THE UNIVERSITY OF HULL

"Design of Software Radio"

being a Dissertation submitted in partial fulfilment of

the requirements for the Degree of

MSc. Wireless Systems and Logistics Technology

in the University of Hull

by

Morine Mashoma Sithole, Bachelor of Engineering in Electronic Engineering (BEng)

First Supervisor:	**Mr Nick G. Riley**
Second Supervisor:	**Dr Kevin S. Paulson**

(September, 2010)

Abstract

Software Defined Radio (SDR) has become a prevalent technology in the wireless systems industry. In SDR, some or all of the signal-specific handling is implemented as software functions, while other functions like decimation, interpolation, digital up-conversion and digital down-conversion, are done on a reprogrammable Digital Signal Processor or Field Programmable Gate Array. This readily enables real-time implementation and reconfiguration of high-throughput radio systems.

The objective of this project was to provide the postgraduate students at Hull University with a platform to learn the concepts that apply to the field of Software Defined Radio, and to learn the specialized digital signal processing techniques used in wireless systems. A total of twelve laboratory exercises were designed to lead the student through the process of using the Universal Software Radio Peripheral (USRP2) hardware and the GNU Radio open source software to perform various digital signal processing tasks, and then finally design software radios that employ the Gaussian Minimum Shift Keying, Differential Phase Shift Keying and Direct Sequence Spread Spectrum modulation schemes.

In this report, a literature review is given of the developments made in software radio technology. A brief theory of digital signal processing is provided. A description of the laboratory configuration is then presented, and finally, the possible directions of future work are discussed.

Acknowledgements

Firstly, I would like to thank my first supervisor, Mr N. G. Riley for his guidance and supervision during the course of the project. Secondly, I would like to extend my appreciation to my second supervisor, Dr K. S. Paulson for all the assistance he provided. Thirdly, I would like to thank the University of Hull for financing the project.

I am also deeply indebted to my sponsor, Canon Collins Trust, for awarding me the opportunity to further my studies in order to make a difference in my home country and region.

Last, but not least, I would like to thank my parents, my family and all my friends for all the support they gave me.

Contents

List of Figures

List of Tables

Abbreviations

3G	Third Generation
ADC	Analogue to Digital Converter
ALU	Arithmetic–Logic Unit
AM	Amplitude Modulation
ASIC	Application-Specific Integrated Circuit
ASK	Amplitude Shift Keying
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPF	Band pass filter
BASK	Binary Amplitude Shift Keying
BFSK	Binary Frequency Shift Keying
BPSK	Binary Phase Shift Keying
CIC	Cascaded Integrator-Comb
DAC	Digital to Analogue Converter
DBPSK	Differential Binary Phase Shift Keying
DFT	Discrete Fourier Transform
DPSK	Differential Phase Shift Keying
DQPSK	Differential Quadrature Phase Shift Keying
DSP	Digital Signal Processor
DSSS	Direct Sequence Spread Spectrum
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FS	Frequency Sampling
FSK	Frequency Shift Keying
Gbps	Giga bits per second
GBps	Giga bytes per second
GMSK	Gaussian Minimum Shift Keying
GPP	General Purpose Processor
GRC	GNU Radio Companion
GSM	Global System for Mobile Communications

GUI	Graphical User Interface
HDD	Hard Disk Drive
HPF	High Pass Filter
IF	Intermediate Frequency
IIR	Infinite Impulse Response
ISM band	Industrial, Scientific and Medical band
LFSR	Linear-Feedback Shift Register
LPF	Low pass filter
LTE	Long Term Evolution
MAC	Multiply-accumulate
Mbps	Mega bits per second
MBps	Mega bytes per second
MFLOP	Million Floating Point Operations Per Second
MIMO	Multiple-input multiple-output
MIPS	Million Instructions Per Second
MSps	Million Samples per second
MMAC	Million MAC Operations Per Second
OS	Operating System
PC	Personal Computer
PCIe	Peripheral Component Interconnect Express
PLL	Phase-Locked Loop
PSK	Phase Shift Keying
QPSK	Quadrature Phase Shift Keying
RAM	Random Access Memory
RF	Radio Frequency
SDR	Software Defined Radio
SER	Symbol Error Rate
SFDR	Spurious Free Dynamic Range
SNR	Signal to Noise Ratio
USRP	Universal Software Radio Peripheral
USRP2	The second version of the original USRP
VHF	Very High Frequency
VLIW	Very Long Instruction Word
VLSI	Very Large Scale Integration

Chapter One

1. Introduction

The past years have seen a shift away from hardware-oriented radio terminals towards software-oriented radio terminals. Software Defined Radio (SDR), or simply Software Radio, refers to radio systems constructed using minimal hardware at the frontend. The backend is implemented in software, thus enabling parameters such as the operating frequency range, the modulation scheme and the maximum radiated power to be easily adjusted without making any changes to the frontend hardware. SDRs facilitate the interworking of different radio access systems and improve spectral efficiency.

1.1 Project objectives

The objective of this project was to design a Software Defined Radio (SDR) system that will support wireless communications research at the University of Hull. Because digital communications theory is innately mathematical in nature, it remains an abstract concept to students and hands-on laboratory experiences will help students to understand the concepts easier. SDR is a technology that continues to evolve, and is therefore, directly relevant to future communications engineers.

The proposed solution was implemented using the Ettus Research Universal Software Radio Peripheral package (USRP2), and the open source software, GNU Radio. The USRP2 was chosen because it is capable of handling signals consistent with various industry standard radio systems, such as GSM, 3G and LTE systems. All the signal processing is visible to the student in the GNU Radio development environment, making it easier to understand SDR system development. The USRP2 and GNU radio were also chosen because they best fulfilled the output power, size and cost constraints of the project. Programming assignments were designed to introduce students to the USRP2 architecture and GNU Radio Software and finally to design software radios that make use of the Gaussian Minimum Shift Keying, Differential Phase Shift Keying and Direct Sequence Spread Spectrum modulation schemes.

1.2 Chapter Summary

Chapter two of this report provides background on the developments made in software radio technology. This is followed by a description of the hardware and software used to develop the SDR system in chapter three. Chapter four presents the procedure followed in designing and testing the system. Chapter five describes the experimental results. Chapter six presents the conclusions and summarizes the developments accomplished in the thesis. Recommendations regarding future work are also suggested in the same chapter. Appendix A outlines the installation procedure for Linux Mandriva operating system and GNU Radio software. Appendix B provides the list of all the dependency packages required by GNU Radio. Appendix C is a source code listing of the PYTHON scripts developed for each simulation and laboratory exercise. Appendix D is a compilation of the laboratory assignments. Appendix E shows the project Gantt chart.

Chapter two

2 Literature Review

2.1 Introduction

This literature review gives an overview of the progressive role that digital signal processing (DSP) technology played in the research and commercial development of SDR. The methods used in DSP education since the inception of the technology are also examined.

2.2 Definitions

DSPs are essentially hardware that is designed to process the continuous signals that come from, and go back into, the real world by performing multiply-accumulate (MAC) functions in real-time (Frantz, 2000). It is this feature which differentiates DSPs from other microprocessors. The advancement of DSP technology over the past few decades brought about the concept of SDR. SDR is the term used to describe a radio communication system where the analogue conversion of signals is made as close as possible to the antenna and functions that are typically implemented in electronic hardware, such as filtering, modulation, demodulation and error correction, are implemented in software (Reed, 2002; Gunn, Barron and Ruczczyk, 1999). Although software radios use digital techniques, they are different from software-controlled digital radios in that the programmability of software radios includes programmable RF bands and the frequency conversion to and from the baseband beyond the IF stage, is performed by software (Mitola, 1995).

SDR allows interoperability and seamless connectivity between different radio communications standards and reduces the time to market of new radio products (Reed, 2002). Software radios employ simple hardware and provide additional flexibility with baseband signals. The physical layer behaviour of software radios can be extensively reconfigured through changes to its software without replacing the hardware (Nakajima, Kohno and Kubota, 2001). Software radios reduce the hardware size, weight and power consumption of radio units significantly (Mitola, 1995). Software radios contain embedded impedance synthesizers and are suitable for use in multiple-input multiple-output (MIMO) systems because the antenna systems are controlled by software.

2.3 The evolution of Digital Signal Processors

The field of digital signal processing became established in the 1970s after J. W. Turkey and J. W. Cooley rediscovered the Fast Fourier Transform (FFT) algorithm, which could be computed much faster than other Discrete Fourier Transform (DFT) algorithms in 1965

(Smith, 1997; Kline, 1990; Heideman, Johnson and Burrus, 1985). The FFT dates back to the work of J. C. F. Gauss in 1805. Literature suggests that the emergence of Very Large Scale Integration (VLSI) in the 1980s ushered in an era of smaller, faster and cheaper DSPs (Eyre and Bier, 2000; Frantz, 2000; Rabaey *et al.*, 1998; Lee, 1988). This led to the wide use of DSPs in wireless communication networks. Most of these DSPs employed the Harvard architecture, where the address spaces and buses for data and multiply-accumulate operands, were separated. Prior to that time, microprocessors were built using the Von Neumann architecture and could not perform real-time intensive computations at acceptable speeds. Most of the DSPs did not support floating point calculations. Once they were fabricated, the DSPs could not be reprogrammed and were therefore used for specific, stand-alone purposes in biomedical engineering, acoustics, voice communication, data communication, image processing, nuclear science, radar, sonar, astronomy and seismology (Kline, 1990).

Re-programmable DSPs and other features like direct memory access control, multiple-data instructions, variable-length instructions and support for interrupts were introduced in DSP architectures a few years later (Frantz, 2000; Rabaey *et al.*, 1998; Lee, 1988). Verbauwhede *et al.* (1996) summarized the evolution of DSPs in five generations shown in table 2.3.1.

Table 2.3.1 – The evolution of DSP technology (Source: Verbauwhede *et al.*, 1996)

Year	**Generation**	**Feature**	**Examples**
1982	1st	Basic Harvard Architecture, 1 data bus, 1 program bus, 1 Multiply- arithmetic-logic unit	TMS32010, NEC 7720
1986	2nd	"Modified" Harvard Architecture, 1 data/program bus, 1 data bus	TMS320C25, AT&T DSP16A
1990	3rd	Extra addressing modes, Extra functions	TMS320C5x, AT&T DSP161x
1994	4th	2 data busses, 1 program bus, separate Multiply-accumulate unit, ALU	TMS320C540
1995	5th	2 data busses, 1 program bus, 2 Multiply-accumulate units, 1 ALU	Lode

Lee's (1988) review of the features of the early DSPs developed by different manufacturers is summarised in table 2.3.2. Research to reduce the silicon area, implementation cost and power consumption of DSPs, while meeting the increased speed and quality requirements of various applications continued at architectural and algorithmic level. As a result, architectures such as the very-long instruction-word (VLIW) architecture and the Lode architecture were developed (Frantz, 2000; Rabaey *et al.*, 1998; Verbauwhede *et al.*, 1996).

Examples of the algorithms which were developed are parallel processing (Chandrakasan and Brodersen, 1995), synchronous pipelining (Verbauwhede *et al.*, 1996; Chandrakasan and Brodersen, 1995; Chandrakasan, Sheng and Brodersen, 1992), asynchronous pipelining (Meng, Brodersen and Messerschmitt, 1991), two-level pipelining of the systolic block householder transformation (Liu, Hsieh and Yao, 1992), bit level pipelining of the delayed least-mean-square algorithm (Wang, 1994), multirate computing (Liu, Wu, Raghupathy and Chen, 1998) and the Viterbi algorithm (Verbauwhede and Nicol, 2000; Forney, 1973; Viterbi, 1967).

Table 2.3.2 – Features of single chip DSPs from different manufacturers (Source: Lee, 1988)

Company	Part	Year of introduction and features
AMI	S2811	1978. First DSP described. 12/16 bit fixed point. 300 ns MAC time.
	S28211/2	1983. Update of 2811.
Analog Devices	ADSP-2100	1986. 125 ns MAC time, 16/40 bit fixed. Memory is off-chip.
	ADSP-2100A	1988. Update of 2100, 80 or 100 ns MAC time.
	ADSP-2101/2	1988. 2100A with internal RAM and peripherals. 2102 has mask-programmable program ROM.
AT&T	DSP1	1979. An early DSP, used internally. 800 ns MAC. 16 and 20 bit operands to the multiplier.
	DSP20	1981. Update of DSP1. 400 ns MAC time. Also not marketed.
	DSP32	1984. 160 ns MAC time. 16 bit fixed. 32/40 bit floating point.
	DSP16	1987. 55 ns MAC time. 16/36 bit fixed.
	DSP32C	1988. 80 ns MAC time. 16 or 24 bit fixed. 32/40 bit floating point.
	DSP16A	1988. 33 ns MAC time. 16/36 bit fixed.
Fujitsu	MB8764	1983. 100 ns MAC time. 16/26 bit fixed.
	MB87064	1985. Version of 8764 with fewer pins.
	MB86232	1987. 32b floating point with 150 ns MAC (75 ns fixed point). IEEE std.
	MB86220	1989? 24b floating point with 160 ns MAC (80ns fixed point). Similar to 8764.
Hitachi	HD61810	1982. 12/16 bit floating point. 250 ns MAC time.
	DSPi	1988. For image processing, very fast I/O. 50 nsec MAC time.
IBM	Hermes	1981. 100 ns MAC time accomplished with a clever technique called "ZIPing". Not marketed outside IBM.
Motorola	DSP56001	1987. 74.1 ns MAC time. 24/56 bit fixed.
	DSP56000	1987. Mask-programmed version of 56001.
	DSP96002	1989. 75 ns MAC time. 32/64 bit fixed. 44/96 bit floating point.
National	LM32900	1987. 100 ns MAC, 16/32 bit fixed, no internal memory.
NEC	μPD7720	1980. 250 ns MAC. Heavily used, early DSP.
	μPD7720A	Update of 7720. 244 ns MAC.
	μPD7281	1984. Data flow machine for image processing. A very unusual device.
	μPD77230	1985.32b floating point, 150 ns MAC.
	μPD77220	1986. 24/48b fixed, 100 ns MAC, subset of 77230.
	μPD77C25	1988. Update of 7720A. 122 ns MAC.
Oki	6992	1986. 100 ns MAC time, 22 bit floating point.
Philips/ Signetics	5010	125 ns MAC time, 16 bit words.
	5011	5010 with no internal program ROM.
Thomson/ Mostek	68931	1986. Noted for complex data. 360 ns complex MAC time, 16/32 bit fixed.
	68930	1987. Version of 68931 with fewer pins.
Toshiba	6386/7	1983. 16/31 bit fixed. 250 ns MAC.
Texas Instr.	TMS32010	1982. 390 ns MAC time. 16/32 bit fixed.
	TMS32011	1985. Version of TMS32010 with μ-law and A-law conversion.
	TMS32020	1985. 195 ns MAC time. 16/32 bit fixed.
	TMS320C10	1986. CMOS version of the TMS32010.
	TMS320C15	1987. CMOS version of the 32010 with larger memories.
	TMS320E15	1987. Version of the 320C15 with EPROM.
	TMS320C17	1987. CMOS version of the 32011 with larger memories.
	TMS320E17	1987. Version of the 320C17 with EPROM.
	TMS320C25	1987. 100 ns MAC time. 16/32 bit fixed.
	TMS320C30	1988. 60 ns MAC time. 24/32 bit fixed. 32/40 bit floating point.
Zoran	34161	1986. "Vector signal processor", 100 ns MAC time, 16 bit block floating point. High level DSP
	34322	instructions (e.g. DFT).
	35325	1988. 32 bit block floating point.
		1989? 32 bit IEEE floating point.

