal that can be processed by the sound card, represented by the Audio Sink block. The FM stereo modulation is illustrated in the diagram below:

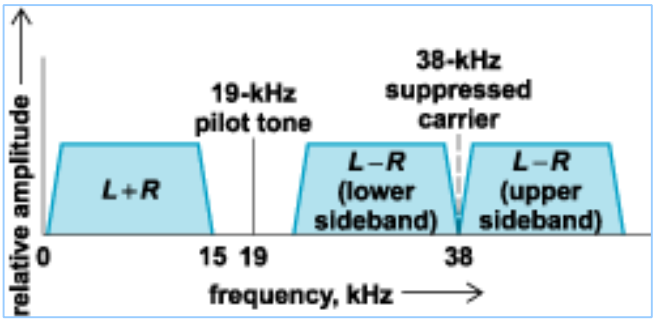


Figure 4.2.2 Stereo FM theory

L corresponds to left signal and R to the right. A mono receiver sees only the L + R signal. The information which can be used to produce the stereo signal is contained in the L + R and the L - R signals. The L - R signal is added to the baseband signal as a DSB signal with suppressed carrier. A pilot tone at 19kHz is used by the receiver to regenerate the carrier for demodulating the DSB L - R signal. The baseband signal shown above is used to frequency modulate a carrier.

The results of this stereo FM receiver allowed us to listen to multiple radio stations with acceptable to exceptional Stereo quality. WX GUI Scope Sink is for observing time domain signals and WX GUI FFT Sink used to observing frequency spectrum which shows on figure4.2.2.

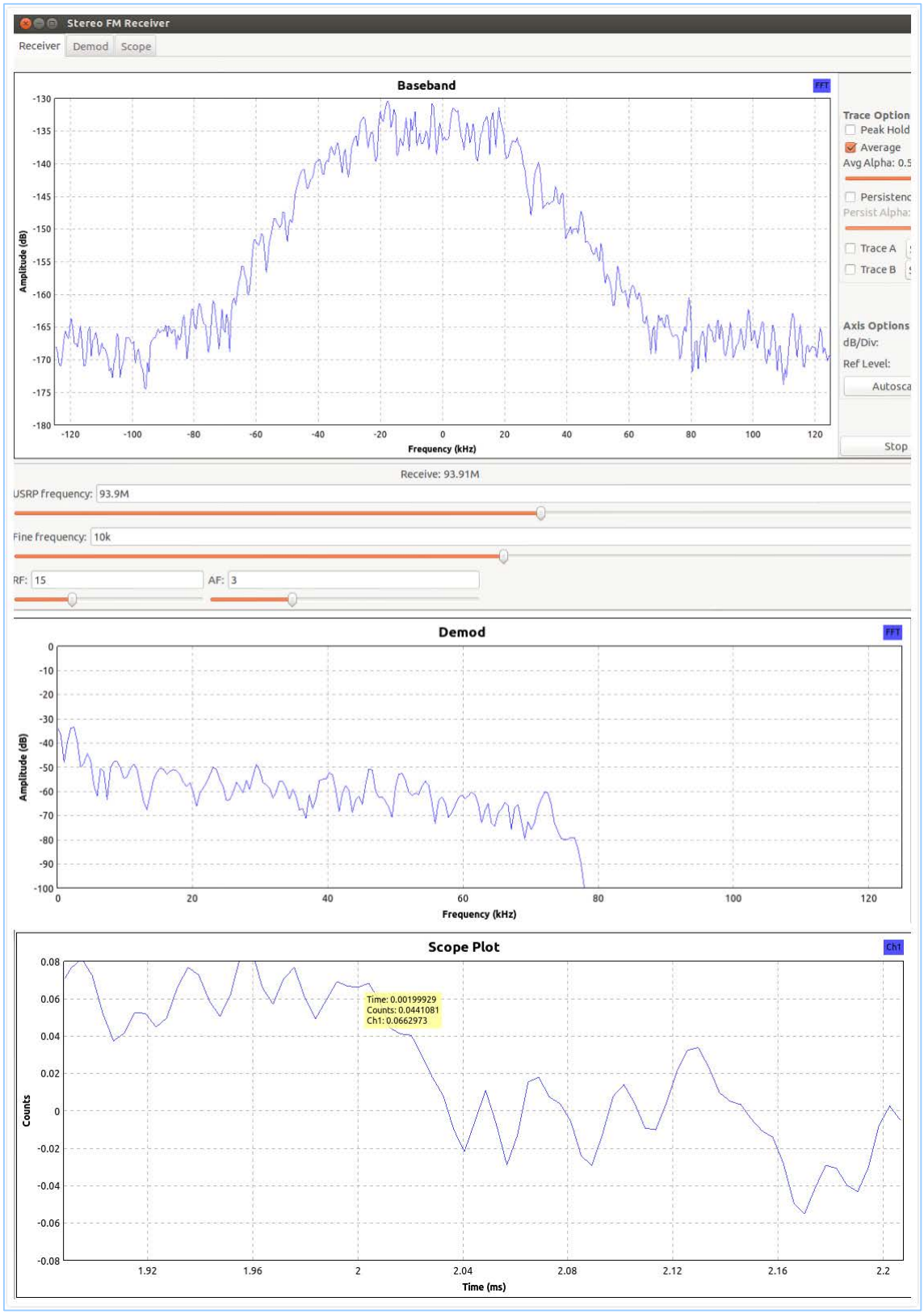


Figure 4.2.2 Received signal spectrum of Stereo FM Receiver system

In this project, the frequencies are read in from the USRP source, fed to an FIR

filter, demodulated with a phase-locked loop, and then resampled to a frequency compatible with sound card of PC and allowed users to listen to multiple radio stations with acceptable to exceptional stereo quality.

4.3. Benchmark OFDM

There are reconfigurable python codes which provided by Ettu Research named "**benchmark_ofdm_tx.py**" and "**benchmark_ofdm_rx.py**" in the folder "**/gnuradio/gnuradio-example/python/ofdm/**" as an OFDM example in the GNU Radio package and used to transmit and receive OFDM signals as the flow chart below[40]:

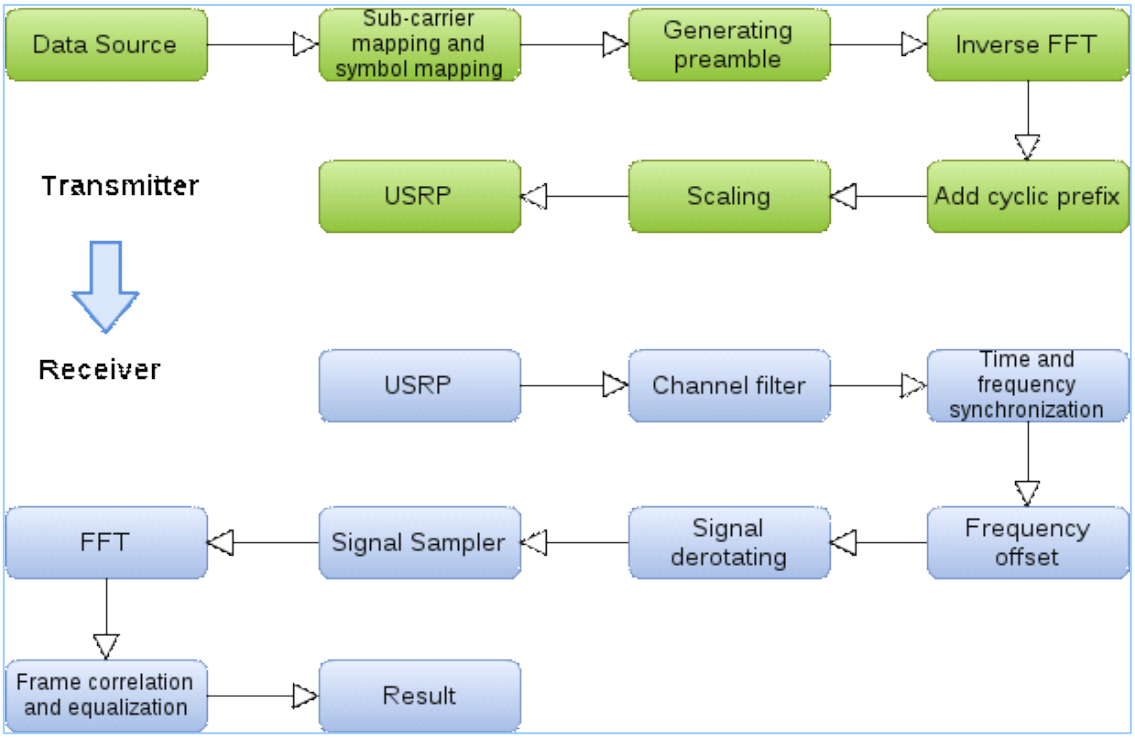


Figure 4.3.1 OFDM transmitter (up) and receiver (down)

In this project, use two USRP equipped WBX daughterboards connected with two apart computers. One as transmitter and the other as receiver. Accordingly, the source code "**benchmark_ofdm_tx.py**" run on the GNU Radio of computer which as the transmitter and "**benchmark_ofdm_rx.py**" for the receiver.

- **OFDM spectrum**

On the transmitter side, connect the port of TX/RX port to spectrum analyzer or use two antenna to get the signal spectrum and set the center frequency to 474Mhz and the span set to 500KHz which is the default sample rate value of "**benchmark_ofdm_tx.py**". Then enter the "**/gnuradio/gnuradio-examples/python/ofdm/**" folder and type the command[41]:

./benchmark_tx.py -f 474M -m bpsk -A TX/RX

It means set the center frequency to 474MHz and modulation is BPSK and use the TX/RX port to transmit the OFDM signal. Other parameters are default value like sample rate is 500KHz and FFT length is 512. if you want to modify these kind of value, type:

./benchmark_tx.py -h

to acquire more command help.

In this case we can obtain the result as follow:

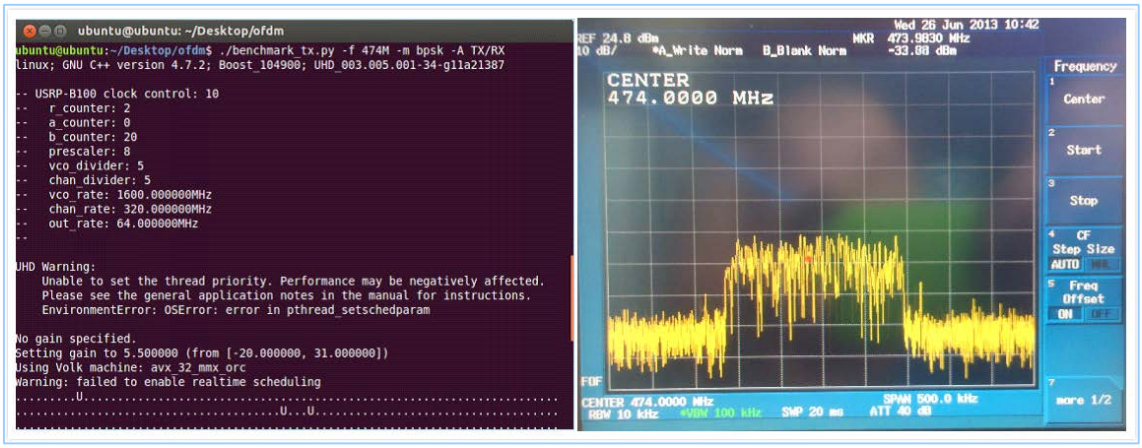


Figure 4.3.2 Using the "benchmark_tx.py"

- **Data Transmission and Reception**

On the receiver side, enter the "**/gnuradio/gnuradio-examples/python/ofdm/**" folder set the corresponding parameters like the transmitter side.

In this case, type:

./benchmark_rx.py -f 474M -m bpsk -A TX/RX

The packets will delivery from the transmitter side to the receiver side.

As discussed in chapter 3, the most intuitive and straightforward way to analyze a signal is to display it graphically. So GNU Radio companion is a better choice to analysis the OFDM spectrum and set the parameters flexibility. Figure4.3.2 is a simulation which designed for this project.

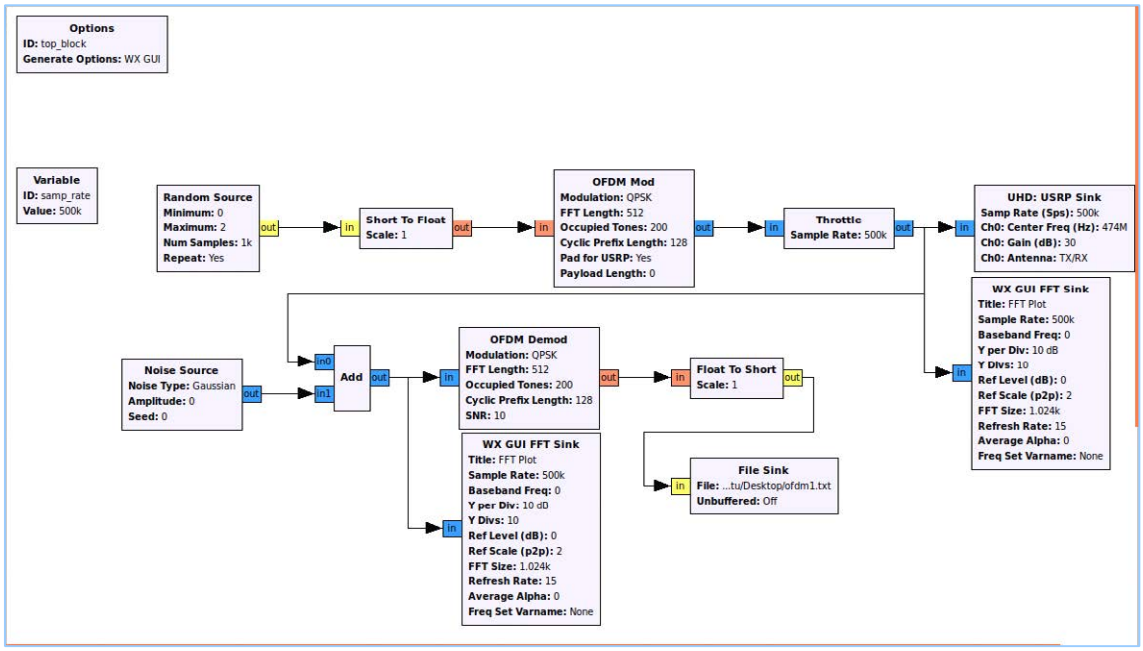


Figure 4.3.3 Simulation for OFDM transmission

A series of random binary data were generated by “Random source” block through the OFDM modulation block and delivery to the USRP to transmit into the air. At the same time, the modulated signal obtained by WXGUI tool. On the other side, the signal is received by the receiver USRP to demodulate and analysis. In this case, the simulation flowchart didn't consider about the packet lost and other problems which occurs in the actual transmission.

After generating and executing the GNU Radio companion will obtain a spectrum which is very similar with the one obtained by the benchmark. The bandwidth of OFDM signal is:

$$Bw = \frac{sample\_rate * occupied\_tones}{FFT\_Length}$$

(3)

So in this case, the bandwidth should be near 200KHz.

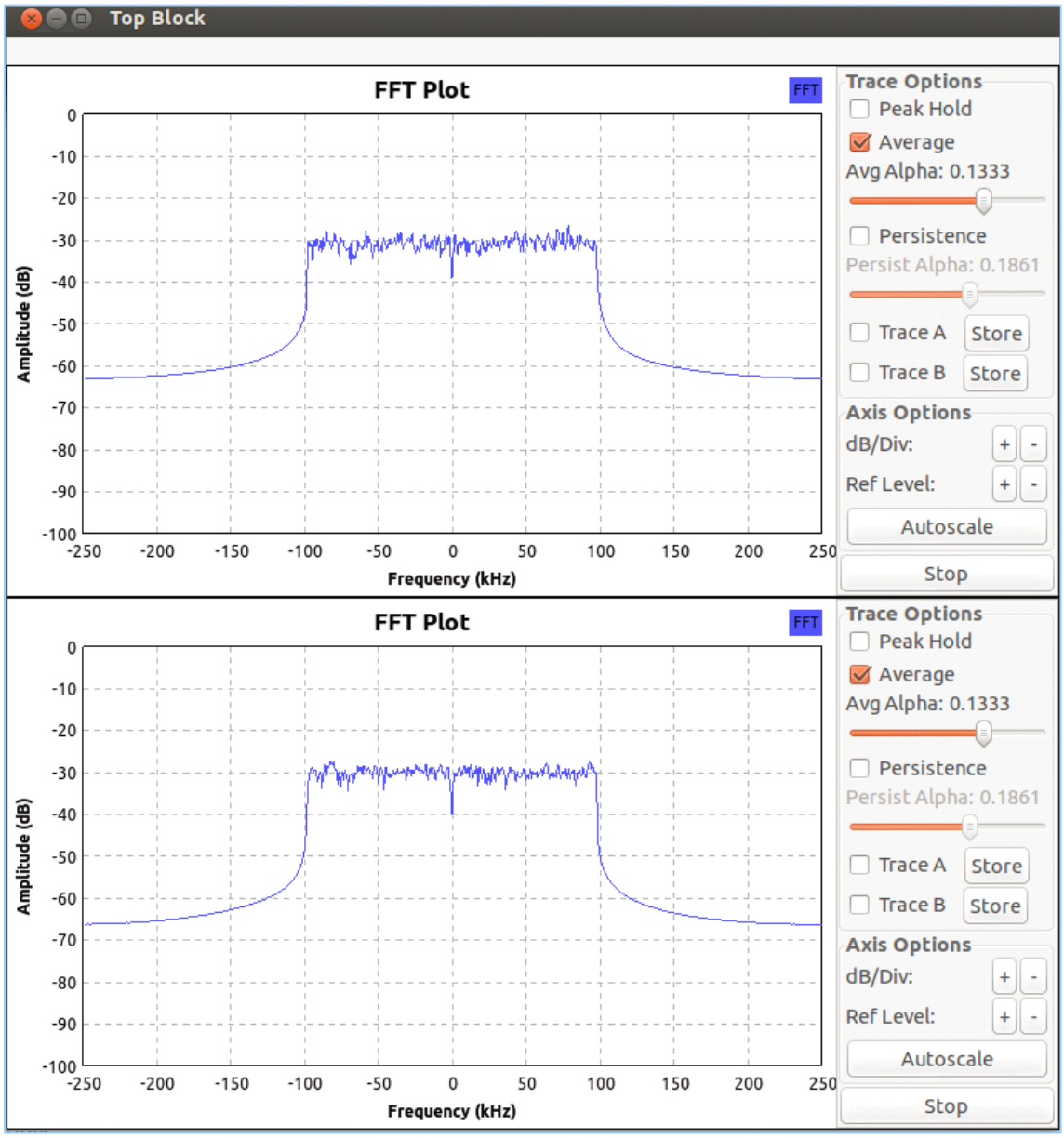


Figure 4.3.4 OFDM signal spectrum(500ksps, occupied tones=200, FFT length=512)

Based on the equation above, modify the occupied tone to 100, and increase the FFT length to 1024. the bandwidth should be near 50KHz and the actual sub-carriers will decreased as the occupied tone decreased. The result is very coincide just

the figure below shows.

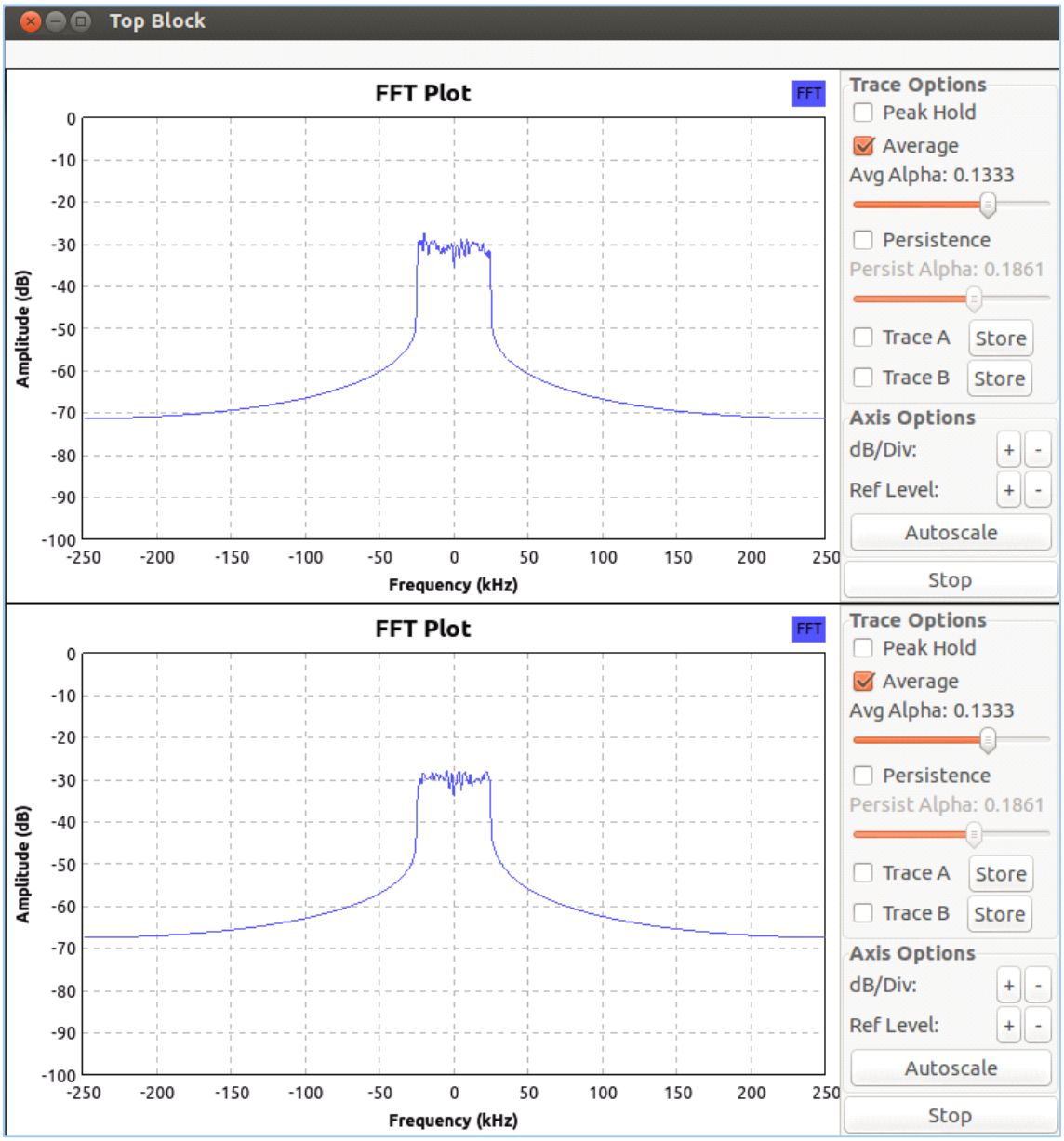


Figure 4.3.5 OFDM signal spectrum(500ksps, occupied tones=100, FFT length=1024)