The architectures of DSPs adopted at different times were influenced by the DSP algorithms known at the time. The application requirements and the complexity of the computed algorithms governed the DSP clock rate. Consequently, the power dissipation per Million Instructions per Second (MIPS) decreased as shown in figure 2.3.1 and the DSP market growth rate exceeded the yearly growth rate of the overall semiconductor market (Frantz, 2000; Verbauwhede and Nicol, 2000; Rabaey *et al.*, 1998).

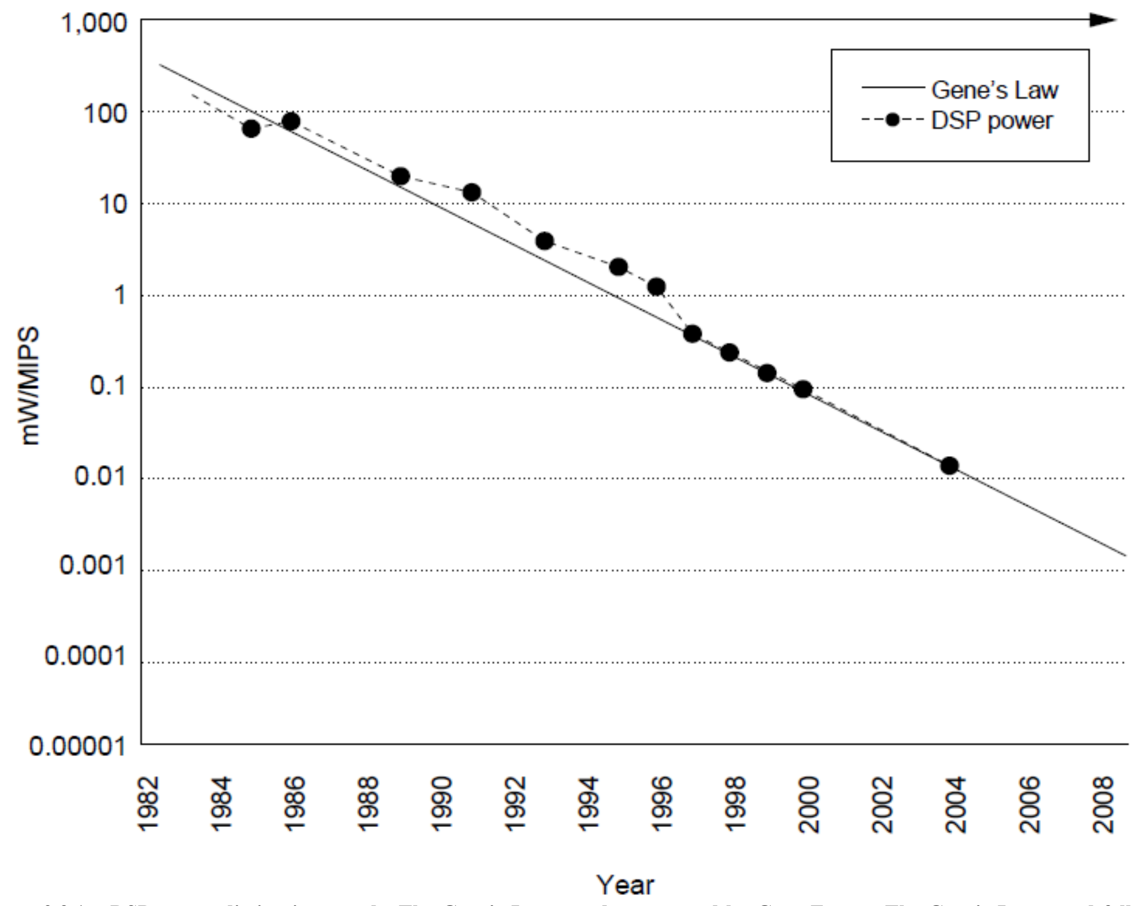


Figure 2.3.1 – DSP power dissipation trends. The Gene's Law trend was named by Gene Frantz. The Gene's Law trend followed that of Moore's Law in that the power dissipation per MIPS was halved every 18 months for the period shown (Source: Frantz, 2000).

The interface to DSP memory of the first high level compilation systems which were used to programme DSPs limited the bandwidth for some applications and it was difficult to get the exact mapping desired (Tessier and Burleson, 2001; Frantz, 2000). The need for reconfigurable radio receivers and transmitters that could be updated remotely and the prevalent interoperability challenges in military applications prompted further research. The expansion of mobile wireless communication also fuelled the demand for low power programmable DSPs. As the cost per MIPS of DSPs dropped, it became possible for more DSP functions to be implemented in software. This led to the concept of software radio (Mitola, 1995).

The term software radio was created in the early 1990s by J. Mitola III and was later used widely in the wireless systems community (Jeyalakshmi and Sankaranarayanan, 2010; Dickens, Dunn and Laneman, 2008; Dillinger, Madani and Alonistioti, 2003; Reed, 2002). The first software radio, named SPEAKeasy, was developed between 1992 and 1995 for use in military applications (Lackey and Upmal, 1995). Software radio took considerable time to develop into a mature wireless communication technique because its potential commercial applications were cost sensitive (Reed, 2002).

2.4 SDR Technological Advancements

The idea that the modifications made to signals, in order to improve their transmission and reception could be implemented in software became apparent after reprogrammable digital signal processors were developed. Thus hardware-oriented radios have been gradually transformed into software-defined radios over the past two decades. SDR technology has been pursued because of its potential benefits. Both commercial and non-commercial organizations have invested in research and development to make SDR a reality.

In an ideal software radio, all of the radio functionality is entirely defined in software. The analog to digital converter (ADC) is connected to the antenna through a low noise amplifier and impedance matching circuitry for correct matching to the antenna. The digital to analog converter (DAC) is connected to the antenna through a power amplifier (Arslan, 2007; Oh *et al.*, 2005; Cetiner *et al.*, 2004). The DSP receives digital data from the ADC and transforms it, in software, to the form required by the application before sending the resulting processed data stream to the DAC. A typical ideal SDR architecture is shown in figure 2.4.1.

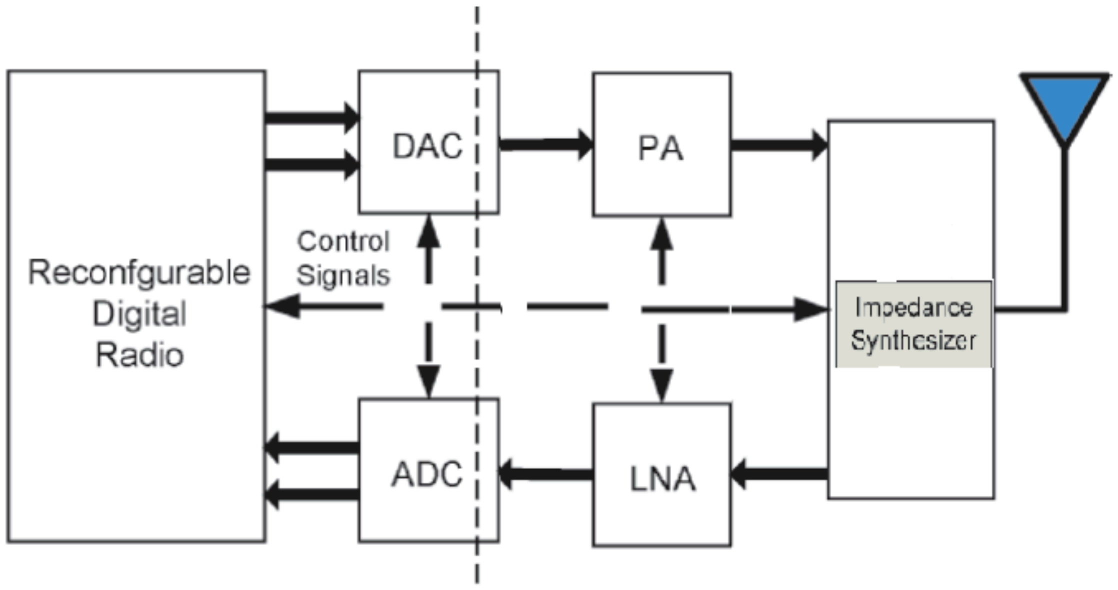


Figure 2.4.1 - Ideal SDR architecture

Earlier research has shown that the ideal software radio is not yet fully realizable for applications operating above the Very High Frequency (VHF) band due to cost considerations and several technological limitations. ADCs which can provide the required sampling rates and bit-error-rates consume too much power, increase in size and become too expensive beyond the VHF band. While the bandwidth is enough for most existing applications, the carrier frequency is usually higher than VHF. Noise from external sources, spurious signals and self-generated phase-noise also interfere with the sampling. The processing time also increases (Farrell, Sanchez and Corley, 2009; Dickens, Dunn and Laneman, 2008; Arslan, 2007). Trade-offs between system flexibility and system performance are made by employing RF frontends for signal down-conversion and up-conversion, amplification and filtering (Hedde *et al.*, 2009; Hueber *et al.*, 2009; Dickens, Dunn and Laneman, 2008; Lu and Lu, 2008; Ramacher, 2007; Xu, Bosisio and Wu, 2006). The typical practical design of a software radio is shown in figure 2.4.2.

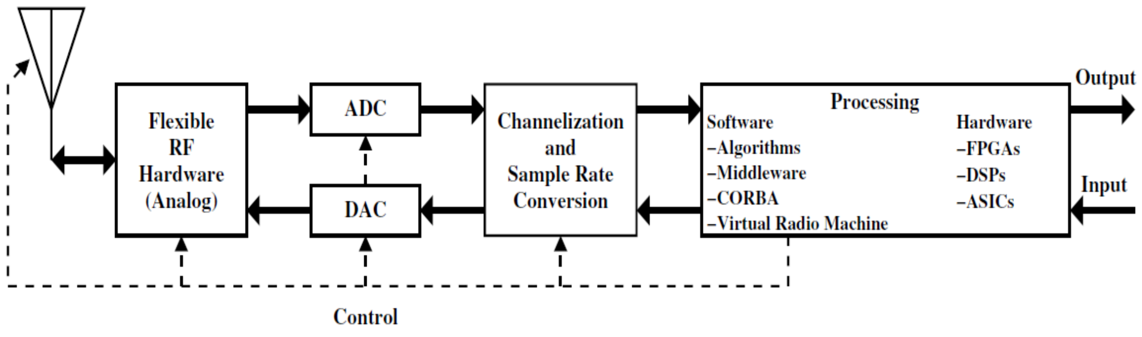


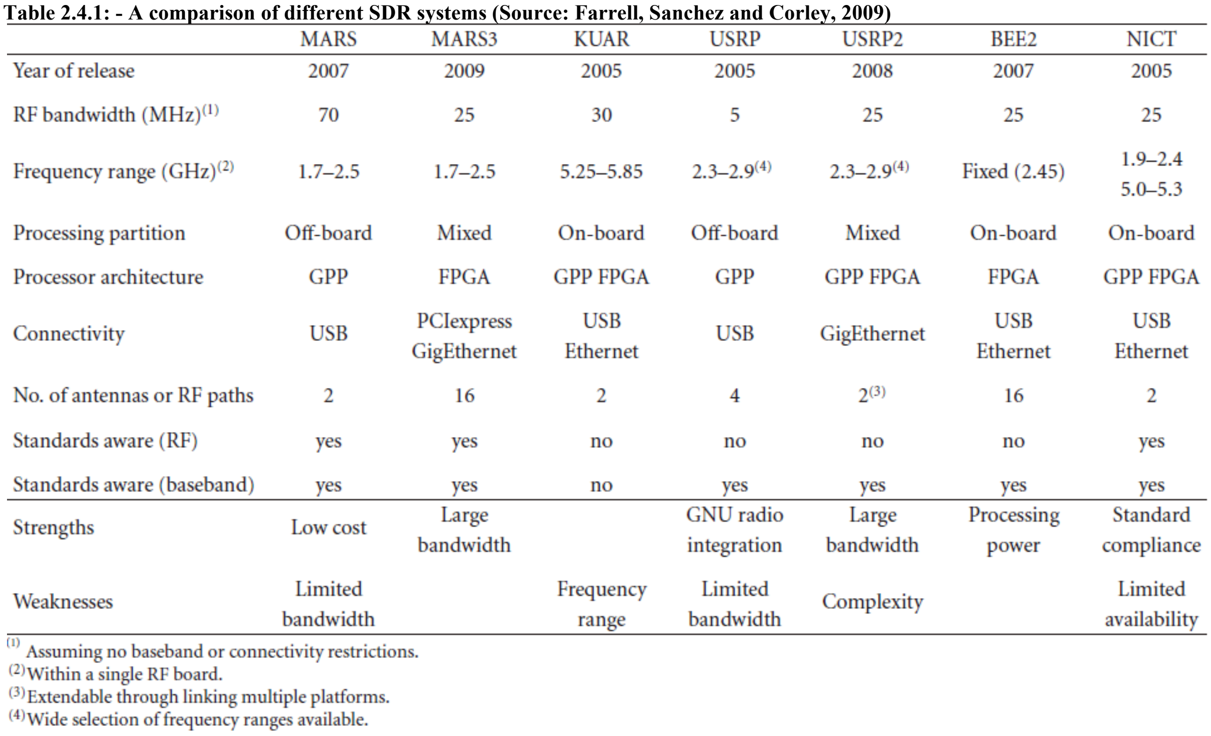
Figure 2.4.2 – Practical Software defined radio architecture (Source: Reed, 2002)

The flexible RF hardware shown in the figure comprises of analogue functions which include filtering at the radio frequency (RF) stage, reference frequency generation, receiver low noise amplification and transmitter power amplification (Reed, 2002).

Some software radios have been developed from open architectures such as the GNU Radio (Dickens, Dunn and Laneman, 2008), while some have remained closed and proprietary (Minden *et al.*, 2007). Mobile cellular base stations that match the performance of hardware defined base stations have been built on SDR architecture (Farrell, Sanchez and Corley, 2009; Burns, 2003). To date, a number of software radio systems have been implemented using different semiconductor processors like FPGAs, DSPs or GPPs (Farrell, Sanchez and Corley, 2009; Dickens, Dunn and Laneman, 2008; Bose, Ismert, Welborn and Guttag, 1999).

Table 2.4.1 shows a comparison of several SDR systems that have been developed in the recent years.

Table 2.4.1: - A comparison of different SDR systems (Source: Farrell, Sanchez and Corley, 2009)



	MARS	MARS3	KUAR	USRP	USRP2	BEE2	NICT
Year of release	2007	2009	2005	2005	2008	2007	2005
RF bandwidth (MHz)[1]	70	25	30	5	25	25	25
Frequency range (GHz)[2]	1.7–2.5	1.7–2.5	5.25–5.85	2.3–2.9[4]	2.3–2.9[4]	Fixed (2.45)	1.9–2.4 5.0–5.3
Processing partition	Off-board	Mixed	On-board	Off-board	Mixed	On-board	On-board
Processor architecture	GPP	FPGA	GPP FPGA	GPP	GPP FPGA	FPGA	GPP FPGA
Connectivity	USB	PCIexpress GigEthernet	USB Ethernet	USB	GigEthernet	USB Ethernet	USB Ethernet
No. of antennas or RF paths	2	16	2	4	2[3]	16	2
Standards aware (RF)	yes	yes	no	no	no	no	yes
Standards aware (baseband)	yes	yes	no	yes	yes	yes	yes
Strengths	Low cost	Large bandwidth		GNU radio integration	Large bandwidth	Processing power	Standard compliance
Weaknesses	Limited bandwidth		Frequency range	Limited bandwidth	Complexity		Limited availability

(1) Assuming no baseband or connectivity restrictions.
(2) Within a single RF board.
(3) Extendable through linking multiple platforms.
(4) Wide selection of frequency ranges available.

Research is currently directed towards developing SDR platforms with wider RF bandwidth, larger operating frequency range and less complexity.

2.5 DSP Education

Over time, digital signal processing education has followed the developments of the theories and commercial implementations of digital signal processors. DSP courses were initially taught at graduate level then later included in undergraduate curricula (Jones, 2001; Morrow, Welch, Wright and York, 2001; Fuchiwaki, Usuki, Arai and Murahara, 2000; Smith, 1997).

Three main DSP teaching methods are used. These are, interactive web-based teaching (Yasin, Karam and Spanias, 2003; Youngwook, Duman and Spanias, 2003; Jackson *et al.*, 2001; Spanias and Bizuneh, 2001; Spanias *et al.*, 2000; Clausen and Spanias, 1998; Mobasseri, 1997; McClellan, Schafer, Schodorf and Yoder, 1996), hands-on experiments using industry standard evaluation kits and equipment (Kang *et al.*, 2007; Gadhiok *et al.*, 2005; Wright, Welch and Morrow, 2003; Mousavinezhad and Abdel-Qader, 2001; Gan, Chong, Gong and Tan, 2000; Dickerson, 1998; Chiang *et al.*, 1996; Kuo and Miller, 1995; Taylor, Mellott and Lewis, 1996; Aboulnasr, 1995; Kurugöllü *et al.*, 1995; Madisetti, McClellan and Barnwell, 1995; Oxtoby, 1995; Trussell and Rajala, 1995; Barnwell, Madisetti and McGrath, 1993; Chassaing, Anakwa and Richardson, 1993) and simulation-based experiments (Chen and Han, 2008; Spanias *et al.*, 2000; Joaquim, Pereira and de Oliveira, 1998).

Interactive web-based laboratories allow teaching of large numbers of students who are geographically dispersed, like distant learners. Students can access the exercises from remote locations at times convenient to them. Online feedback, testing, and exercises are provided by web-based laboratories. The sessions are usually timed, and will time out after a preset period of time has elapsed.

Hands-on laboratory experiments allow students to benefit from the tangible experience of applying the complex, mathematical theories of DSP as they generate and receive physical signals to and from the real world. Hands-on laboratories enhance the students' information retention and improve their design skills.

In simulation-based laboratories, software is used to replicate real-world applications at much lower costs. Simulations help to develop students' conceptual understanding of DSP. However, simulations do not replace that real-world experience provided by hands-on laboratories. Normally, universities with high student enrollment use this teaching method. Simulation-based experiments are different from web-based in that they cannot be accessed remotely over the internet.

Different learning institutions have focused on different aspects of DSP. Some institutions have focused on teaching the fundamental principles including sampling, quantization, filtering, convolution, transforms and modulation (Baez-Lopez, Trueba-Marcos, Ramirez and Jimenez-Lopez, 2001; DeBrunner, DeBrunner, Radhakrishnan and Khan, 1996; Trussell and Rajala, 1995). Other institutions have focused on the applications of DSP like real-time beam forming (Morrow, Welch, Wright and York, 2001), biomedical DSP (Melton, Finelli and Rust, 1999; Stewart, 1993), acoustic echo cancellation, image compression and active noise control (Kuo and Miller, 1995), image processing, speech processing and array signal processing (Allebach, Zoltowski and Bouman, 1994).

Recently, SDR platforms have been used for research and testing of different applications (Gonzalez *et al.*, 2009; Katz and Flynn, 2009; Nagurney, 2009; Dickens, Dunn and Laneman, 2008; Minden *et al.*, 2007; Hwang, 2003). Examples are Gonzalez et al., (2009), who presented an open source software development platform designed for modelling SDRs and teaching long term evolution wireless communications systems. Minden *et al.* (2007) also developed a software-defined radio development platform named the Kansas University Agile Radio (KUAR) to carry out advanced research in the fields of wireless radio networks, dynamic spectrum access, and cognitive radios.

2.6 Conclusion

Today faster versions of programmable DSPs based on leading-edge algorithms and capable of handling super high frequency signals are continually designed to meet the requirements of various software radio applications. Significant improvements in signal processing performance have been made in applications like multichannel base stations and low power wireless phones. However, optimal RF frontend solutions to support the flexibility brought about by software radios in multi-mode implementations remain elusive and further research is required in that area. Further research could also be extended to identifying feasible solutions for the transition from isolated software radio applications to large deployments of cognitive radio networks.

Chapter three

3 Project Specification

3.1 The Hardware

Each laboratory workstation comprises of a host computer and the Ettus Research Universal Software Radio Peripheral (USRP2). The host computer is a Pentium IV machine with 3 GB RAM, 80 GB HDD memory and a 2 GHz 32-bit processor. The host computer also has a Dell 4850E sound card and a set of loud speakers for signal output. Figure 3.1.1 illustrates the hardware setup.

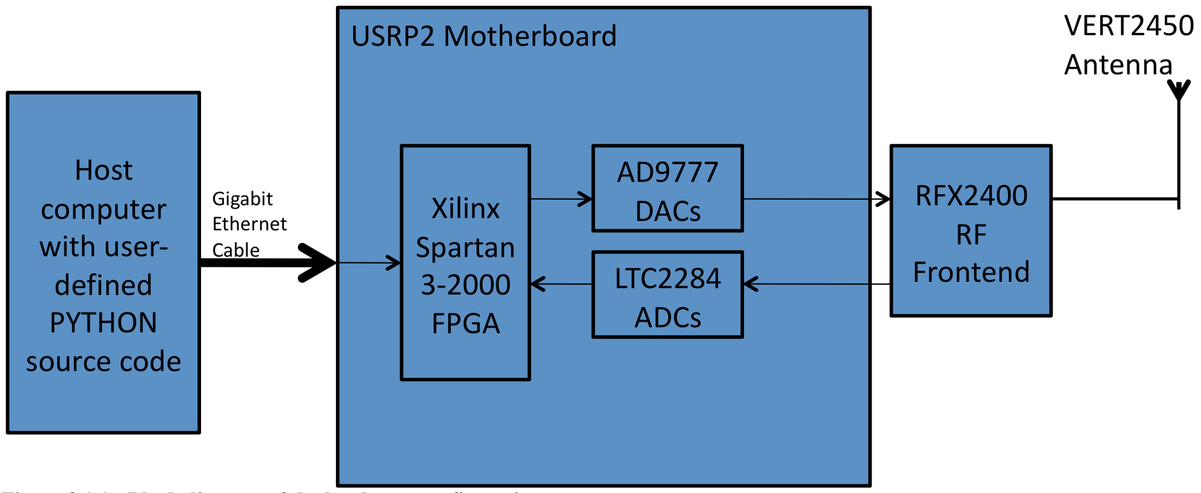


Figure 3.1.1 - Block diagram of the hardware configuration

The USRP2 encompasses a Xilinx Spartan 3-2000 FPGA, two AD9777 chips that provide the digital-to-analogue conversion capability and two LTC2284 analogue-to-digital converters. There is an auxiliary ADC, the AD7922, and an auxiliary DAC, the AD5623. A 2.3-2.9 GHz transceiver, the RFX2400, is mounted on the USRP2 motherboard as an RF frontend. The RFX2400 transmits and receives via a VERT2450 tri-band antenna. The USRP2 system interfaces directly with the host computer via a 1 Gbps Broadcom BCM5751 PCIe Ethernet cable. The semiconductor chip, DP83865, on the USRP2 is used to manage the Gigabit Ethernet interface. The USRP2 is clocked internally by a 100 MHz clock and is equipped with a 2 GB SD memory card which stores the FPGA configuration and firmware. Table 3.1.1 summarises the specifications of the USRP2.

Table 3.1.1:- USRP2 specifications (Source: Ettus Research LLC, 2010)

Feature	Value	Quantity
FPGA	Xilinx Spartan 3-2000	1
Interface to host computer	Gigabit Ethernet	1
RF Bandwidth to/from host computer	50 MHz @ 16bits	-
Analogue-to-Digital Converter	14-bit, 100 Million Samples per second	2
Digital-to-Analogue Converter	16-bit, 400 Million Samples per second	2
Frequency Range	DC to 5.9 GHz	-
Transmit - receive capacity	1 Transmitter and 1 Receiver or 1 Transceiver	1
Static Random Access Memory (SRAM)	1 Megabyte	1
Power supply requirements	6Volts, 3Amperes	-
Reference clock stability	20 ppm	-
External clock reference	10 MHz, less than $3V_{peak\text{-}to\text{-}peak}$	1
Multi-channel systems (MIMO)	Up to 8 antennas	-

The FPGA is the central control of the USRP2 system. It configures the operation of all its peripherals, monitors their status and handles the control channel for the Gigabit Ethernet. The FPGA responds to data transfer requests passed between the host computer and the RF frontend. In the FPGA, the signal received from the ADCs is down-converted from the IF band to the base band. The FPGA also decimates the signal to a data rate that is compatible with the Gigabit Ethernet and the host computers' computing capability. The FPGA performs the pre-processing and interpolation of signals before they are transmitted via the DACs. The FPGA also handles clock domain crossing and alignment.

In the receive path of the USRP2, the RFX2400 frontend converts the received signal from RF to IF. The RFX2400 has a bandpass filter with an ISM passband of 2.4 - 2.483 GHz which filters out and attenuates any interfering RF signals outside that ISM band. The LTC2284 ADC then converts the received analogue signal into digital samples. Since the maximum sample rate of the LTC2284 is 100 Million Samples per second (MSps), the maximum bandwidth that can be digitized without aliasing is 100 MHz/2=50 MHz, according to the Sampling theorem for band-limited signals (Proakis, 1989). If a signal with a bandwidth larger than 50 MHz is sampled, aliasing is introduced and the band of the wanted signal is translated to other parts of the frequency range -50 MHz to 50 MHz. Digital down-conversion on the FPGA is accomplished with 4 stages of cascaded integrator-comb (CIC) filters and two half-band filters, which use adders and delays only, for spectral shaping and out of band signal rejection. The high rate half-band filter has 7 taps and the low rate half-band filter has 31 taps. For a decimation value of *D* and an ADC sample rate of *F*, the output sample rate, *N*, of the received waveform is given by:

[illegible]=[illegible] ×[illegible] (3.1.1)

(Smith, 1997). The frequency band [-*F/D*, *F/D*] is selected and the signals' spectrum is de-spread from [-*F*, *F*] to [-*N*, *N*]. So the bandwidth of the wanted signal is narrowed by a factor of *D*. The minimum decimation value of the USRP2 is 4 while the maximum decimation value is 512. Narrow frequency ranges are achieved by increasing the decimation value. The maximum IF bandwidth is therefore:

[illegible]=[illegible] (3.1.2)

=100 [illegible]4

=[illegible]

Data samples are sent over the Gigabit Ethernet as 16-bit signed integers in IQ format. So each complex sample is 4 bytes, that is 16-bit I and 16-bit Q data. The maximum IF Ethernet data rate is therefore:

[illegible]= 4 [illegible] ×(25 [illegible])

=100 [illegible] ×8 [illegible]

=[illegible]

The input voltage range of the LTC2284 ADC is 2V$_{peak\text{-}to\text{-}peak}$, and the output load impedance is 50 Ω (Linear Technology Corporation, 2006). The maximum power is therefore:

$$Maximum\ Power = \frac{\left(\frac{V_{pp}}{2\sqrt{2}}\right)^2}{R} = \frac{\left(\frac{2}{2\sqrt{2}}\right)^2}{50\Omega} = 10\ mW \qquad (3.1.3)$$

$$Maximum\ Power = 10\log_{10}\left(\frac{10\ mW}{1\ mW}\right) = 10\ dBm$$

The LTC2284 is a 14-bit ADC. This yields a Spurious Free Dynamic Range (SFDR) of:

$$SFDR = -20\log\frac{1}{2^{14}} = 84.3\ dBc$$

In the transmit path of the USRP2, baseband IQ complex data is sent from the host computer to the FPGA where it is interpolated and up-converted to the IF band. The maximum sample rate between the FPGA and the DAC is 100 MSps. The interpolation from 100 MSps to 400 MSps takes place inside the AD9777 DAC chip. This is a way of simplifying analogue reconstruction filtering. Direct frequencies of up to 170 MHz can be transmitted by the USRP2. Digital up-conversion is also accomplished with a 4-stage CIC and two half-band filters. The minimum interpolation value is 4 and the maximum interpolation value is 512. The DAC sample rate, interpolation value and the output sample rate of the transmitted signal are related by the formula:

$$DAC\ Sample\ Rate = Interpolation\ Value \times Output\ Sample\ Rate \qquad (3.1.4)$$

Odd values of decimation or interpolation enable the CICs only with no filter half-bands. Even values of decimation or interpolation that are not a multiple of 4 yield one low rate half-band. Even values of decimation or interpolation that are a multiple of 4 enable use of both half-bands. The AD9777 DAC can supply 2V$_{peak\text{-}to\text{-}peak}$ to a 100 ohm differential load (Analog Devices, 2006). From equation 3.1.3, this is equivalent to an output power of 20mW (13 dBm). A programmable gain amplifier (PGA) after the DAC provides up to 20dB gain. The SFDR of the AD9777 is:

$$SFDR = -20\log\frac{1}{2^{16}} = 96.3\ dBc$$

The USRP2 motherboard can accommodate either a single transceiver or 1 transmit module and 1 receive module. The USRP2 can operate in full duplex mode as long as the combined data rate for the transmitter and receiver sides over the Gigabit Ethernet does not exceed 800

Mbps. Table 3.1.2 summarises the specifications of the RFX2400 transceiver and Table 3.1.3 summarises the specifications of the VERT2450 antenna. The RFX2400 has an RF synthesizer which enables a split-frequency operation for the transmitter side and receiver side. It has a narrow bandwidth and the maximum SFDR of 95 dBc reduces noise and interference.