4.4. Real-time Digital Video Broadcasting

Digital Video Broadcasting (DVB) is a transmission scheme based on the MPEG-2 video compression / transmission scheme and utilizing the standard MPEG-2 Transmission scheme. however It is much more than a simple replacement for existing analogue television transmission. In the first case, DVB provides superior picture quality with the opportunity to view pictures in standard format or wide screen (16:9) format, along with mono, stereo or surround sound. It also allows a range of new

features and services including subtitling, multiple audio tracks, interactive content, multimedia content - where, for instance, programme may be linked to world wide web material[42].

In this case, the objective of this project is to create a real-time digital video broadcasting setup using easily available components like VLC media player, GNU Radio and Universal Software Radio Peripherals which illustrated on the figure 4.4.1.

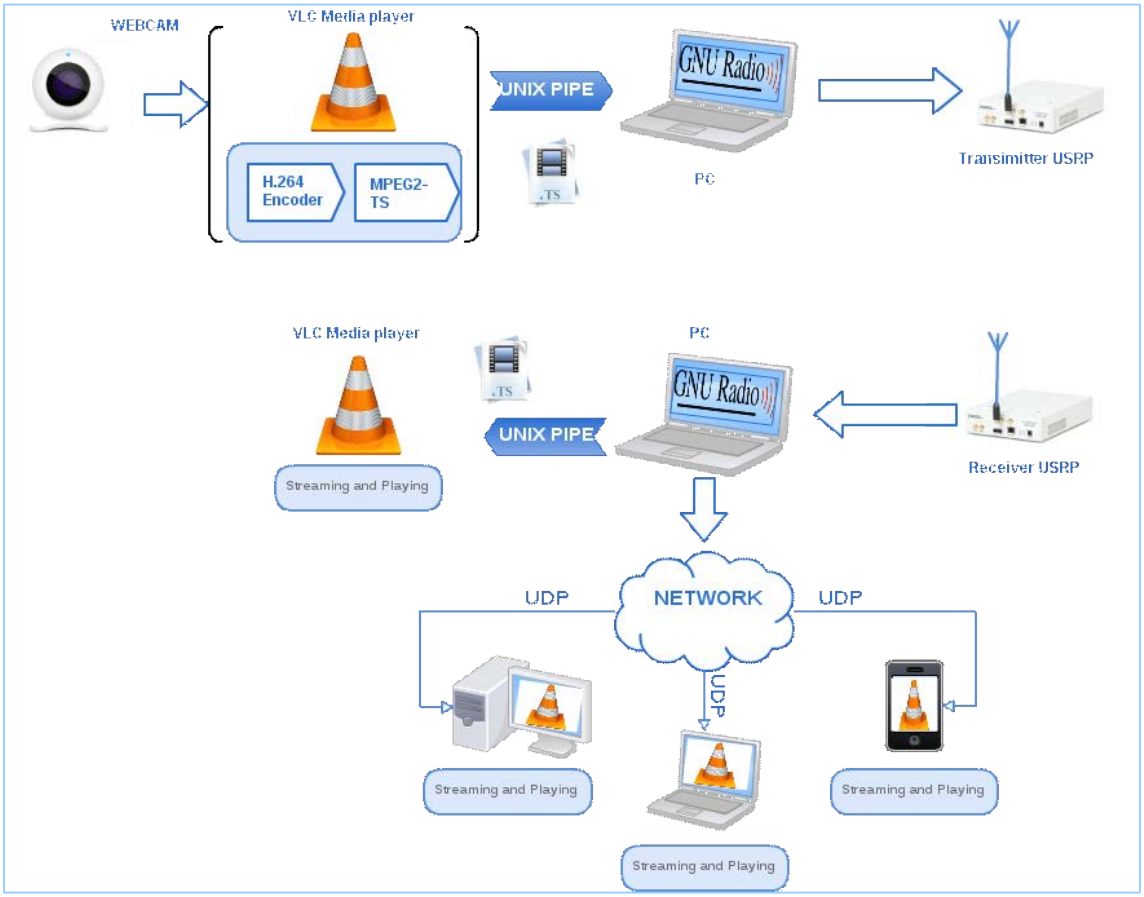


Figure 4.4.1 DVB project description diagram

In the project description diagram, the source file can be a existed video file or the video stream which captured by the webcam and encoded in H.264 standard by VLC media player. When the TS stream was generating, GNU Radio will read the stream through UNIX pipe, encoder the packet and then modulated by GMSK(Gaussian Minimum Shift Keying) modulation scheme[43].

Then a transmitter USRP is used to transmit the GMSK modulated signal into air. On the other side, the transmitted signal will received by another receiver USRP

which has been set the same center frequency, demodulation and packet decoding implemented in GNU Radio,the output of GNU Radio should be the TS stream which can be played by VLC media player through the UNIX pipe on the receiver side. Notice that the receiver computer can be a UDP server which can broadcast the video to other clients computer. Finally the whole system achieved a real-time digital video broadcasting.

4.4.1. Transmitter Side

CNU Radio companion

On the transmitter side, as the block diagram below shows, first, an existed video file or a real time writing UNIX pipe was using in the File source block, then the TS file delivery to packet encoder block, in this case, it was set 2 samples per symbol and one bit per symbol. So the packet encoding here was 2 bit per sample or 16M bit per second, because the variable “samp_rate” was set to 8MHz.

After the encoding finished, the encoded packet was modulated by “GMSK mod” block which was set 2 samples per symbol too. A “Multiply Const” block follow the modulation block was used as a amplifier. Finally three different kind of WXGUI tools detected the amplified and modulated signal to analysis the spectrum from time domain and frequency domain. At the same time, the amplified and modulated signal was delivery to the USRP to finish the rest operation.

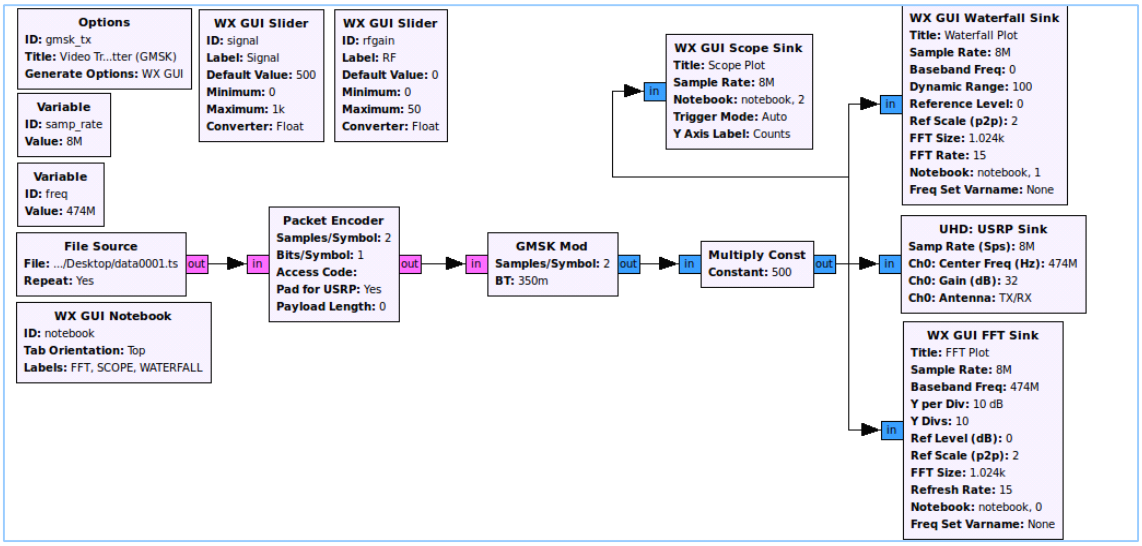


Figure 4.4.2 Transmitter GNU Radio block diagram

Transmitter USRP

When the transmitter USRP which connected with transmitter GNU Radio receive the amplified and modulated signal, the FPGA will finish the interpolation and up-conversion by (DUC), then the DAC (digital-to-analog converter) convert the signal to sent into the air.

Configuring VLC[44]

As we know from the tutorials which provided by VLC team[45]. In the transmitter side, to know the video and audio files which captured by the webcamera, open the terminal window and type:

*ls /dev/video**
*ls /dev/audio**

Or open the VLC media player select "**Media-> Open Capture Device**.." on the top bar. Then you can find the folder of video and audio files in the option named "**Device selection**".

In our case, the video and audio files associated were video0 and audio. Then to run the VLC camera streaming from the terminal, type the following commands:

vlc v4l2:// :v4l-vdev="/dev/video0" :v4l-adev="/dev/audio" :v4l-norm=3 :v4l-frequency=-1 :v4l-caching=300 :v4l-chroma="" :v4l-fps=-1.000000 :v4l-samplerate=44100 :v4l-channel=0 :v4l-tuner=-1 :v4l-audio=-1 :v4l-stereo :v4l-width=640 :v4l-height=480 :v4l-brightness=-1 :v4l-colour=-1 :v4l-hue=-1 :v4l-contrast=-1 :no-v4l-mjpeg :v4l-decimation=1 :v4l-quality=100
--sout
"#transcode{vcodec=mp2v,vb=800,scale=1,acodec=mpga,ab=128,channels=2,samplerate=44100}:duplicate{dst=std{access=file,mux=ts,dst=/home/ubuntu/Desktop/tx.ts},dst=display}"

where the VLC captured video file "**tx.ts**" at the address which is defined by

“**dst**” in the above command the is the UNIX pipe combine to the GNU radio of transmitter. Additionally it is configured for MPEG2 codec and MUX = MPEG/TS.

4.4.2. Receiver Side

Receiver USRP

On the receiver side, the DDC of FPGA and ADC(analog-to-digital converter) deal with the signal down-conversion and convert to digital signal for baseband processing.