Table 3.1.2:- RFX2400 specifications (Source: Ettus Research LLC, 2010a)

Feature	**Value**
Power supply	6Volts, 1.2 Amperes
Frequency Range	2.3 to 2.9 GHz
Maximum Transmit Power	17dBm (Adjustable)
ISM Bandpass filter range	2.4 to 2.483 GHz
PLL lock time	less than 200 microseconds
Digital Input / Output lines	16
Automatic gain control range	70 dB

Table 3.1.3:- VERT2450 specifications (Source: L-Com Global Connectivity, 2010)

Feature	**Value**
Frequency Range	2.3 to 2.5 GHz, 4.9 to 5.35 GHz, 5.725 to 5.85 GHz
Maximum Transmit Power	17dBm
Gain	3 dBi
Impedance	50 Ω
Voltage Standing Wave Ratio	Less than 2:1

Figure 3.1.2 is a picture showing the USRP2 motherboard construction and the RF2400 transceiver board.

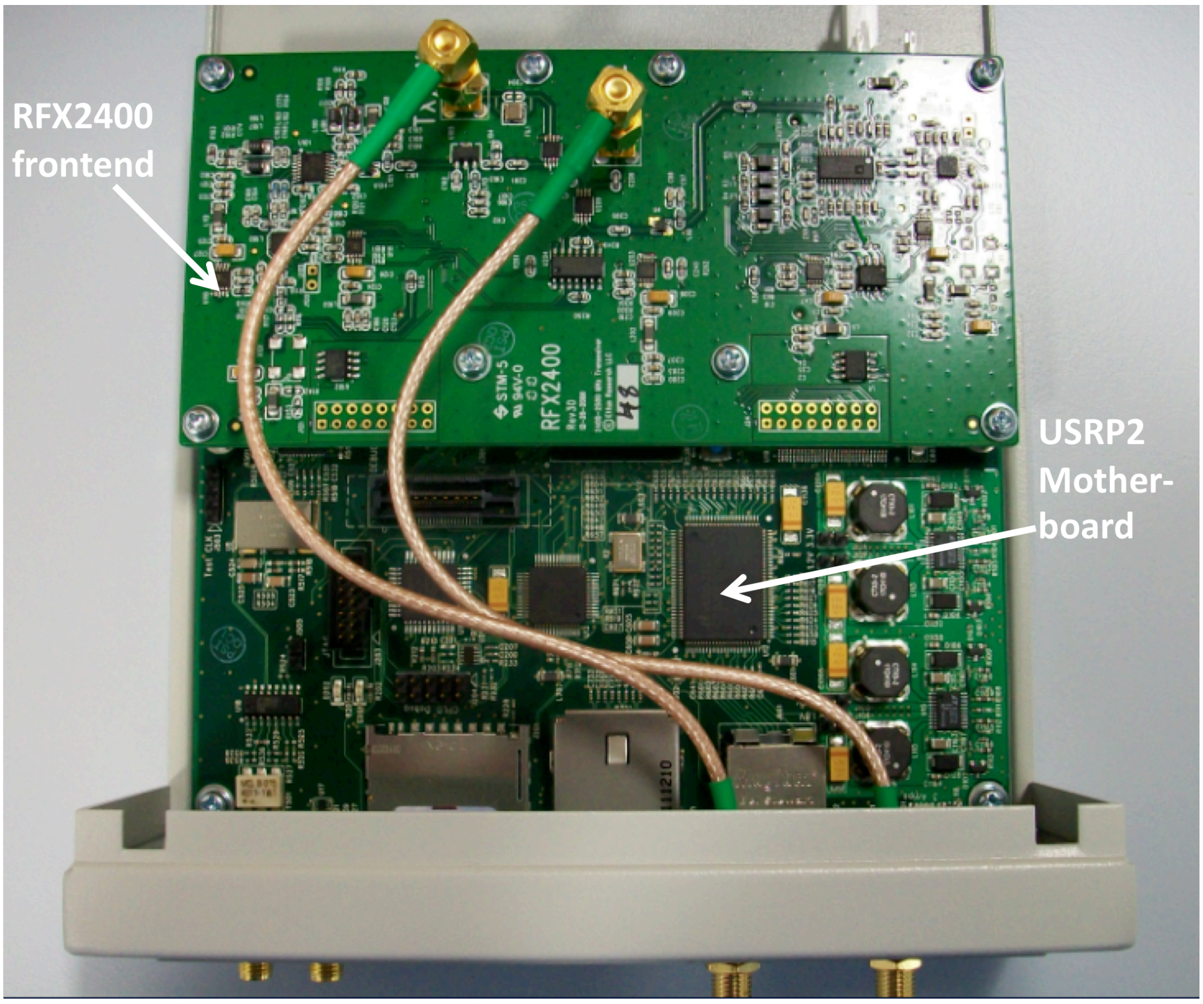


Figure 3.1.2: The USRP2 motherboard and the RF2400 transceiver

3.2 The GNU Radio Software

The USRP2 hardware was used with the GNU Radio software (version 3.3.1) to provide the development environment for software radio design. The GNU Radio is an open source software development framework which runs on the Linux Mandriva 2010.0-i586 operating system. Digital Signal processing functions and all the critical low-level algorithms are written as blocks of C++ implementations. These C++ blocks are linked together and callable from PYTHON scripts. The PYTHON scripts control the data flow among blocks. GNU Radio has a GUI called GNU Radio Companion (GRC) which can be used to generate PYTHON flow graphs. GRC provides easy reconfiguration and control of signal flow. Signal flow is controlled by a GNU Radio scheduler using PYTHON's built-in module threading and not the Linux OS. GNU Radio supports floating-point specific instructions.

Chapter four

4 Project Execution

The Linux operating system (OS), Mandriva 2010.0-i586, was installed on two host computers. Following the OS installation, the dependency software packages required for the operation of GNU Radio v3.3.1 were installed using the Mandriva package installer, RPMdrake. The list of these dependencies is provided in Appendix B. One of the dependencies, Microblaze v4.1.1, was installed from source through the Linux command line interface. The GNU Radio software was then installed from source on each PC. The installation procedure for Mandriva and GNU Radio is detailed in Appendix A. Figure 4.1 shows the setup of the equipment that was used for experimentation.

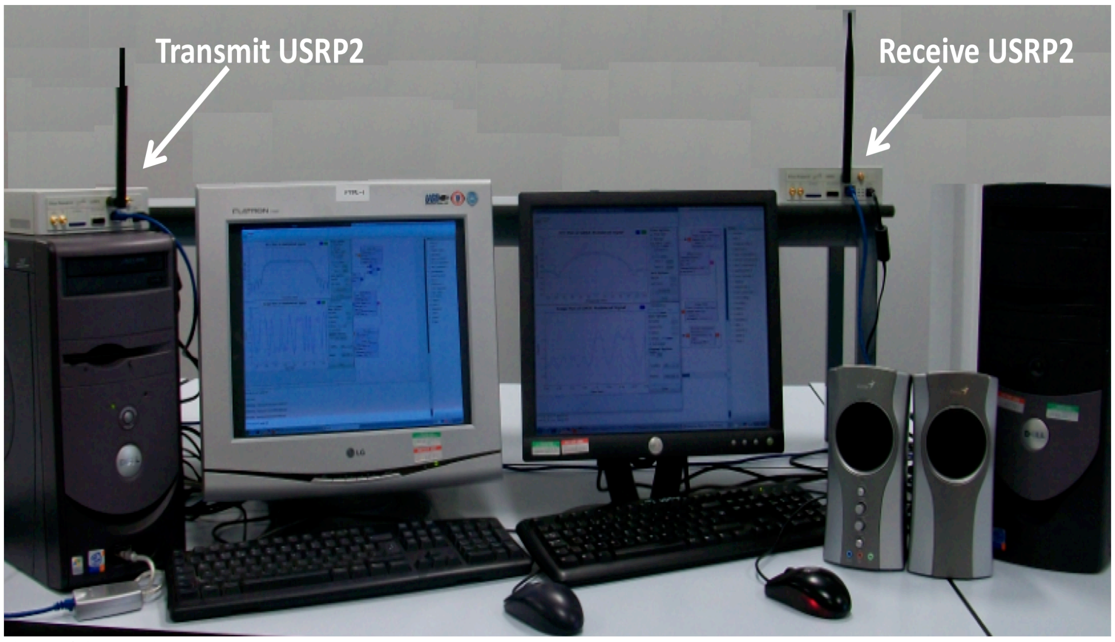


Figure 4.1 – Equipment layout

Twelve simulations were created and tested before implementation on the USRP2 as described hereafter. These simulations are:

- Signal Sampling
- Amplitude Modulation
- FIR and IIR filtering
- Binary Amplitude Shift Keying
- 4-Level Amplitude Shift Keying

- Non-coherent Binary Frequency Shift Keying
- Continuous Phase Frequency Shift Keying
- Binary Phase Shift Keying
- Differential Phase Shift Keying
- Gaussian Minimum Shift Keying
- Direct Sequence Spread Spectrum

PYTHON scripts of each simulation were used to generate and save an FPGA file with the file extension '.rbf' in the directory /usr/local/share/usrp. The '.rbf' file was loaded onto the FPGA over the Gigabit Ethernet at runtime.

4.1 Sampling Theorem Simulation

The sampling theorem states that a bandlimited analog signal which has no frequency components higher than *B* Hz can be completely reconstructed from a sequence of its discrete samples if the sampling rate exceeds 2B samples per second (Haykin, 2001; Proakis, 1989). In GNU Radio, the sampling rate of the software radio signal flow graph is defined by the parameters of the signal blocks used in the flow graph. The sampling theorem was illustrated by a 3.2 kHz continuous sinusoidal signal initially sampled at a rate greater than twice its frequency and then later sampled at a rate less than twice its frequency. The test signal was multiplied by a signal with a higher frequency of 32 kHz and the frequency domains of the two scenarios were compared.

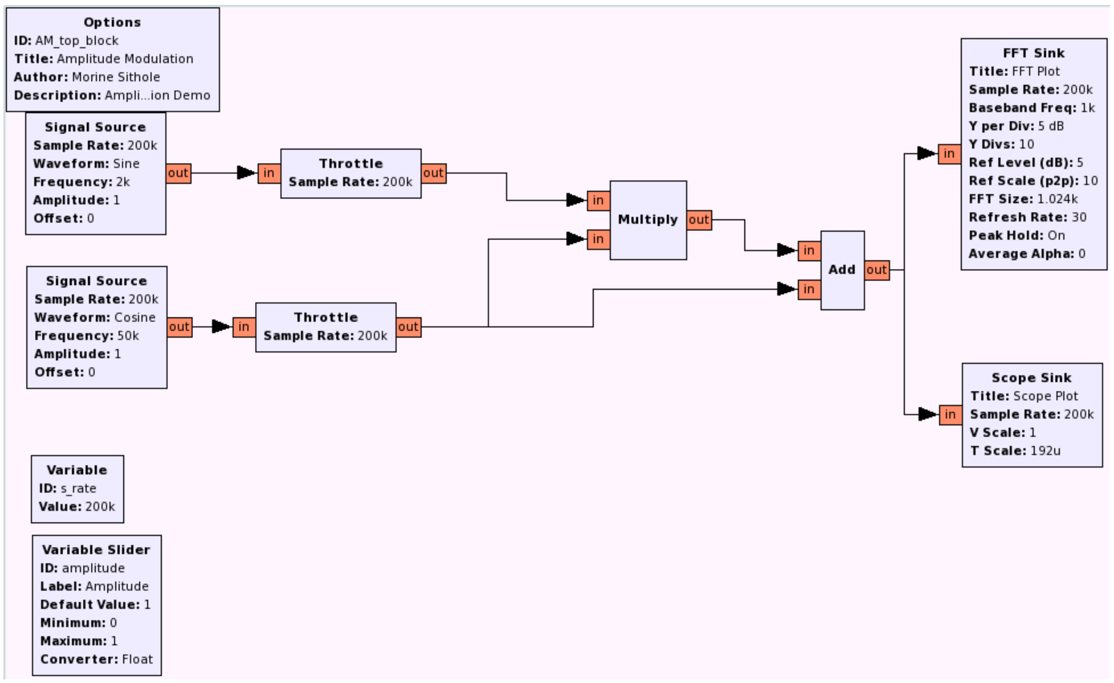
4.2 Amplitude Modulation

This simulation illustrated the basic equation of an amplitude modulated signal.

$$s(t)=A_c[1+h\,m(t)]\cos(2\pi f_c t) \qquad (4.2.1)$$

$$s(t)=A_c\cos 2\pi f_c t+A_c h\,m(t)\cos 2\pi f_c t$$

where *s(t)* is the modulated signal, A_c is the carrier amplitude, f_c is the carrier frequency, *h* is the modulation index and *m(t)* is the modulating signal (Haykin, 2001). The frequency of the modulating signal was 2 kHz and the frequency of the carrier signal was 50 kHz. Figure 4.2.1 shows the implementation of amplitude modulation in GRC.


Options
ID: AM_top_block
Title: Amplitude Modulation
Author: Morine Sithole
Description: Ampli...ion Demo

Signal Source
Sample Rate: 200k
Waveform: Sine
Frequency: 2k
Amplitude: 1
Offset: 0

Throttle
Sample Rate: 200k

Signal Source
Sample Rate: 200k
Waveform: Cosine
Frequency: 50k
Amplitude: 1
Offset: 0

Throttle
Sample Rate: 200k

Multiply

Add

FFT Sink
Title: FFT Plot
Sample Rate: 200k
Baseband Freq: 1k
Y per Div: 5 dB
Y Divs: 10
Ref Level (dB): 5
Ref Scale (p2p): 10
FFT Size: 1.024k
Refresh Rate: 30
Peak Hold: On
Average Alpha: 0

Scope Sink
Title: Scope Plot
Sample Rate: 200k
V Scale: 1
T Scale: 192u

Variable
ID: s_rate
Value: 200k

Variable Slider
ID: amplitude
Label: Amplitude
Default Value: 1
Minimum: 0
Maximum: 1
Converter: Float

Figure 4.2.1- Amplitude modulation

4.3 FIR and IIR Filtering

A square wave of 1 kHz was passed through a 3-tap finite impulse response (FIR) filter and a 3-tap infinite impulse (IIR) filter in parallel. The time and frequency domain of the outputs were compared.

4.4 Amplitude Shift Keying

4.4.1 Binary Amplitude Shift Keying (BASK)

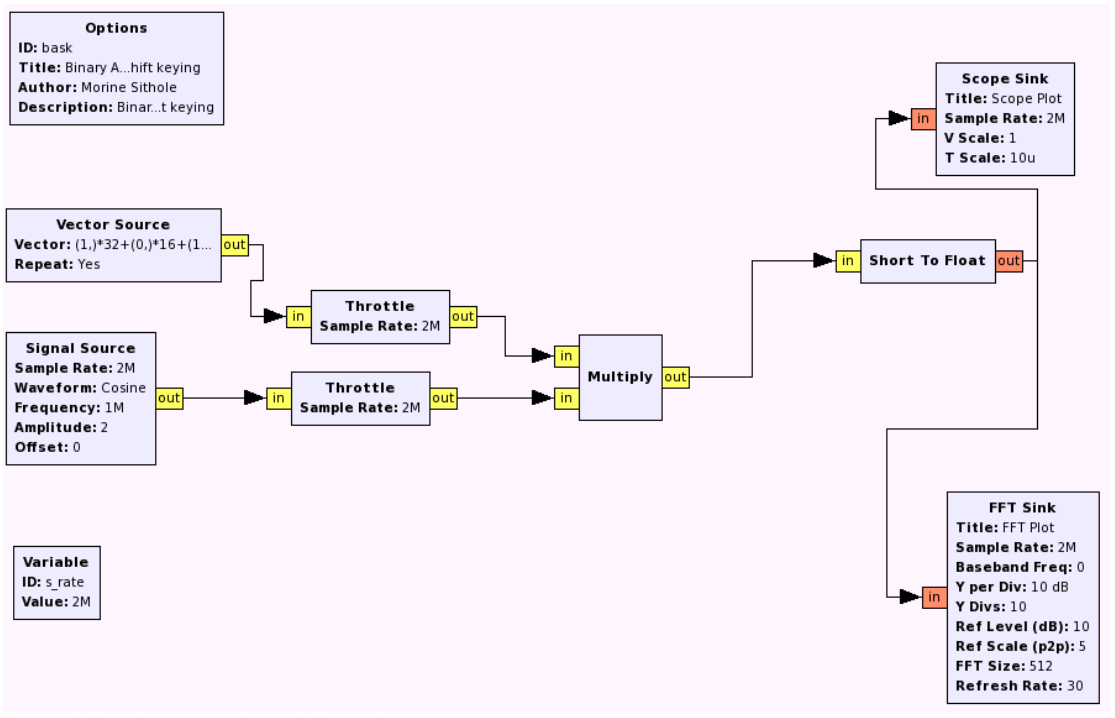
BASK was achieved by implementing the expression:

□□= □□□□cos2πp□□□, 0<□ <□ (4.4.1.1)

A_c is the carrier amplitude, $m(t)$ is the binary information signal, f_c is the carrier frequency, and T is the information bit duration (Peebles, 1987). $m(t)$ is either binary 1 or binary 0. The Fourier transform, $S(f)$, of the BASK signal $s(t)$ is expressed as:

□□=□□2□□−□□+□□2□□+□□ (4.4.1.2)

$S(f)$ shows that the spectrum of the information signal is shifted to f_c. The GNU Radio ‘vector source’ block was used to represent the information signal of binary 1s and 0s, denoted as a repetitive sequence. Figure 4.4.1 shows the implementation of BASK in GRC.


Options
ID: bask
Title: Binary A...hift keying
Author: Morine Sithole
Description: Binar...t keying

Vector Source
Vector: (1,)*32+(0,)*16+(1...
Repeat: Yes

Signal Source
Sample Rate: 2M
Waveform: Cosine
Frequency: 1M
Amplitude: 2
Offset: 0

Variable
ID: s_rate
Value: 2M

Throttle
Sample Rate: 2M

Throttle
Sample Rate: 2M

Multiply

Short To Float

Scope Sink
Title: Scope Plot
Sample Rate: 2M
V Scale: 1
T Scale: 10u

FFT Sink
Title: FFT Plot
Sample Rate: 2M
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 10
Ref Scale (p2p): 5
FFT Size: 512
Refresh Rate: 30

Figure 4.4.1- Binary Amplitude Shift Keying

4.4.2 Four-Level Amplitude Shift Keying

In GNU Radio, 4-Level ASK was achieved by defining the equation:

□□=□□□□□□2□ □□□ 0<□<□ 0 □□□□□ ℎ□□□

(4.4.2.1)

(Peebles, 1987), where the level of each symbol 00, 01, 10 and 11 was represented by A_i = -3, -1, 1, 3 in that order. The carrier frequency, f_c, was set to 2 MHz and the symbol duration, T, was 8 μs.

4.5 Non-coherent Binary Frequency Shift Keying

Non-coherent BFSK was implemented through the use of two GNU radio ‘vector source’ blocks which represented the binary data signal. Data from the first vector source was multiplied by a sinusoidal signal whose frequency represented the mark frequency. Data from the second vector source was multiplied by a second sinusoidal signal whose frequency represented the space frequency. The two signals resulting from the multiplications were added together to give the frequency shift keyed signal. The non-coherent BFSK signal can be represented by the equations:

□1□=□□□□□□2□ □1□ 0<□< □, □□□ □□□ 1 0
□□□□□ h□□□ (4.5.1)

□2□=□□□□□□2□ □2□ 0<□<□, □□□ □□□ 0 0
□□□□□ h□□□ (4.5.2)

where f_1 is the mark frequency and f_2 is the space frequency. A_c is the carrier amplitude and T is the bit period (Peebles, 1987). The mark frequency and space frequency were calculated from equation 4.5.3 so that they were equidistant from the centre frequency.

□□=□□+0.5+□ h□□ (4.5.3)

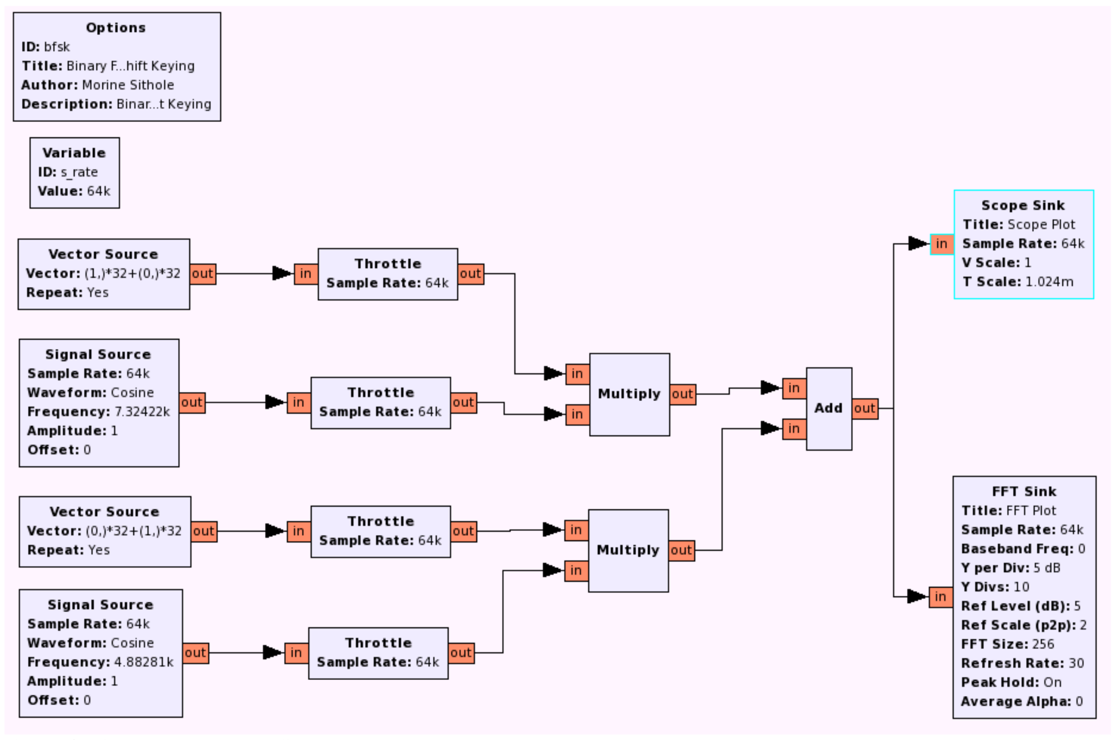
where f_i is either the mark frequency or the space frequency, h is the modulation index, f_b is the bit rate and i is binary 0 or 1.

□□=1□ (4.5.4)

The bit rate, of the information signal was 2.441 kHz. The mark frequency was calculated as 6.1025 kHz and the space frequency was calculated as 3.6615 kHz for a modulation index of 1. The spectrum of a BFSK signal is given by:

□□=□□2□1□−□1+□□2□1□+□1+□□2□2□−□2+□□2□2□+□2 (4.5.5)

Figure 4.5.1 shows the implementation of non-coherent BFSK in GRC.


Options
ID: bfsk
Title: Binary F...hift Keying
Author: Morine Sithole
Description: Binar...t Keying

Variable
ID: s_rate
Value: 64k

Vector Source
Vector: (1,)*32+(0,)*32
Repeat: Yes

Signal Source
Sample Rate: 64k
Waveform: Cosine
Frequency: 7.32422k
Amplitude: 1
Offset: 0

Vector Source
Vector: (0,)*32+(1,)*32
Repeat: Yes

Signal Source
Sample Rate: 64k
Waveform: Cosine
Frequency: 4.88281k
Amplitude: 1
Offset: 0

Throttle
Sample Rate: 64k

Multiply

Add

Scope Sink
Title: Scope Plot
Sample Rate: 64k
V Scale: 1
T Scale: 1.024m

FFT Sink
Title: FFT Plot
Sample Rate: 64k
Baseband Freq: 0
Y per Div: 5 dB
Y Divs: 10
Ref Level (dB): 5
Ref Scale (p2p): 2
FFT Size: 256
Refresh Rate: 30
Peak Hold: On
Average Alpha: 0

Figure 4.5.1- Non-coherent BFSK

4.6 Continuous Phase Frequency Shift Keying

4.6.1 Binary Frequency Shift Keying using a Voltage controlled oscillator

The output frequency of a voltage controlled oscillator (VCO) is controlled by the instantaneous amplitude of the baseband binary data signal that is applied to its input (Proakis, 1989). The centre frequency, f_c, lies halfway between the mark frequency, f_1, and space frequency f_2, in order to get an equal frequency shift for both binary levels when the input value is multiplied by the deviation sensitivity of the VCO.

□□□□□□□□□□□ □ ℎ□□□=□1−□2
(4.6.1.1)

□□=□1+□22 (4.6.1.2)

Δ□= □1−□22
(4.6.1.3)

To achieve phase continuity, the peak frequency deviation, Δ*f*, must be a multiple of half the product of the bit rate and the modulation index:

Δ□= ℎ□□2 (4.6.1.4)

In GNU Radio, a repetitive bit stream of 1’s and 0’s with a data rate of 8 kHz was fed into a VCO block with output amplitude of 1. From equation 4.6.1.4, Δf was 4 kHz for a modulation index of 1. With a centre frequency of 6 kHz, the VCO output shifted back and forth between the mark frequency of 10 kHz and space frequency of 2 kHz. The deviation sensitivity of the VCO was calculated as

▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯▯▯▯▯= ▯▯▯▯
▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯▯▯×▯ ▯▯▯▯▯▯▯▯ ▯▯▯
▯▯▯▯▯▯▯▯▯▯ (4.6.1.5)

=4000 ▯▯ ×▯ ▯▯▯▯▯▯▯▯ 1▯▯▯▯

=▯▯▯▯▯.▯▯ ▯▯▯▯▯▯▯▯
▯▯▯ ▯▯▯▯▯▯▯ ▯▯▯ ▯▯▯▯▯

4.6.2 Binary Frequency Shift Keying using the CPFSK block

Continuous phase Frequency Shift Keying was also implemented using the GNU Radio CPFSK modulator block. The CPFSK modulator performed continuous phase 2-level frequency shift keying on an input stream of unpacked bits according to the equation:

▯▯=2▯ ▯▯cos(2▯ ▯▯▯+∅▯,▯) (4.6.2.1)

∅▯,▯ =2▯ ℎ∞▯▯=∞∞▯▯▯▯−▯▯ ▯▯

▯▯=12▯ 0<▯<▯ 0 ▯▯▯▯▯ ℎ▯▯▯

E_b is the average energy per bit, T is the bit duration and h is the modulation index. Within a given bit interval, ∅▯,▯ increases or decreases linearly. This is what gives the CPFSK signal its continuous phase. *g(t)* is the waveform related to the information signal (Proakis, 1989). The modulation index, output amplitude and samples per symbol of the CPFSK modulator block were configured as 0.35, 1 and 2 respectively.

4.7 Binary Phase Shift Keying

In Binary Phase Shift Keying (BPSK), the phase of the carrier is modified according to changes in the level of the baseband data signal. Phase shifts in the carrier represent the change in the binary state of the data sequence according to the equation:

□□= □□□□□□2□ p□□□, □□□ □□□ 1
0<□<□ □□□□□□2□ p□□□+□ , □□□ □□□ 0
(4.7.1)

A_c is the carrier amplitude, f_c is the carrier frequency and T is the bit duration (Peebles, 1987). BPSK was achieved through use of the GNU Radio ‘delay’ block to provide the phase delay of π radians for bit 0. The frequency response of the BPSK signal was given by:

□□=□□2□□−□□+□□2□□+□□ (4.7.2)

The bandwidth between the first nulls around the carrier frequency must be $2f_b$.