GNU Radio

In contrast with the transmitter, the modulated baseband signal will demodulate by the “GMSK Demod” which follow with the “UHD: USRP source” block. Correspondingly the “Packet Decoder” block is for unpack and extract the data. The parameters “threshold” is for detecting the access code with up to threshold bits wrong (0 -> use default). In this case, there is no access code was used. Finally, the video data will broadcast to different kind of device to play. For high speed and real time video transmission, UDP protocol was chosen in this project, so for the destinations of video data, we assigned three “UPD Sink” block with correspond IP address and port to broadcast to three different clients. A desktop computer with Ubuntu 12.10, a laptop using Windows 7 and a mobile phone with android 4.0 were tested in this project. And certainly, a file sink which for create a file with “.ts” extension would record the final data was set as another destination for the video stream. The GRC setup for the receiver as below.

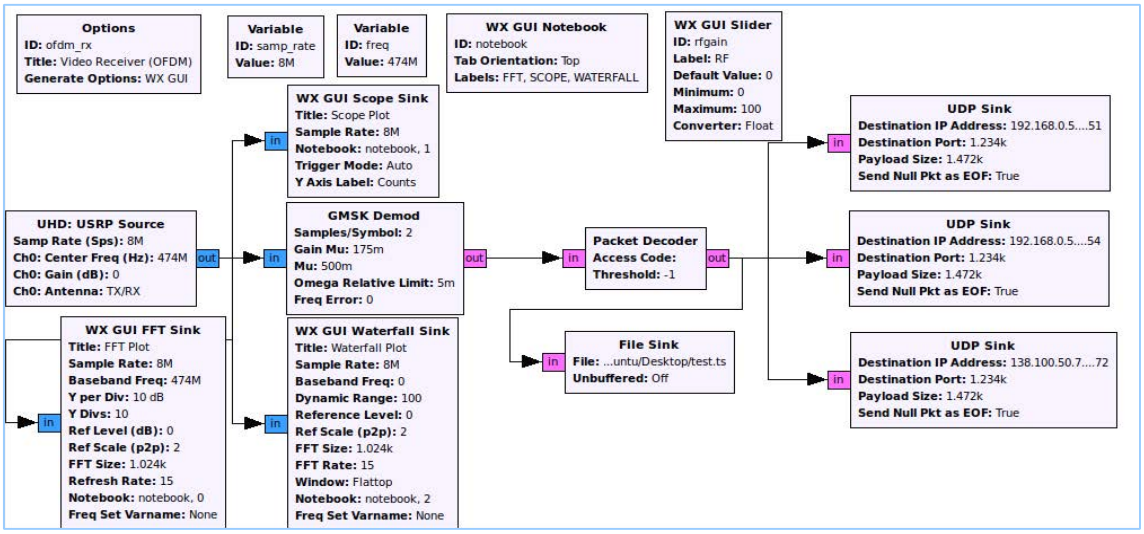


Figure 4.4.3 Receiver GNU Radio block diagram

Configuring VLC

In this project we choose UDP protocol as we discussed in the GNU Receiver setup, UDP is a better choice for real-time video transmission because of it's high transmit speed and "no ACK" mechanism which save much network resources .

So in this case, open VLC player. Click on "**Media**" and select "**Open network stream...**". In the "**Network Protocol**" enter a network URL as the format: "**udp://:@port number**" like the figure 4.4.2 shows. the port number is the port of client to receive the TS stream and delivery to VLC media player.

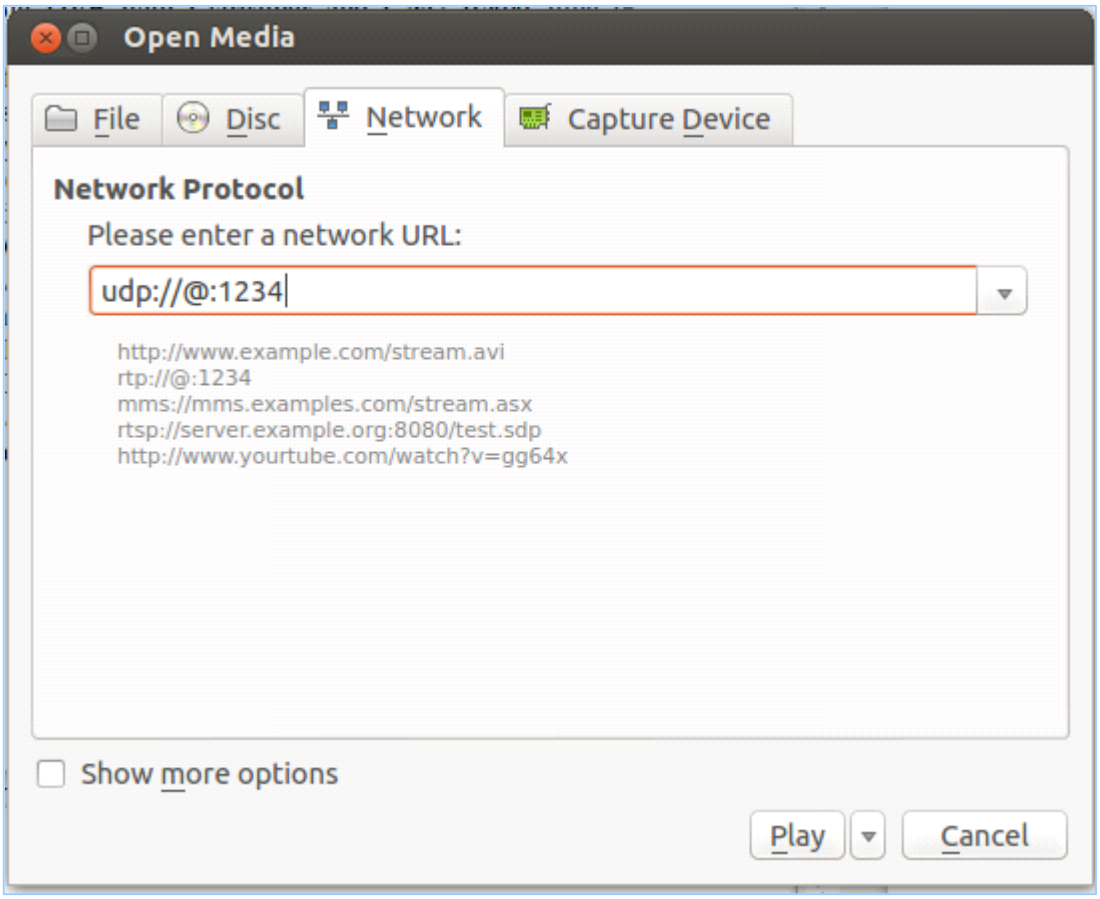


Figure 4.4.4 UDP client setup

When all of the steps above finished, click the "**play**" bottom, VLC will set to listen mode to listen the stream will flow from the assigned port in "**udp://:@port number**".

Note that if the receiver side need to play with broadcast clients or more clients require the video broadcast, just add another "UDP Sink" and type their own IP address and opened port as above setting.

4.4.3. Simulation

For evaluate the impact of noise for the transmission. The block diagram below introduced a simulation in GNU Radio companion. It makes the modulated signal add a Gaussian noise which can control the value by "WX GUI Slider" block.

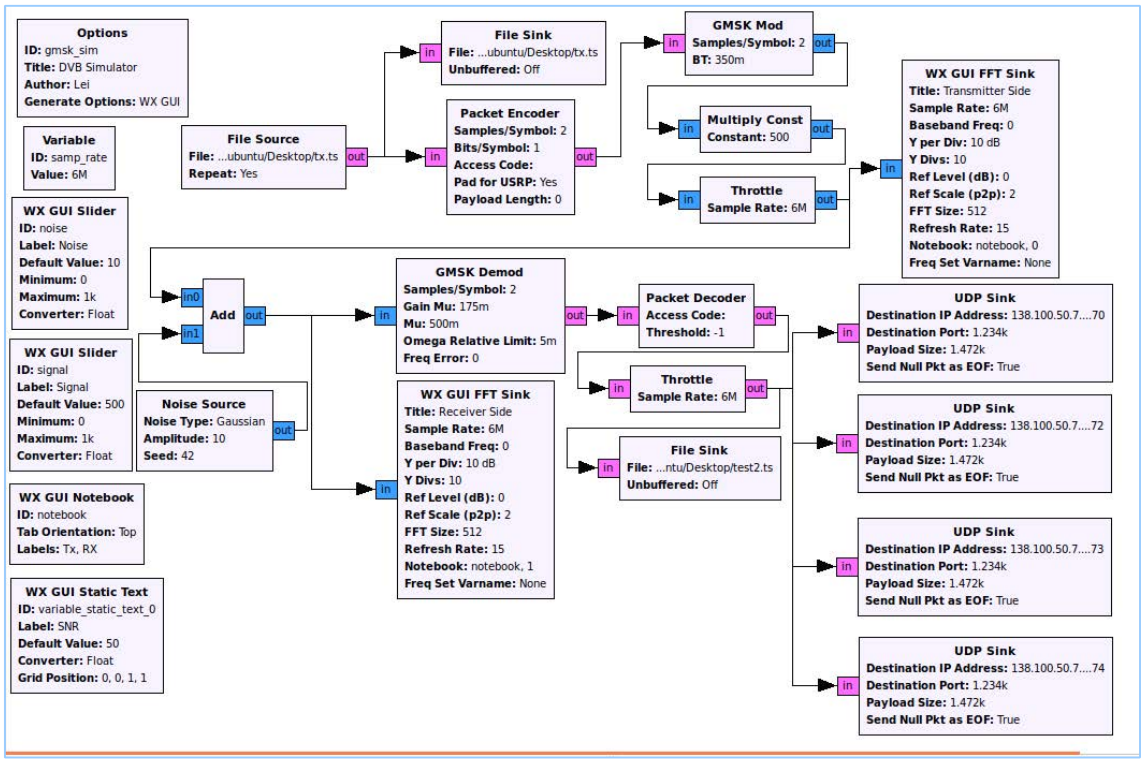


Figure 4.4.5 DVB simulation block diagram

- **Result**

Based on the previous introduction and preparation, One existed video file and a real-time captured video stream by VLC media player were tested in this project. After executing the GNU Radio companion and we obtained the result below.