4.8 Differential Phase Shift Keying Simulation

Differential Binary Phase Shift keying (DBPSK) uses phase angles relative to the phase in the immediately preceding bit period to modulate the current bit being transmitted. If the input binary bit is 1, the differentially encoded output is left unchanged with respect to the previous bit. If the input bit is 0, the differentially encoded output is changed by adding π radians to the current phase (Haykin, 2001). DBPSK can be represented by the equations:

□1□=2□□□□cos2□□□□, 0<□<□, □□□ □□□ 12□□□□cos2□□□□,
□<□<2□ (4.8.1)

□2□=2□□□□cos2□□□□, 0<□<□, □□□ □□□ 02□□□□cos2□□□□+□,
□<□<2□ (4.8.2)

$s_1(t)$ denotes the DBPSK signal when the input bit is 1 and $s_2(t)$ denotes the DBPSK signal when the input bit is 0.

DBPSK was implemented in GNU Radio through use of the ‘dbpsk_mod’ and ‘dbpsk_demod’ blocks. The ‘dbpsk_mod’ block was used to perform Raised-Root Cosine-filtered DBPSK modulation with Gray coding. The output of the ‘dbpsk_mod’ block was a complex modulated signal at baseband. The ‘dbpsk_demod’ block performed differential coherent detection of the DBPSK signal to output a stream of demodulated bits. A ‘file source’ block was used to read the data stream from a test file. The data stream was first encoded before it was fed into the ‘dbpsk_mod’ block. A ‘file sink’ block was used to write the demodulated data stream to an output file.

During simulation, channel noise effects were implemented by an Additive White Gaussian Noise (AWGN) source added to the modulated DBPSK signal. The noise level was gradually increased from 0 to determine the level at which the file contents were lost and could not be detected. The time-bandwidth product of the modulated signal, BT, was varied and the frequency spectrum changes were observed.

[illegible] = [illegible] 3 [illegible] × [illegible] (4.8.3)

where $f_{b\ 3dB}$ is the 3 dB baseband bandwidth and T is the bit period.

4.9 Using the USRP2 to transmit Differential Phase Shift Keyed signals

The setup in figure 4.1 was used to transmit a 480 Hz sinusoidal signal from one host computer to the other. The simulation in section 4.8 was modified by adding an interface to the transmit USRP2 and the receive USRP2. For the transmit USRP2, the interpolation value was chosen to be 512. Using equation 3.1.4, the maximum output sampling rate of the transmitted signal was calculated as:

[illegible] = 100 000 000 [illegible] 512 = [illegible] [illegible].[illegible] [illegible]

An IF bandwidth of 97 656.25 Hz was chosen to avoid loss of spectral information. The IF Ethernet data rate was:

[illegible] = 4 [illegible] × (97 656.25 [illegible])

= 390 625 [illegible] × 8 [illegible]

= [illegible].[illegible] [illegible]

The decimation value of the receive USRP2 was set to 512 and the output sample rate was set to 97 656.25 Hz. The sinusoidal signal was transmitted over the air interface at 2.44 GHz, a carrier frequency in the range of the ISM band. Using the same parameters for each USRP2, the contents of a test file were then transmitted from one host computer to the other and the received file was compared to the original file.

4.10 Gaussian Minimum Shift Keying Simulation

Gaussian minimum shift keying (GMSK) is a type of continuous-phase frequency shift keying modulation technique (Haykin, 2001). GMSK employs 1 bit per symbol. So the number of symbols is

▯▯▯▯▯▯ ▯▯ ▯▯▯▯▯▯▯=21=2 ▯▯▯▯▯▯▯

In this simulation, a ‘wavfile_source’ block was used to read the data stream from a 16 kHz, 8-bit Microsoft ‘.wav’ file. The bit rate, f_b, of the ‘.wav’ file was:

▯▯=16 ▯▯▯×8 ▯▯▯▯ ▯▯▯ ▯▯▯▯▯▯=▯▯▯ ▯▯▯▯

▯▯▯ ▯▯▯▯=▯▯▯▯ ▯▯▯ ▯▯▯▯▯▯×▯▯▯▯▯▯ ▯▯▯▯

(4.10.1)

therefore,

▯▯▯▯▯▯ ▯▯▯▯=128 000 ▯▯▯1 ▯▯▯ ▯▯▯ ▯▯▯▯▯▯=▯▯▯ ▯▯▯▯ ▯▯▯▯▯▯▯ ▯▯▯ ▯▯▯▯▯▯

The sampling rate was set to twice the bit rate, that is, 256 kHz.

▯▯▯▯▯▯ ▯▯▯▯=▯▯▯▯▯▯▯ ▯▯▯ ▯▯▯▯▯▯×▯▯▯▯▯▯ ▯▯▯▯

(4.10.2)

therefore,

▯▯▯▯▯▯▯ ▯▯▯ ▯▯▯▯▯▯=256 000 ▯▯▯▯128 000 ▯▯▯▯ ▯▯▯ ▯▯▯ ▯▯▯ =▯ ▯▯▯▯▯▯▯ ▯▯▯ ▯▯▯▯▯▯

The output from the ‘wavfile_source’ block was encoded by the ‘packet_encoder’ block before it was directed to the ‘gmsk_mod’ block. In the ‘gmsk_mod’ block, the data stream was processed by a ‘simple_framer’. The ‘simple_framer’ placed the sync word, 0x40952416, before each byte of data. The output of the ‘simple_framer’ was separated into individual bits, converted into polar bits, shaped with a Gaussian Low pass filter (LPF) and then sent to the input of a Voltage Controlled Oscillator. The Gaussian LPF introduced a frequency response with narrow bandwidth and sharp cutoff frequency. The pulse response, *g(t)*, of the Gaussian LPF is expressed as the difference between the two complementary error functions (erfc):

□□=12□□□□□2□□□102□□□□□−12−□□□□□2□□□102□□□□□+12 (4.10.3)

where f_b is the bit rate and T is the bit duration (Haykin, 2001). The VCO output was a complex GMSK signal at baseband. The GMSK signal was then passed through the 'gmsk_demod' block via a noisy channel simulator. The 'gmsk_demod' block demodulated the GMSK data at baseband. A 'packet_decoder' block decoded the demodulated data before rational re-sampling. The Rational Re-sampling FIR filter block was used to change the sampling rate of the '.wav' signal to the sampling rate of the 'audio_sink' block. The interpolation-to-decimation ratio of the rational re-sampler was

□□□□□□□□□□□□□−□□−□□□□□□□□□□ □□□□□=256 000 □□44 100 □□=□□□□□□□

The 'audio_sink' block output the '.wav' signal to the host computer's sound card. The noise level of the channel simulator was gradually increased from 0 to determine the level at which the '.wav' signal could not be detected. The time-bandwidth product was varied to determine the changes in the frequency response of the modulated signal.

4.11 Using the USRP2 to transmit Gaussian Minimum Shift Keyed signals

The setup in figure 4.1 was used to transmit the 16 kHz, 8-bit Microsoft '.wav' signal from one host computer to the other. The interpolation and decimation values of the USRP2's were set to 196. Using equation 3.1.4, the maximum output sampling rate of the transmitted signal was:

□□□□□□□ □□□□□□ □□□□=100 000 000 □□□□□□□ □□□ □□□□□□196=□□□ □□□.□ □□□

An IF bandwidth of 256 kHz was selected, to avoid any loss of spectral information. The IF Ethernet data rate was:

□□ □□□□ □□□□= 4 □□□□□ □□□
□□□□□□×(256 000 □□□□□□□ □□□ □□□□□□)

=1 024 000 □□□□□ □□□ □□□□□□ ×8
□□□□ □□□ □□□□

=□.□□□ □□□□

The GMSK signal was transmitted over the air interface at 2.46 GHz and the frequency response of the signal was observed at each stage of the receiver.

4.12 Direct Sequence Spread Spectrum

In a transmitter that employs the Direct Sequence Spread Spectrum (DSSS) technique, the narrowband data signal is used to modulate a wideband pseudo-random sequence with a chip rate higher than the data signal's bit rate. The resulting waveform is then binary phase shift-keyed before it is transmitted. The data signal is spread to make it less susceptible to interception and interference by signals of finite power. The Fourier transform of the pseudo-random sequence's autocorrelation function is given by

□□=1□2□□+1+□□2□=−∞∞□□□□2□□□□−□□□□ (4.12.1)

where X is the period of the maximum-length pseudo-random sequence and T_c is the chip duration of the pseudo-random sequence. At the receiver the direct-sequence spread BPSK signal is demodulated and then de-spread by a locally generated pseudo-random sequence to recover the baseband signal. The receiver's pseudo-random sequence is identical to and in synchronization with the transmitter's pseudo-random sequence. The same results are achieved if spectrum spreading is performed after binary phase shift keying in the transmitter because both operations are linear. If the binary phase shift keyed signal is

□□=□ □□(□) (4.12.2)

where *m(t)* is the data signal and *c(t)* is the carrier signal, the DSSS signal is then given by

$$v(t) = m(t)g(t) \tag{4.12.3}$$

where *g(t)* is the pseudo-random binary sequence (Haykin, 2001; Proakis, 1989). As *v(t)* is transmitted along the communication channel, it is corrupted by noise. The received signal is given by

$$r(t) = v(t) + n(t) \tag{4.12.4}$$

$$r(t) = m(t)g(t) + n(t)$$

where *n(t)* is the noise. De-spreading *r(t)* with the pseudo-random binary sequence *g(t)* will give

$$r(t)g(t) = m(t)g^2(t) + n(t)g(t) \tag{4.12.5}$$

The effect of the receiver's pseudo-random sequence is to spread the interfering signals and noise, while at the same time it de-spreads the wanted signal because the square of *g(t)* is equal to 1. The ratio of the pseudo-random sequence's chip rate to the data signal's bit rate is defined as the processing gain (PG).

$$PG = \frac{R_c}{R_b} \tag{4.12.6}$$

DSSS was accomplished by using the Galois Linear Feedback Shift Register (LFSR) pseudo-random source block, 'gr_glfsr_source'. The length of the LFSR was set to 4, making the period of the maximum-length sequence to become:

$$N = 2^4 - 1 = 15$$

The chip duration of the 'gr_glfsr_source' block was related to the bit period of the data signal by

$$T_c = \frac{T_b}{N} = T_b \times \frac{1}{15} \tag{4.12.7}$$

A data signal with the bit sequence 100011110001101 was used in the simulation. The bit duration of the data signal was 5 μs, making the chip duration 0.33 μs. The expected processing gain was

$$PG = 10\log_{10}\frac{5}{0.33} =$$ [illegible]

The performance of DSSS in the presence of noise and interference was investigated by the use of an AWGN source. Bit Error Rate (BER) measurements were taken for various levels of signal-to-noise ratio (SNR) through use of the ‘grc_blks2.error_rate’ block. The ‘grc_blks2.error_rate’ block compared the demodulated data signal to the original data signal and calculated the BER over a million samples. The ‘grc_blks2.error_rate’ block accumulated the 1 million samples before calculating the BER value according to the equation:

[illegible]=[illegible]

[illegible] (4.12.8)

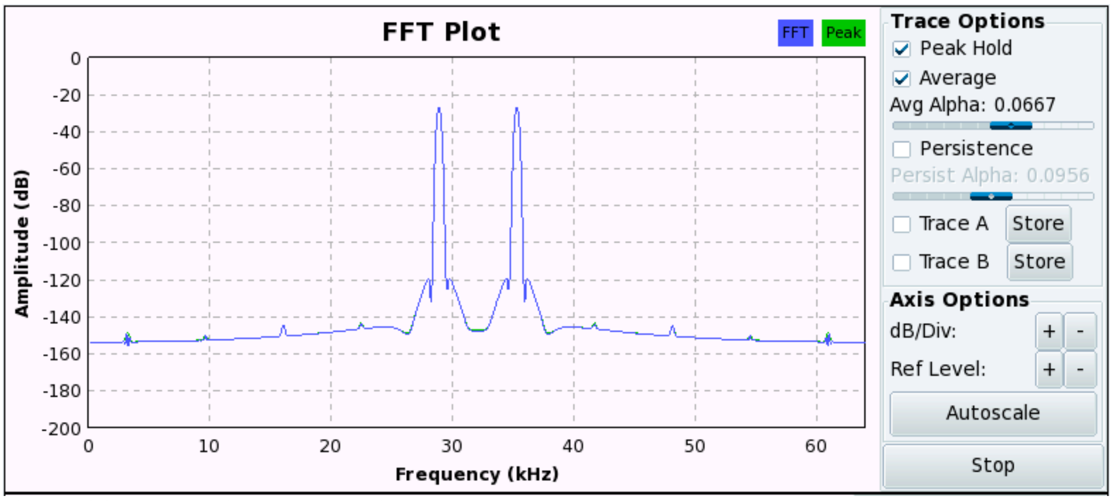
Chapter five

5 Experimental Results

The test results obtained from the simulations described in chapter four are reported in the following sections.

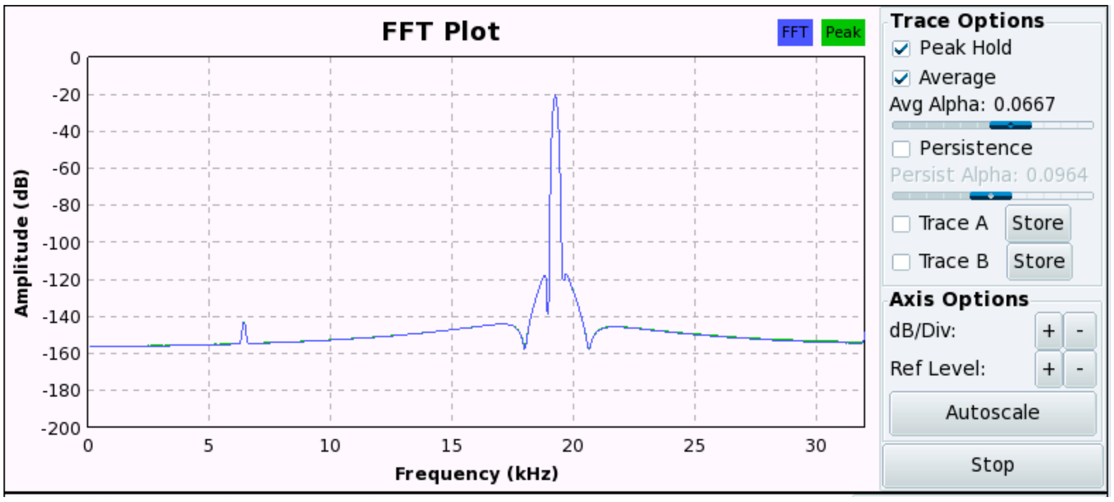
5.1 Sampling Theorem Simulation

The FFT plots of the properly sampled 3.2 kHz signal and the undersampled 3.2 kHz signal are shown in Figure 5.1.1 and Figure 5.1.2. Figure 5.1.1 shows the desired frequency translation to 35.2 kHz and 28.8 kHz with correct sampling. Figure 5.1.2 shows that sampling an analog signal at a sampling frequency less than twice its highest frequency component will result in aliasing whereby all frequency components higher than the sampling frequency are translated to undesired parts of the spectrum, rendering it impossible to recover the signal successfully (Haykin, 2001).



[✓] Peak Hold
[✓] Average
Avg Alpha: 0.0667
[ ] Persistence
Persist Alpha: 0.0956
[ ] Trace A
[ ] Trace B

Figure 5.1.1- FFT plot with proper sampling

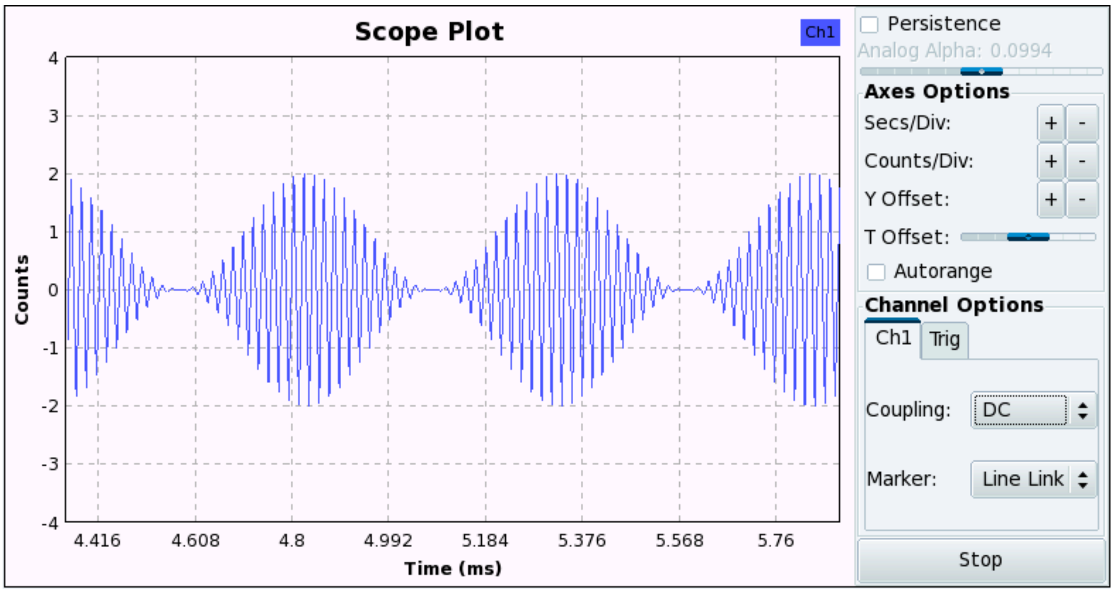


[✓] Peak Hold
[✓] Average
Avg Alpha: 0.0667
[ ] Persistence
Persist Alpha: 0.0964
[ ] Trace A
[ ] Trace B

Figure 5.1.2- FFT plot with undersampling

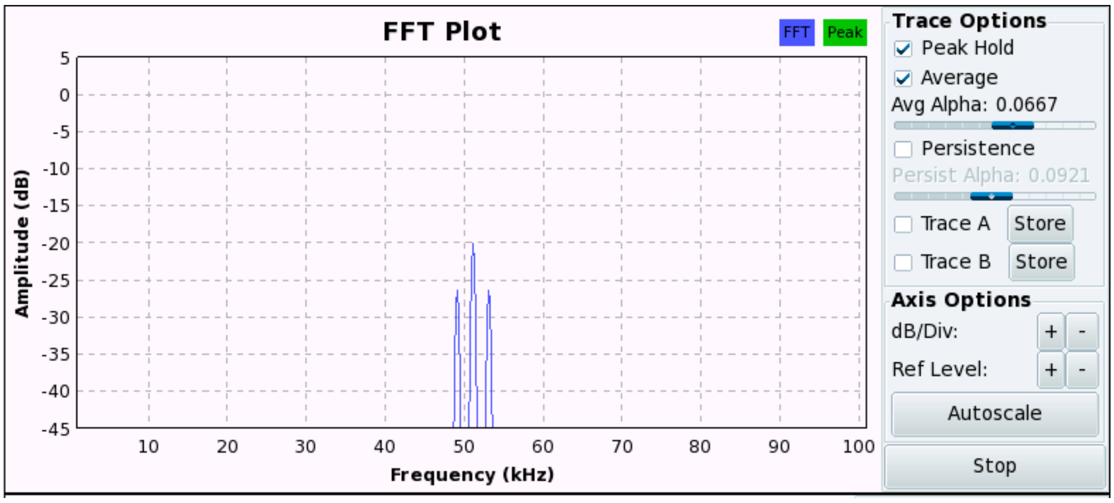
5.2 Amplitude Modulation

The output of the amplitude modulator gave a satisfying result of the modulated signal in the time domain as shown in figure 5.2.1. The frequency domain, depicted in figure 5.2.2, showed the two sidebands of the modulated signal 6 dB below the carrier. The baseband signal modulation with the carrier signal resulted in a shift of the baseband signal in the frequency domain by 50 kHz. The sidebands were adjacent the carrier 2 kHz above and below it.



Persistence
Analog Alpha: 0.0994
Axes Options
Secs/Div:
Counts/Div:
Y Offset:
T Offset:
Autorange
Channel Options
Ch1 Trig
Coupling: DC
Marker: Line Link

Figure 5.2.1- Time domain of amplitude modulated signal

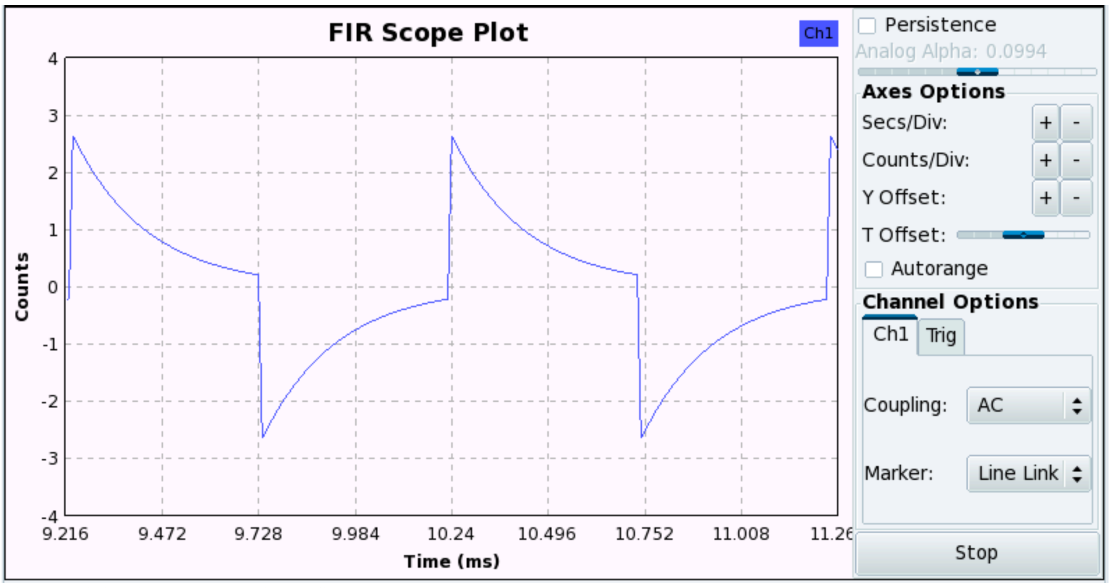


Trace Options
Peak Hold ✓
Average ✓
Avg Alpha: 0.0667
Persistence
Persist Alpha: 0.0921
Trace A Store
Trace B Store
Axis Options
dB/Div:
Ref Level:

Figure 5.2.2- Frequency domain of amplitude modulated signal

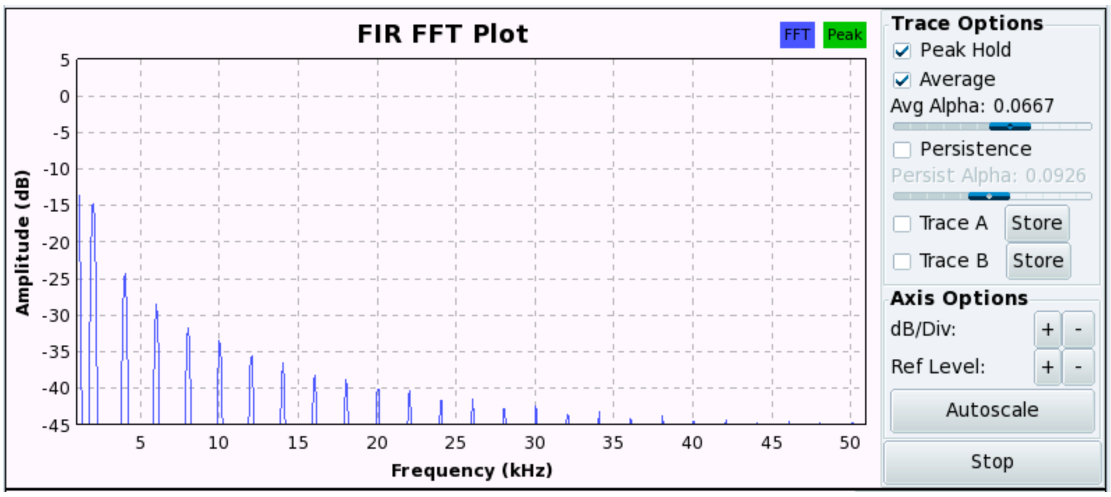
5.3 FIR and IIR filtering

The effects of non-linear phase response were observed in the distorted square wave that was output from the IIR filter while the FIR filter showed phase linearity. The frequency response of each output is shown in figures 5.3.2 and 5.3.4.



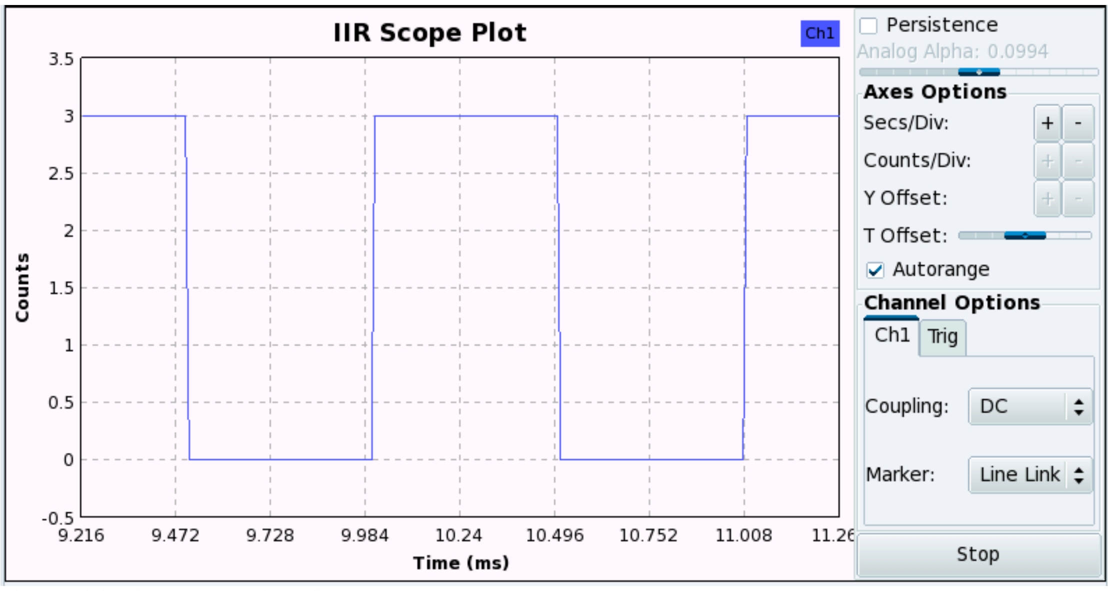
☐ Persistence
Analog Alpha: 0.0994
☐ Autorange
Coupling: AC
Marker: Line Link

Figure 5.3.1- Time domain of FIR filter



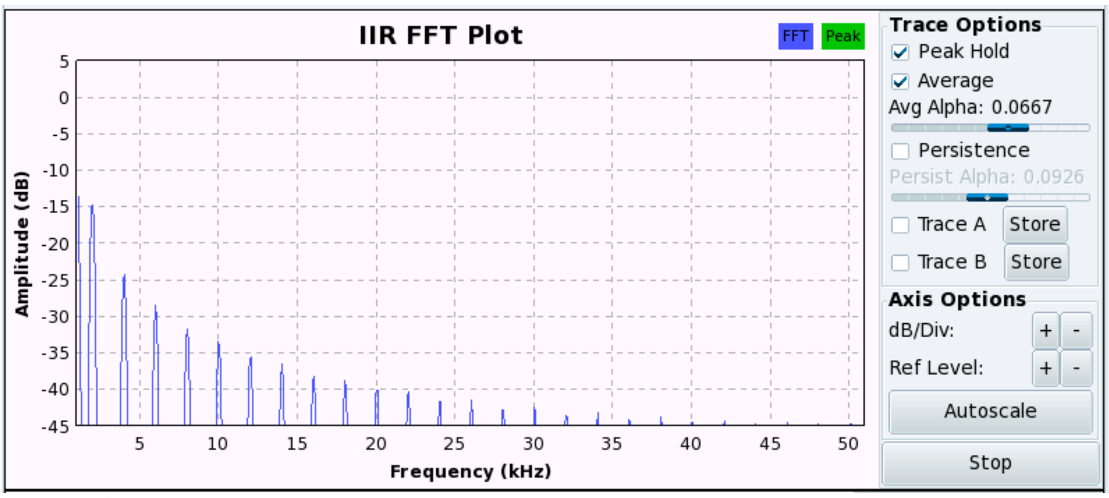
✓ Peak Hold
✓ Average
Avg Alpha: 0.0667
☐ Persistence
Persist Alpha: 0.0926
☐ Trace A
☐ Trace B