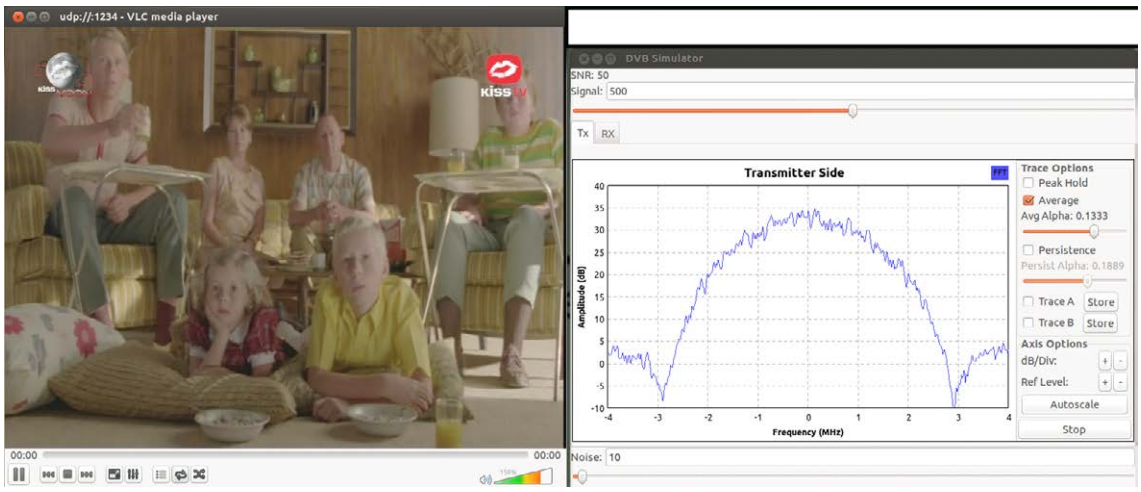


Figure 4.4.6 Video transmission when SNR=50

When the transmission in condition of every parameters set in the default value and the network works stable, the SNR equal to 50 as the right window shows in figure 4.4.5, the video broadcasting very good and the GMSK spectrum of transmitter and receiver almost the same which is a gently undulating waveform, and the receiver side almost has no delay and the quality playing on clients seems well too. Figure 4.4.6 shows the broadcasting video played on three different clients. Synchronization was excellent when the network environment is stable.

Figure 4.4.7 Broadcasting video on three different clients

After a period of steady test, make the value of Gaussian noise which add with modulated signal slowly began to increase. When the SNR=10 which is the 1/5 the default value, the quality of video transmission decreased dramatically. The video playing on clients Appeared a large number of blocky noise, image and sound have some degree of non-smooth. And the GMSK spectrum become to a rapid undulating waveform like figure 4.4.7 shows. Additionally, the video playing on mobile phone even stoped.

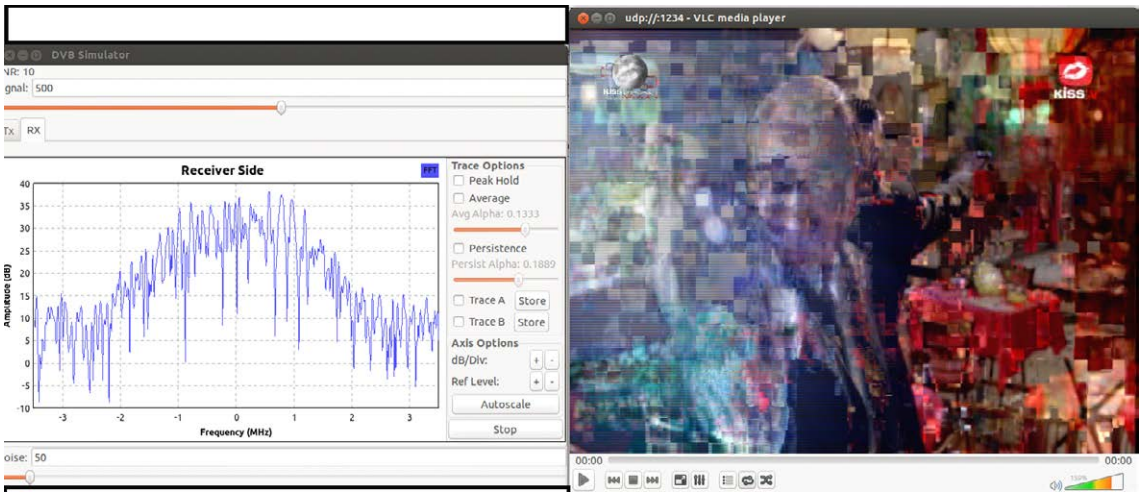


Figure 4.4.8 Video transmission when SNR=10

The similar experiment has done on webcamera mode. The video captured by VLC in real-time transmitted well, but when we increased the noise like the previous test, the same situation which was the bad transmission quality occurred as the figure 4.4.8 shows. When the noise value continue to increased, the video playing on receiver side and other clients will stoped.

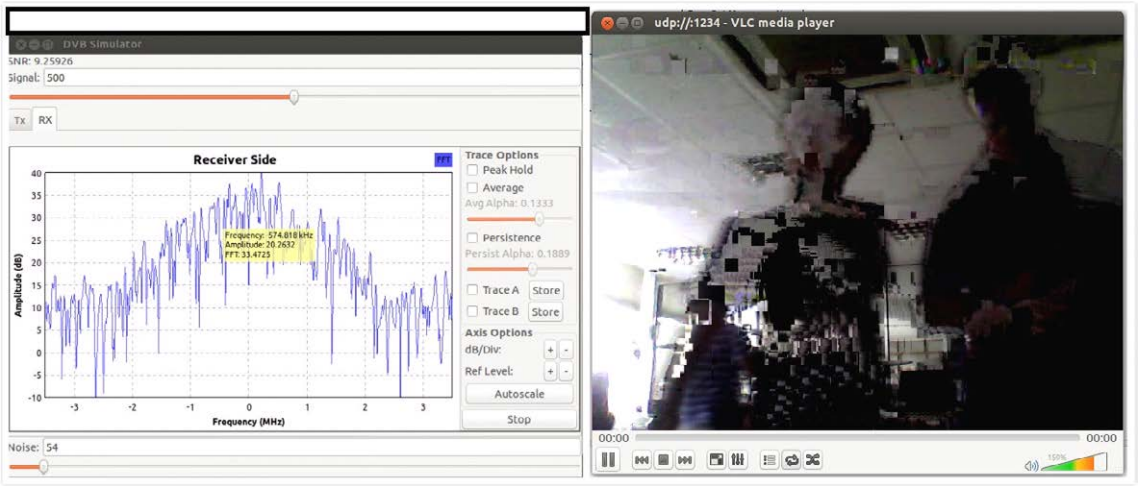


Figure 4.4.9 VLC captured video transmission when SNR<10

4.4.4. Problems

During the experiment, there are some problems can not be ignored as follow:

- Delay:

There is a 5~6 seconds delay during the video transmission in webcam mode. In order to exclude the reason like the format of video file or network delay, the video file which captured by the webcam was stored and transmitted in the file mode which was in the same network environment. Through the observation on the receiver side, it was found that there was no apparent delay on three different client devices. So the reason should related with the UNIX pipe. The camera capturing, the encoding and multiplexing by the VLC player, and the routing to GNU Radio jointly caused the delay. But 5~6 second normally can be accepted in real-time transmission.

- Network:

First, if the IP address for the destination of “UDP Sink” block was a privacy IP address which need a access code. The clients use this kind of IP addresses can’t receive any video stream. The transmission in public wireless network environment worked as normal. Second, when the network was unstable or poor quality, there will be a considerable amount of erroneous packets receiving or packets loss which coursed the video can not play or play sluggish.

- Device:

At the first beginning, this experiment was test on a laptop as the transmitter which has a “AMD A4” processor inside. And there is always a very serious delay and slow playback on the client device. When the simulation move to a computer which has a “Intel i5” processor, this kind of problem was solved. Additionally, the VLC player for mobile phone was just a beta version, some android phone can’t receive the video stream by UDP protocol. When there was serious noise or bad network, the playing on the phone may stop.

- Obstacle:

The last problem is not considered in this project, but based on wireless communication principle. The obstacle and distance would be a very serious problem which should be test and verify in the further experiment.

5 Conclusion and Future Work

5.1. Conclusion

Software Defined Radio is bound to bring about a technological revolution for the current wireless communication system. Excellent software GNU Radio and outstanding hardware USRP family products composed of a combination of ease of use, saving time, low cost but powerful Software Defined Radio platform. Through this thesis, SDR as a new concept combined with some popular modulation and demodulation techniques in wireless communication system are introduced, then some actual project was implemented based on GNU Radio and USRP: GSM scanning, Stereo FM radio receiver, OFDM signal transmission and observations and Real-time digital video broadcasting were achieved on a unified platform. More versatile and lower cost SDR platform will certainly be developed to apply to more practical applications in the wireless communications.

5.2. Future Work

Software Radio is the future trend of communication systems. So there is a great scope for improving the current project of wireless communication projects. The effective area of research and development work includes :

1. As the development of SDR platform, extending the project to increase the spectral efficiency and decrease BER based on advanced algorithms of channel coding and other novel communication concepts.

2. The cognitive radio represents an SDR with not only the ability to adapt to spectrum availability, protocols, and waveforms but the capability to learn waveforms and protocols, to adapt to local spectrum activity, and to learn the current needs of its user. And CR will also be capable of sensing, responding,and determining optimal responses to network and geographic operating conditions.

3. Implementation for more communication standards such as DVB-T, IEEE 802.11 a/g/p and LTE.

6 Reference

[1] Tuttlebee W H W. Software-defined radio: facets of a developing technology[J]. Personal Communications, IEEE, 1999, 6(2): 38-44.

[2] Dillinger M, Madani K, Alonistioti N. Software defined radio: Architectures, systems and functions[M]. Wiley, 2005.

[3] Ettus M. Universal software radio peripheral (USRP)[J]. Ettus Research LLC http://www. ettus. com, 2008.

[4] Chapman E, El Choueiry R, Jackson J, et al. Software defined radio[C]//Proceeding of KGCOE-MD2004: Multi-Disciplinary Engineering Design Conference. 2004.

[5] Rondeau T W, Le B, Maldonado D, et al. Cognitive radio formulation and implementation[C]//Cognitive Radio Oriented Wireless Networks and Communications, 2006. 1st International Conference on. IEEE, 2006: 1-10.

[6] Li Z, Xu W, Miller R, et al. Securing wireless systems via lower layer enforcements[C]//Proceedings of the 5th ACM workshop on Wireless security. ACM, 2006: 33-42.

[7] Watermeyer K. Design of a hardware platform for narrow-band Software Defined Radio applications[D]. University of Cape Town, 2007.

[8] Mate A, Lee K H, Lu I T. Spectrum sensing based on time covariance matrix using GNU radio and USRP for cognitive radio[C]//Systems, Applications and Technology Conference (LISAT), 2011 IEEE Long Island. IEEE, 2011: 1-6.

[9] Sarijari M A, Marwanto A, Fisal N, et al. Energy detection sensing based on GNU radio and USRP: An analysis study[C]//Communications (MICC), 2009 IEEE 9th Malaysia International Conference on. IEEE, 2009: 338-342.

[10] Zivkovic M, Auras D, Mathar R. OFDM-based dynamic spectrum access[C]//New Frontiers in Dynamic Spectrum, 2010 IEEE Symposium on. IEEE, 2010: 1-2.

[11] Braun M, Müller M, Fuhr M, et al. A USRP-based Testbed for OFDM-based Radar and Communication Systems[J].

[12] Selim A, Doyle L. Real-time interference reduction for OFDM-based dynamic spectrum access networks[C]//Dynamic Spectrum Access Networks (DYSPAN), 2012 IEEE International Symposium on. IEEE, 2012: 268-269.

[13] Mitola J. Software Radio: Wireless Architecture for the 21st Century[J]. Mitola's STATISfaction, ISBN 0-9671233-0-5.

[14] HPSDR Website: http://openhpsdr.org/.

[15] WebSDR Website: http://websdr.org/.

[16] Chirp Signals analyzed using SDR http://websdr.ewi.utwente.nl:8901/chirps/

[17] Ettus M. USRP User's and Developer's Guide[J]. Ettus Research LLC, 2005.

[18] "Using Open Source and Open Hardware Technologies": http://statist.h16.ru/how_build_interceptor.html.

[19] Raghavendra Rao, Qi Cheng, Aditya Kelkar, Dhaval Chaudhri, "Cooperative Cognitive Radio Network Testbed", 2011.

[20] Blossom, Eric. Exploring GNU Radio. 2009.

[21] Ettus Research LLC. URL http://www.ettus.com/order. October 2011.

[22] Wikipedia Cellular network: http://en.wikipedia.org/wiki/Cellular_network

[23] OpenBTS Website: http://www.openbts.org/

[24] Gqrx SDR receiver Website: http://www.oz9aec.net/index.php/gnu-radio/gqrx-sdr

[25] OSMOSDR Website: http://sdr.osmocom.org

[26] Ian Poole. "What is GMSK Modulation - Gaussian Minimum Shift Keying".

[27] Louis Litwin and Michal Pugel. "The Principles of OFDM".

[28] Mohamed Essam Khedr. EC74 4 Wireless Communication ,2008.

[29] David P. Reed. RFC 768 p1.

[30] Kurose J. F.; Ross K. W. Computer Networking: A Top-Down Approach (5th ed.). Boston, MA: Pearson Education. ISBN 978-0-13-136548-3, 2010.

[31] Clark, M.P. Data Networks IP and the Internet, 1st ed. West Sussex, England: John Wiley & Sons Ltd, 2003.

[32] John Carbone, "Speed Communications for Selected Applications with UDP". November, 2012.

[33] GNU Radio official website. http://gnuradio.org/.

[34] C++ Language tutorial. http://www.cplusplus.com/doc/tutorial/

[35] Python. Website: http://www.python.org/.

[36] Alex Verduin, "GNU Radio wireless protocol analysis approach". October, 2008.

[37] BBN Technologies Corp. "GNU Radio Architectural Changes".

[38] Dawei Shen, “Tutorial 8: Getting Prepared for Python in GNU Radio by Reading the FM Receiver Code Line by Line - Part II”. July, 2005.

[39] Firas Abbas ,Simple User Manual for Gnuradio 3.1.1.Free Software Foundation, Inc. 2007.

[40] A. Marwanto, M.A. Sarijari, N. Fisal, S.K.S. Yusof, and R.A. Rashid, Experimental study of OFDM implementation utilizing GNU Radio and USRP - SDR," Communications (MICC), IEEE 9th Malaysia International Conference on , vol., no., pp.132-135, 15-17 Dec. 2009.

[41] L. Yang, W. Hou, L. Cao, B. Y. Zhao, and H. Zheng, "Supporting demanding wireless applications with frequency-agile radios," in Proceeding of the 7th NSDI, 2010.

[42] Isla Hernandez, Sergio. Simulation and Evaluation of a DVB system using simulink (Vol I). 2005

[43] Alexandru Csete,Simple DVB with Gstreamer and GNU Radio:

http://www.oz9aec.net.

[44] Asha Mariam Iype and Shashanka C. D. Video transmission using USRP. 2011.

[45] Video LAN team tutorial, “Chapter 3. Advanced streaming using the command line.”

APPENDIX A

USRP B100 Datasheet:

Ettus Research

USRP™ B100 BUS SERIES

FEATURES:

- Use with GNU Radio
- Modular Architecture: DC-6 GHz
- Dual 64 MS/s, 12-bit ADC
- Dual 128 MS/s, 14-bit DAC
- DDC/DUC w/ 15 mHz Resolution
- USB 2.0 Interface to Host
- Spartan 3A-1400 FPGA
- Supported by Free Xilinx Design Tools
- Auxiliary Digital and Analog I/O
- 2.5 ppm TCXO Frequency Reference
- Configurable Reference Clock Frequency
- 1 PPS and 10 MHz Reference Inputs

B100 PRODUCT OVERVIEW:

The Ettus Research USRP™ B100 is a member of the USRP™ (Universal Software Radio Peripheral) family of products, which enables engineers to rapidly design and implement powerful, flexible software radio systems. The B100 hardware provides low-cost RF processing capability, and up to 16 MS/s of signal streaming through the USB 2.0 host interface.

The USRP B100 is an ideal model for users that require an entry-level software defined radio device for cost-sensitive applications. Utilizing the USB 2.0 interface, users can get the B100 up and running quickly. A reconfigurable clock also allows users to more easily target specific applications.

The USRP Hardware Driver™ is the official driver for all Ettus Research products, and supports rapid development in a comprehensive environment. The USRP Hardware Driver supports Linux, Mac OSX and Windows. UHD allows portability across the entire USRP product line, enabling application migration to higher performance platforms such as the USRP N200/N210.

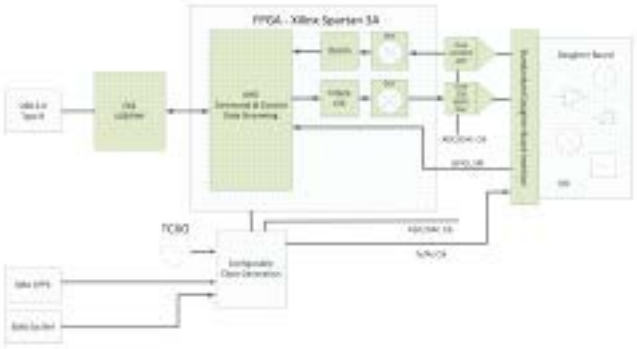
USRP™ B100 BUS SERIES

SPECIFICATIONS

Spec	Typ.	Unit
POWER		
DC Input	6	V
Current Consumption	0.6	A
w/ WBX Daughterboard	1.6	A
CONVERSION PERFORMANCE AND CLOCKS		
ADC Sample Rate	64	MS/s
ADC Resolution	12	bits
ADC Wideband SFDR	83	dBc
DAC Sample Rate	128	MS/s
DAC Resolution	14	bits
DAC Wideband SFDR	83	dBc
Host Sample Rate (8b/16b)	16/8	MS/s
Frequency Accuracy	2.5	ppm

Spec	Typ.	Unit
RF PERFORMANCE (W/ WBX)		
SSB/LO Suppression	35/50	dBc
Phase Noise (1.8 Ghz)		
10 kHz	-80	dBc/Hz
100 kHz	-100	dBc/Hz
1 MHz	-137	dBc/Hz
Power Output	15	dBm
IIP3	0	dBm
Receive Noise Figure	5	dB
PHYSICAL		
Operating Temperature	0 to 55°	C
Dimensions (l x w x h)	22 x 16 x 5	cm
Weight	1.2	kg

* All specifications are subject to change without notice.



Ettus Research™
A National Instruments Company

ABOUT ETTUS RESEARCH:

Ettus Research is an innovative provider of software defined radio hardware, including the original Universal Software Radio Peripheral (USRP) family of products. Ettus Research products maintain support from a variety of software frameworks, including GNU Radio. Ettus Research is a leader in the GNU Radio open-source community, and enables users worldwide to address a wide range of research, industry and defense applications. The company was founded in 2004 and is based in Mountain View, California. As of 2010, Ettus Research is a wholly owned subsidiary of National Instruments.

1043 North Shoreline Blvd
Suite 100
Mountain View, CA 94043
P 650.967.2870 www.ettus.com
F 866.807.9801

APPENDIX B

Python code for Stereo FM receiver:

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: Stereo FM Receiver
# Description: WFM receiver
# Generated: Thu Jun 27 17:39:45 2013
##################################################

from gnuradio import audio
from gnuradio import blks2
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import uhd
from gnuradio import window
from gnuradio.eng_option import eng_option
from gnuradio.gr import firdes
from gnuradio.wxgui import fftsink2
from gnuradio.wxgui import forms
from gnuradio.wxgui import scopesink2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class wfm_rx_pll(grc_wxgui.top_block_gui):

	def __init__(self):
		grc_wxgui.top_block_gui.__init__(self, title="Stereo FM Receiver")
		_icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
		self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

		##################################################
		# Variables
		##################################################
		self.decim = decim = 128
		self.xlate_tune = xlate_tune = 0
		self.usrp_freq = usrp_freq = 91e6
		self.samp_rate = samp_rate = 64e6/decim
		self.rx_freq = rx_freq = usrp_freq+xlate_tune
		self.rf_gain = rf_gain = 15
		self.filter_taps = filter_taps = firdes.low_pass(1, samp_rate, 250000, 20000, firdes.WIN_HAMMING,
6.76)
		self.af_gain = af_gain = 3

		##################################################
		# Blocks
		##################################################
		_xlate_tune_sizer = wx.BoxSizer(wx.VERTICAL)
		self._xlate_tune_text_box = forms.text_box(
			parent=self.GetWin(),
			sizer=_xlate_tune_sizer,
			value=self.xlate_tune,
			callback=self.set_xlate_tune,
			label="Fine frequency",
			converter=forms.float_converter(),
			proportion=0,
		)
		self._xlate_tune_slider = forms.slider(
			parent=self.GetWin(),
			sizer=_xlate_tune_sizer,
			value=self.xlate_tune,
			callback=self.set_xlate_tune,
			minimum=-250e3,
			maximum=250e3,
			num_steps=500,
			style=wx.SL_HORIZONTAL,
			cast=float,
			proportion=1,
		)
		self.GridAdd(_xlate_tune_sizer, 7, 0, 1, 5)
```

```
_usrp_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._usrp_freq_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_usrp_freq_sizer,
        value=self.usrp_freq,
        callback=self.set_usrp_freq,
        label="USRP frequency",
        converter=forms.float_converter(),
        proportion=0,
)
self._usrp_freq_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_usrp_freq_sizer,
        value=self.usrp_freq,
        callback=self.set_usrp_freq,
        minimum=76e6,
        maximum=108e6,
        num_steps=200,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
)
self.GridAdd(_usrp_freq_sizer, 6, 0, 1, 5)
_rf_gain_sizer = wx.BoxSizer(wx.VERTICAL)
self._rf_gain_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_rf_gain_sizer,
        value=self.rf_gain,
        callback=self.set_rf_gain,
        label="RF",
        converter=forms.float_converter(),
        proportion=0,
)
self._rf_gain_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_rf_gain_sizer,
        value=self.rf_gain,
        callback=self.set_rf_gain,
        minimum=0,
        maximum=50,
        num_steps=50,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
)
self.GridAdd(_rf_gain_sizer, 8, 0, 1, 2)
self.nbook = self.nbook = wx.Notebook(self.GetWin(), style=wx.NB_TOP)
self.nbook.AddPage(grc_wxgui.Panel(self.nbook), "Receiver")
self.nbook.AddPage(grc_wxgui.Panel(self.nbook), "Demod")
self.nbook.AddPage(grc_wxgui.Panel(self.nbook), "Scope")
self.GridAdd(self.nbook, 0, 0, 5, 5)
_af_gain_sizer = wx.BoxSizer(wx.VERTICAL)
self._af_gain_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_af_gain_sizer,
        value=self.af_gain,
        callback=self.set_af_gain,
        label="AF",
        converter=forms.float_converter(),
        proportion=0,
)
self._af_gain_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_af_gain_sizer,
        value=self.af_gain,
        callback=self.set_af_gain,
        minimum=0,
        maximum=10,
        num_steps=100,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
)
self.GridAdd(_af_gain_sizer, 8, 2, 1, 2)
self.xlating_fir_filter = gr.freq_xlating_fir_filter_ccc(1, (filter_taps), -xlate_tune, samp_rate)
self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
```

```
                self.nbook.GetPage(2).GetWin(),
                title="Scope Plot",
                sample_rate=samp_rate/2,
                v_scale=0,
                v_offset=0,
                t_scale=0,
                ac_couple=False,
                xy_mode=False,
                num_inputs=1,
                trig_mode=gr.gr_TRIG_MODE_AUTO,
                y_axis_label="Counts",
        )
        self.nbook.GetPage(2).Add(self.wxgui_scopesink2_0.win)
        self.wxgui_fftsink2_0 = fftsink2.fft_sink_f(
                self.nbook.GetPage(1).GetWin(),
                baseband_freq=0,
                y_per_div=10,
                y_divs=10,
                ref_level=0,
                ref_scale=2.0,
                sample_rate=samp_rate/2,
                fft_size=512,
                fft_rate=15,
                average=False,
                avg_alpha=None,
                title="Demod",
                peak_hold=False,
        )
        self.nbook.GetPage(1).Add(self.wxgui_fftsink2_0.win)
        self.wfm_rcv_pll = blks2.wfm_rcv_pll(
                demod_rate=samp_rate,
                audio_decimation=10,
        )
        self.uhd_usrp_source_0 = uhd.usrp_source(
                device_addr="",
                stream_args=uhd.stream_args(
                        cpu_format="fc32",
                        channels=range(1),
                ),
        )
        self.uhd_usrp_source_0.set_samp_rate(samp_rate)
        self.uhd_usrp_source_0.set_center_freq(usrp_freq, 0)
        self.uhd_usrp_source_0.set_gain(rf_gain, 0)
        self.uhd_usrp_source_0.set_antenna("TX/RX", 0)
        self._rx_freq_static_text = forms.static_text(
                parent=self.GetWin(),
                value=self.rx_freq,
                callback=self.set_rx_freq,
                label="Receive",
                converter=forms.float_converter(),
        )
        self.GridAdd(self._rx_freq_static_text, 5, 3, 1, 1)
        self.rr_stereo_right = blks2.rational_resampler_fff(
                interpolation=48,
                decimation=50,
                taps=None,
                fractional_bw=None,
        )
        self.rr_stereo_left = blks2.rational_resampler_fff(
                interpolation=48,
                decimation=50,
                taps=None,
                fractional_bw=None,
        )
        self.fftsink_rf = fftsink2.fft_sink_c(
                self.nbook.GetPage(0).GetWin(),
                baseband_freq=0,
                y_per_div=10,
                y_divs=10,
                ref_level=0,
                ref_scale=13490.0,
                sample_rate=samp_rate/2,
                fft_size=512,
                fft_rate=10,
                average=True,
                avg_alpha=0.5,
```

```
                title="Baseband",
                peak_hold=False,
                size=(1120,527),
        )
        self.nbook.GetPage(0).Add(self.fftsink_rf.win)
        self.audio_sink = audio.sink(48000, "", True)
        self.af_gain_stereo_right = gr.multiply_const_vff((af_gain, ))
        self.af_gain_stereo_left = gr.multiply_const_vff((af_gain, ))

        ##################################################
        # Connections
        ##################################################
        self.connect((self.xlating_fir_filter, 0), (self.fftsink_rf, 0))
        self.connect((self.xlating_fir_filter, 0), (self.wfm_rcv_pll, 0))
        self.connect((self.af_gain_stereo_right, 0), (self.audio_sink, 1))
        self.connect((self.rr_stereo_right, 0), (self.af_gain_stereo_left, 0))
        self.connect((self.rr_stereo_left, 0), (self.af_gain_stereo_right, 0))
        self.connect((self.wfm_rcv_pll, 0), (self.rr_stereo_right, 0))
        self.connect((self.wfm_rcv_pll, 1), (self.rr_stereo_left, 0))
        self.connect((self.uhd_usrp_source_0, 0), (self.xlating_fir_filter, 0))
        self.connect((self.af_gain_stereo_left, 0), (self.audio_sink, 0))
        self.connect((self.wfm_rcv_pll, 1), (self.wxgui_fftsink2_0, 0))
        self.connect((self.wfm_rcv_pll, 0), (self.wxgui_scopesink2_0, 0))


def get_decim(self):
        return self.decim

def set_decim(self, decim):
        self.decim = decim
        self.set_samp_rate(64e6/self.decim)

def get_xlate_tune(self):
        return self.xlate_tune

def set_xlate_tune(self, xlate_tune):
        self.xlate_tune = xlate_tune
        self._xlate_tune_slider.set_value(self.xlate_tune)
        self._xlate_tune_text_box.set_value(self.xlate_tune)
        self.set_rx_freq(self.usrp_freq+self.xlate_tune)
        self.xlating_fir_filter.set_center_freq(-self.xlate_tune)

def get_usrp_freq(self):
        return self.usrp_freq

def set_usrp_freq(self, usrp_freq):
        self.usrp_freq = usrp_freq
        self.set_rx_freq(self.usrp_freq+self.xlate_tune)
        self.uhd_usrp_source_0.set_center_freq(self.usrp_freq, 0)
        self._usrp_freq_slider.set_value(self.usrp_freq)
        self._usrp_freq_text_box.set_value(self.usrp_freq)

def get_samp_rate(self):
        return self.samp_rate

def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.fftsink_rf.set_sample_rate(self.samp_rate/2)
        self.wxgui_scopesink2_0.set_sample_rate(self.samp_rate/2)
        self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)
        self.wxgui_fftsink2_0.set_sample_rate(self.samp_rate/2)
        self.set_filter_taps(firdes.low_pass(1, self.samp_rate, 250000, 20000, firdes.WIN_HAMMING, 6.76))

def get_rx_freq(self):
        return self.rx_freq

def set_rx_freq(self, rx_freq):
        self.rx_freq = rx_freq
        self._rx_freq_static_text.set_value(self.rx_freq)

def get_rf_gain(self):
        return self.rf_gain

def set_rf_gain(self, rf_gain):
        self.rf_gain = rf_gain
        self._rf_gain_slider.set_value(self.rf_gain)
```

```
                self._rf_gain_text_box.set_value(self.rf_gain)
                self.uhd_usrp_source_0.set_gain(self.rf_gain, 0)

        def get_filter_taps(self):
                return self.filter_taps

        def set_filter_taps(self, filter_taps):
                self.filter_taps = filter_taps
                self.xlating_fir_filter.set_taps((self.filter_taps))

        def get_af_gain(self):
                return self.af_gain

        def set_af_gain(self, af_gain):
                self.af_gain = af_gain
                self._af_gain_slider.set_value(self.af_gain)
                self._af_gain_text_box.set_value(self.af_gain)
                self.af_gain_stereo_left.set_k((self.af_gain, ))
                self.af_gain_stereo_right.set_k((self.af_gain, ))

if __name__ == '__main__':
        parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
        (options, args) = parser.parse_args()
        tb = wfm_rx_pll()
        tb.Run(True)
```

APPENDIX C

Python code for OFDM simulation:

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: OFDM simulation
# Author: Lei
# Generated: Tue Jun 25 20:44:45 2013
##################################################

from gnuradio import analog
from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import window
from gnuradio.eng_option import eng_option
from gnuradio.gr import firdes
from gnuradio.wxgui import fftsink2
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import numpy
import wx

class sim_ofdm(grc_wxgui.top_block_gui):

	def __init__(self):
		grc_wxgui.top_block_gui.__init__(self, title="OFDM simulation")
		_icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
		self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

		##################################################
		# Variables
		##################################################
		self.samp_rate = samp_rate = 500e3

		##################################################
		# Blocks
		##################################################
		self.wxgui_fftsink2_0_0 = fftsink2.fft_sink_c(
			self.GetWin(),
			baseband_freq=0,
			y_per_div=10,
			y_divs=10,
			ref_level=0,
			ref_scale=2.0,
			sample_rate=samp_rate,
			fft_size=1024,
			fft_rate=15,
			average=True,
			avg_alpha=None,
			title="FFT Plot",
			peak_hold=False,
		)
		self.Add(self.wxgui_fftsink2_0_0.win)
		self.wxgui_fftsink2_0 = fftsink2.fft_sink_c(
			self.GetWin(),
			baseband_freq=0,
			y_per_div=10,
			y_divs=10,
			ref_level=0,
			ref_scale=2.0,
			sample_rate=samp_rate,
			fft_size=1024,
			fft_rate=15,
			average=True,
			avg_alpha=None,
			title="FFT Plot",
			peak_hold=False,
		)
```

```
                self.Add(self.wxgui_fftsink2_0.win)
                self.random_source_x_0 = gr.vector_source_s(map(int, numpy.random.randint(0, 2, 1000)), True)
                self.gr_short_to_float_0 = gr.short_to_float(1, 1)
                self.gr_float_to_short_0 = gr.float_to_short(1, 1)
                self.gr_file_sink_1 = gr.file_sink(gr.sizeof_short*1, "/home/ubuntu/Desktop/ofdm1.txt")
                self.gr_file_sink_1.set_unbuffered(False)
                self.gr_add_xx_0 = gr.add_vcc(1)
                self.digital_ofdm_mod_0 = grc_blks2.packet_mod_f(digital.ofdm_mod(
                                options=grc_blks2.options(
                                                modulation="qpsk",
                                                fft_length=1024,
                                                occupied_tones=100,
                                                cp_length=128,
                                                pad_for_usrp=True,
                                                log=None,
                                                verbose=None,
                                ),
                        ),
                        payload_length=0,
                )
                self.digital_ofdm_demod_0 = grc_blks2.packet_demod_f(digital.ofdm_demod(
                                options=grc_blks2.options(
                                                modulation="qpsk",
                                                fft_length=512,
                                                occupied_tones=200,
                                                cp_length=128,
                                                snr=10,
                                                log=None,
                                                verbose=None,
                                ),
                                callback=lambda ok, payload: self.digital_ofdm_demod_0.recv_pkt(ok, payload),
                        ),
                )
                self.blocks_throttle_0 = blocks.throttle(gr.sizeof_gr_complex*1, samp_rate)
                self.analog_noise_source_x_0 = analog.noise_source_c(analog.GR_GAUSSIAN, 0, 0)

                ##################################################
                # Connections
                ##################################################
                self.connect((self.random_source_x_0, 0), (self.gr_short_to_float_0, 0))
                self.connect((self.gr_short_to_float_0, 0), (self.digital_ofdm_mod_0, 0))
                self.connect((self.blocks_throttle_0, 0), (self.wxgui_fftsink2_0, 0))
                self.connect((self.digital_ofdm_mod_0, 0), (self.blocks_throttle_0, 0))
                self.connect((self.blocks_throttle_0, 0), (self.gr_add_xx_0, 0))
                self.connect((self.analog_noise_source_x_0, 0), (self.gr_add_xx_0, 1))
                self.connect((self.gr_add_xx_0, 0), (self.digital_ofdm_demod_0, 0))
                self.connect((self.digital_ofdm_demod_0, 0), (self.gr_float_to_short_0, 0))
                self.connect((self.gr_float_to_short_0, 0), (self.gr_file_sink_1, 0))
                self.connect((self.gr_add_xx_0, 0), (self.wxgui_fftsink2_0_0, 0))


        def get_samp_rate(self):
                return self.samp_rate

        def set_samp_rate(self, samp_rate):
                self.samp_rate = samp_rate
                self.blocks_throttle_0.set_sample_rate(self.samp_rate)
                self.wxgui_fftsink2_0.set_sample_rate(self.samp_rate)
                self.wxgui_fftsink2_0_0.set_sample_rate(self.samp_rate)

if __name__ == '__main__':
        parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
        (options, args) = parser.parse_args()
        tb = sim_ofdm()
        tb.Run(True)
```

APPENDIX D

Python code for Real-time DVB simulation:

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: DVB Simulator
# Author: Lei
# Generated: Thu Jun 27 17:38:51 2013
##################################################

from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import window
from gnuradio.eng_option import eng_option
from gnuradio.gr import firdes
from gnuradio.wxgui import fftsink2
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class gmsk_sim(grc_wxgui.top_block_gui):

	def __init__(self):
		grc_wxgui.top_block_gui.__init__(self, title="DVB Simulator ")
		_icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
		self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

		##################################################
		# Variables
		##################################################
		self.signal = signal = 500
		self.noise = noise = 10
		self.variable_static_text_0 = variable_static_text_0 = signal/noise
		self.samp_rate = samp_rate = 8e6

		##################################################
		# Blocks
		##################################################
		_signal_sizer = wx.BoxSizer(wx.VERTICAL)
		self._signal_text_box = forms.text_box(
			parent=self.GetWin(),
			sizer=_signal_sizer,
			value=self.signal,
			callback=self.set_signal,
			label="Signal",
			converter=forms.float_converter(),
			proportion=0,
		)
		self._signal_slider = forms.slider(
			parent=self.GetWin(),
			sizer=_signal_sizer,
			value=self.signal,
			callback=self.set_signal,
			minimum=0,
			maximum=1000,
			num_steps=1000,
			style=wx.SL_HORIZONTAL,
			cast=float,
			proportion=1,
		)
		self.Add(_signal_sizer)
		self.notebook = self.notebook = wx.Notebook(self.GetWin(), style=wx.NB_TOP)
		self.notebook.AddPage(grc_wxgui.Panel(self.notebook), "Tx")
		self.notebook.AddPage(grc_wxgui.Panel(self.notebook), "RX")
		self.Add(self.notebook)
		_noise_sizer = wx.BoxSizer(wx.VERTICAL)
		self._noise_text_box = forms.text_box(
			parent=self.GetWin(),
```

```
                sizer=_noise_sizer,
                value=self.noise,
                callback=self.set_noise,
                label="Noise",
                converter=forms.float_converter(),
                proportion=0,
        )
        self._noise_slider = forms.slider(
                parent=self.GetWin(),
                sizer=_noise_sizer,
                value=self.noise,
                callback=self.set_noise,
                minimum=0,
                maximum=1000,
                num_steps=1000,
                style=wx.SL_HORIZONTAL,
                cast=float,
                proportion=1,
        )
        self.Add(_noise_sizer)
        self.wxgui_fftsink2_0_0_0 = fftsink2.fft_sink_c(
                self.notebook.GetPage(0).GetWin(),
                baseband_freq=0,
                y_per_div=10,
                y_divs=10,
                ref_level=0,
                ref_scale=2.0,
                sample_rate=samp_rate,
                fft_size=512,
                fft_rate=15,
                average=False,
                avg_alpha=None,
                title="Transmitter Side",
                peak_hold=False,
        )
        self.notebook.GetPage(0).Add(self.wxgui_fftsink2_0_0_0.win)
        self.wxgui_fftsink2_0_0 = fftsink2.fft_sink_c(
                self.notebook.GetPage(1).GetWin(),
                baseband_freq=0,
                y_per_div=10,
                y_divs=10,
                ref_level=0,
                ref_scale=2.0,
                sample_rate=samp_rate,
                fft_size=512,
                fft_rate=15,
                average=False,
                avg_alpha=None,
                title="Receiver Side",
                peak_hold=False,
        )
        self.notebook.GetPage(1).Add(self.wxgui_fftsink2_0_0.win)
        self._variable_static_text_0_static_text = forms.static_text(
                parent=self.GetWin(),
                value=self.variable_static_text_0,
                callback=self.set_variable_static_text_0,
                label="SNR",
                converter=forms.float_converter(),
        )
        self.GridAdd(self._variable_static_text_0_static_text, 0, 0, 1, 1)
        self.gr_udp_sink_0_2_1_0 = gr.udp_sink(gr.sizeof_char*1, "138.100.50.70", 1234, 1472, True)
        self.gr_udp_sink_0_2_1 = gr.udp_sink(gr.sizeof_char*1, "138.100.50.72", 1234, 1472, True)
        self.gr_udp_sink_0_2_0 = gr.udp_sink(gr.sizeof_char*1, "138.100.50.74", 1234, 1472, True)
        self.gr_udp_sink_0_2 = gr.udp_sink(gr.sizeof_char*1, "138.100.50.73", 1234, 1472, True)
        self.gr_throttle_0_0 = gr.throttle(gr.sizeof_char*1, samp_rate)
        self.gr_throttle_0 = gr.throttle(gr.sizeof_gr_complex*1, samp_rate)
        self.gr_noise_source_x_0 = gr.noise_source_c(gr.GR_GAUSSIAN, noise, 42)
        self.gr_multiply_const_vxx_0 = gr.multiply_const_vcc((signal, ))
        self.gr_file_source_0 = gr.file_source(gr.sizeof_char*1, "/home/ubuntu/Desktop/tx.ts", True)
        self.gr_file_sink_1 = gr.file_sink(gr.sizeof_char*1, "/home/ubuntu/Desktop/test2.ts")
        self.gr_file_sink_1.set_unbuffered(False)
        self.gr_file_sink_0 = gr.file_sink(gr.sizeof_char*1, "/home/ubuntu/Desktop/tx.ts")
        self.gr_file_sink_0.set_unbuffered(False)
        self.gr_add_xx_0 = gr.add_vcc(1)
        self.digital_gmsk_mod_0 = digital.gmsk_mod(
                samples_per_symbol=2,
```

```
                bt=0.35,
                verbose=False,
                log=False,
        )
        self.digital_gmsk_demod_0 = digital.gmsk_demod(
                samples_per_symbol=2,
                gain_mu=0.175,
                mu=0.5,
                omega_relative_limit=0.005,
                freq_error=0.0,
                verbose=False,
                log=False,
        )
        self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
                        samples_per_symbol=2,
                        bits_per_symbol=1,
                        access_code="",
                        pad_for_usrp=True,
                ),
                payload_length=0,
        )
        self.blks2_packet_decoder_0 = grc_blks2.packet_demod_b(grc_blks2.packet_decoder(
                        access_code="",
                        threshold=-1,
                        callback=lambda ok, payload: self.blks2_packet_decoder_0.recv_pkt(ok,
payload),
                ),
        )

        ##################################################
        # Connections
        ##################################################
        self.connect((self.gr_multiply_const_vxx_0, 0), (self.gr_throttle_0, 0))
        self.connect((self.gr_add_xx_0, 0), (self.digital_gmsk_demod_0, 0))
        self.connect((self.gr_noise_source_x_0, 0), (self.gr_add_xx_0, 1))
        self.connect((self.gr_throttle_0_0, 0), (self.gr_file_sink_1, 0))
        self.connect((self.blks2_packet_encoder_0, 0), (self.digital_gmsk_mod_0, 0))
        self.connect((self.digital_gmsk_demod_0, 0), (self.blks2_packet_decoder_0, 0))
        self.connect((self.gr_throttle_0, 0), (self.gr_add_xx_0, 0))
        self.connect((self.gr_add_xx_0, 0), (self.wxgui_fftsink2_0_0, 0))
        self.connect((self.gr_file_source_0, 0), (self.blks2_packet_encoder_0, 0))
        self.connect((self.gr_file_source_0, 0), (self.gr_file_sink_0, 0))
        self.connect((self.blks2_packet_decoder_0, 0), (self.gr_throttle_0_0, 0))
        self.connect((self.gr_throttle_0, 0), (self.wxgui_fftsink2_0_0_0, 0))
        self.connect((self.digital_gmsk_mod_0, 0), (self.gr_multiply_const_vxx_0, 0))
        self.connect((self.gr_throttle_0_0, 0), (self.gr_udp_sink_0_2, 0))
        self.connect((self.gr_throttle_0_0, 0), (self.gr_udp_sink_0_2_0, 0))
        self.connect((self.gr_throttle_0_0, 0), (self.gr_udp_sink_0_2_1, 0))
        self.connect((self.gr_throttle_0_0, 0), (self.gr_udp_sink_0_2_1_0, 0))


    def get_signal(self):
        return self.signal

    def set_signal(self, signal):
        self.signal = signal
        self.gr_multiply_const_vxx_0.set_k((self.signal, ))
        self.set_variable_static_text_0(self.signal/self.noise)
        self._signal_slider.set_value(self.signal)
        self._signal_text_box.set_value(self.signal)

    def get_noise(self):
        return self.noise

    def set_noise(self, noise):
        self.noise = noise
        self.gr_noise_source_x_0.set_amplitude(self.noise)
        self.set_variable_static_text_0(self.signal/self.noise)
        self._noise_slider.set_value(self.noise)
        self._noise_text_box.set_value(self.noise)

    def get_variable_static_text_0(self):
        return self.variable_static_text_0

    def set_variable_static_text_0(self, variable_static_text_0):
        self.variable_static_text_0 = variable_static_text_0
```

```
                self._variable_static_text_0_static_text.set_value(self.variable_static_text_0)

        def get_samp_rate(self):
                return self.samp_rate

        def set_samp_rate(self, samp_rate):
                self.samp_rate = samp_rate
                self.wxgui_fftsink2_0_0.set_sample_rate(self.samp_rate)
                self.gr_throttle_0.set_sample_rate(self.samp_rate)
                self.wxgui_fftsink2_0_0_0.set_sample_rate(self.samp_rate)
                self.gr_throttle_0_0.set_sample_rate(self.samp_rate)

if __name__ == '__main__':
        parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
        (options, args) = parser.parse_args()
        tb = gmsk_sim()
        tb.Run(True)
```