Figure 5.3.2- Frequency response of FIR filter



☐ Persistence
Analog Alpha: 0.0994
✓ Autorange
Coupling: DC
Marker: Line Link

Figure 5.3.3- Time domain of IIR filter

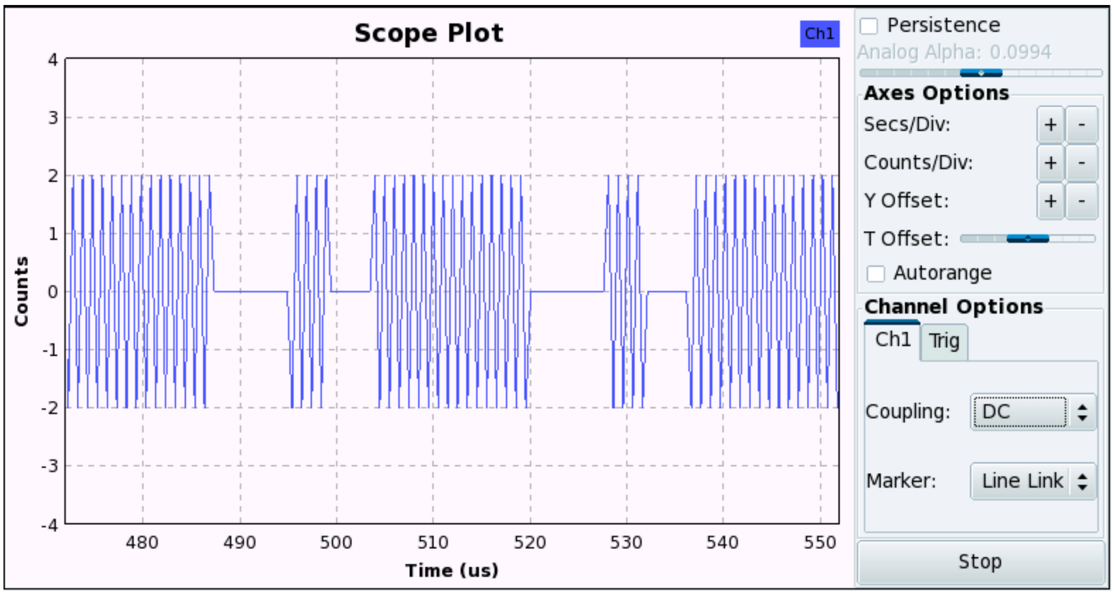


Peak Hold ✓
Average ✓
Avg Alpha: 0.0667
Persistence
Persist Alpha: 0.0926
Trace A
Trace B

Figure 5.3.4- Frequency response of IIR filter

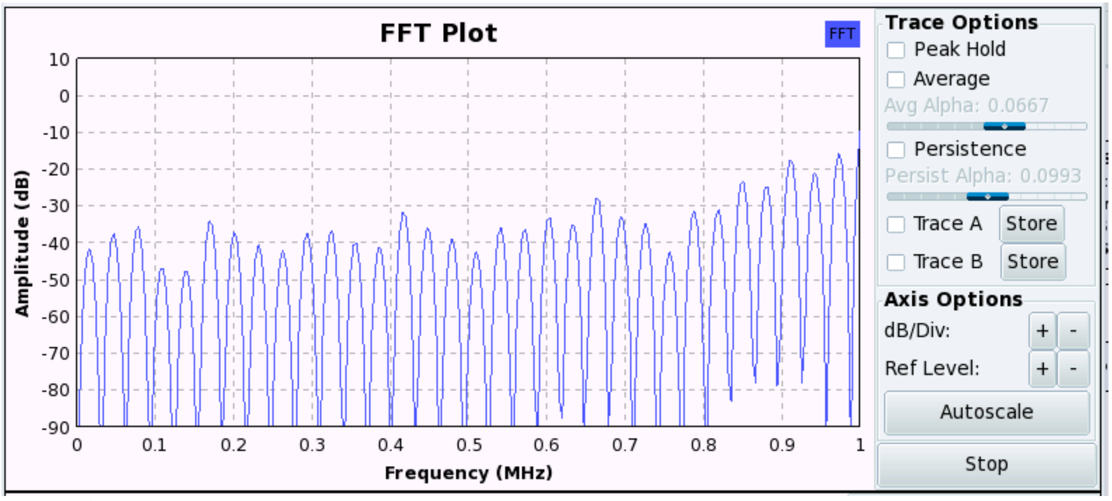
5.4 Amplitude Shift Keying

5.4.1 Binary Amplitude Shift Keying



Persistence
Analog Alpha: 0.0994
Autorange
Coupling: DC
Marker: Line Link

Figure 5.4.1.1- BASK signal in time domain



Peak Hold
Average
Avg Alpha: 0.0667
Persistence
Persist Alpha: 0.0993
Trace A
Trace B

Figure 5.4.1.2- Frequency domain of BASK signal

Figure 5.4.1.1 shows the BASK modulated signal in the time domain, with the amplitude changing in response to the amplitude of the information signal. Figure 5.4.1.2 depicts the BASK signal in the frequency domain. Because the amplitude of the BASK signal varies with the amplitude of the data signal, it is severely affected by noise, fading, and interference close to the carrier frequency (Peebles, 1987). Therefore BASK offers poor performance compared to other modulation schemes and is not used for data transmissions over long distances.

5.4.2 4-Level Amplitude Shift Keying

The resulting oscilloscope plot and FFT plot of the 4-ary signal was consistent with theory as shown in figure 5.4.2.1 and figure 5.4.2.2. There was a phase change between the negative and positive levels of each symbol. The results showed that 4-level ASK is less spectrally efficient than BASK because of the phase changes.

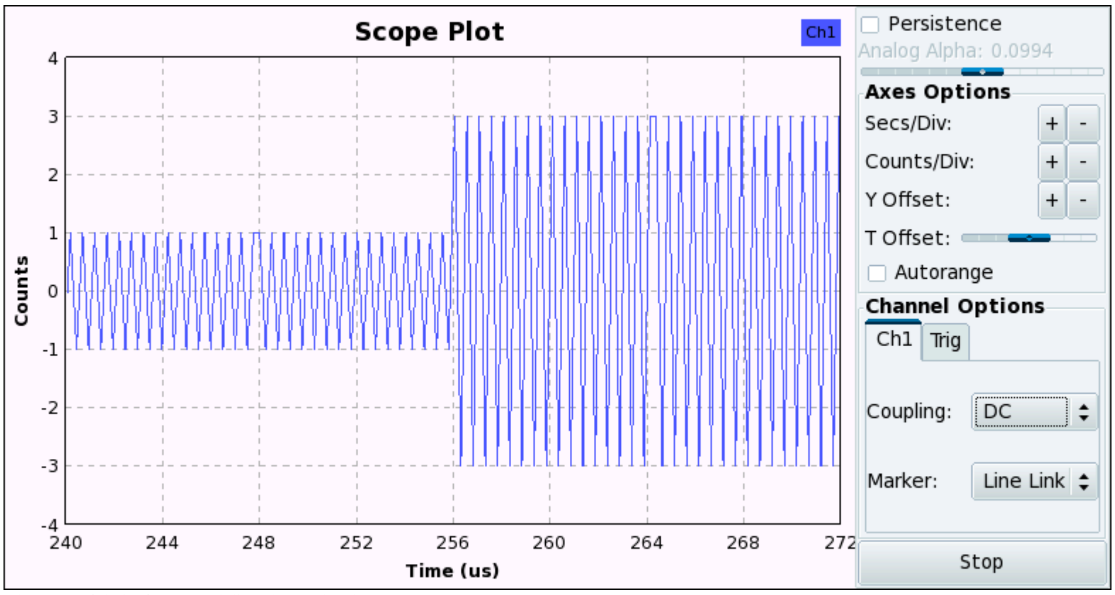


Figure 5.4.2.1- 4-Level ASK signal in time domain

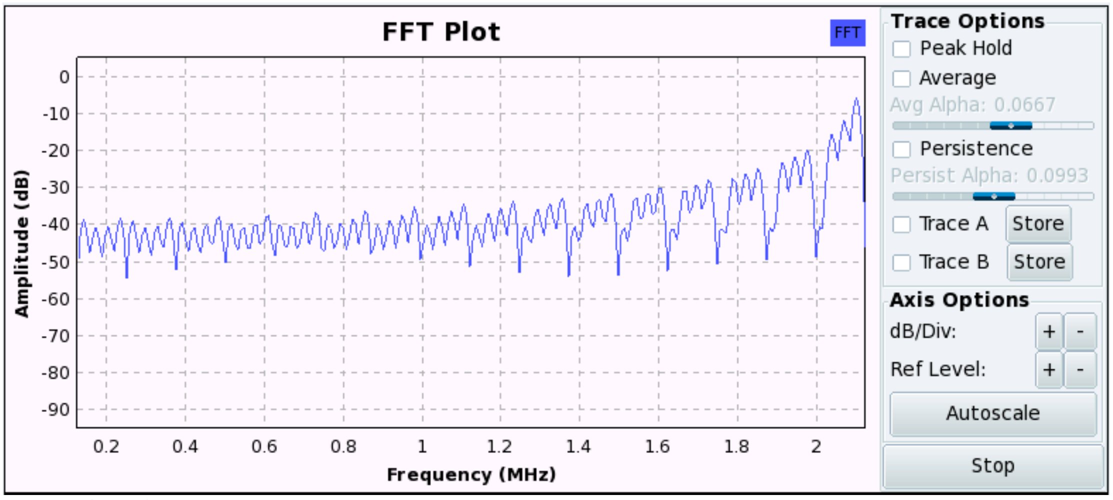


Figure 5.4.2.2- 4-Level ASK signal in frequency domain

5.5 Non-coherent Binary Frequency Shift Keying

Figure 5.5.1 shows the oscilloscope output of the non-coherent BFSK signal. Figure 5.5.2 shows the FFT output. It was observed that the amplitude of the BFSK signal did not change in the time domain. Only the frequency of the BFSK signal changed according to the binary level of the data signal. With the bit rate constant, the bandwidth of the BFSK signal increased as the modulation index increased (Peebles, 1987). But the higher the modulation index, the higher the Signal-to-Noise ratio (SNR). Phase discontinuity due to the instantaneous phase switches between binary levels introduced wideband frequency components as shown in figure 5.5.2.

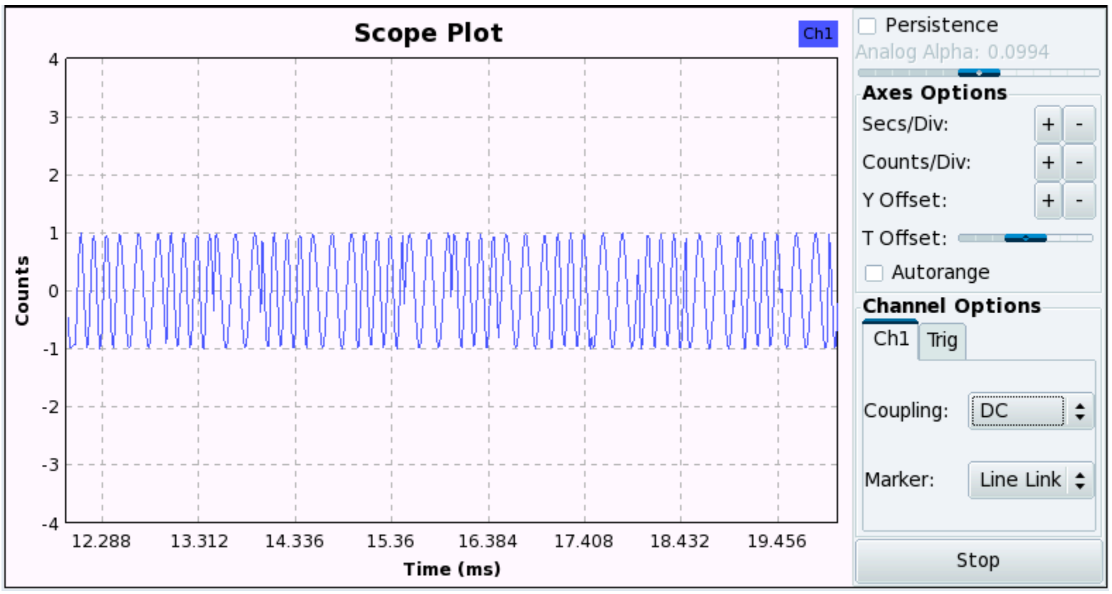


Figure 5.5.1- Time domain of non-coherent BFSK signal

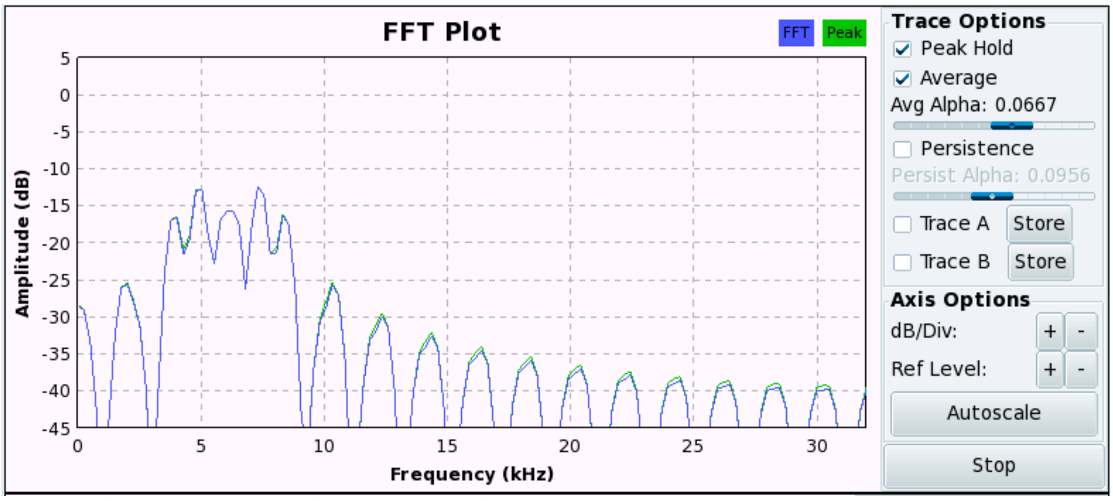


Figure 5.5.2- Frequency domain of non-coherent BFSK signal

When the frequency deviation was increased above the bit rate, a wideband BFSK signal of bandwidth equal to $2\Delta f$ was obtained. When the frequency deviation was decreased below the bit rate, a narrowband BFSK signal of bandwidth equal to $2f_b$ was obtained (Haykin, 2001).

5.6 Continuous Phase Binary Frequency Shift Keying

The CPFSK signal in the time and frequency domains is shown in figures 5.6.1 and 5.6.2. The CPFSK signal maintained constant amplitude in the time domain. In the frequency domain, the carriers f_1 and f_2 were shifted proportionally to the baseband data rate. The CPFSK waveform also included $f_1 + f_2$ and $f_1 - f_2$ and other harmonics.

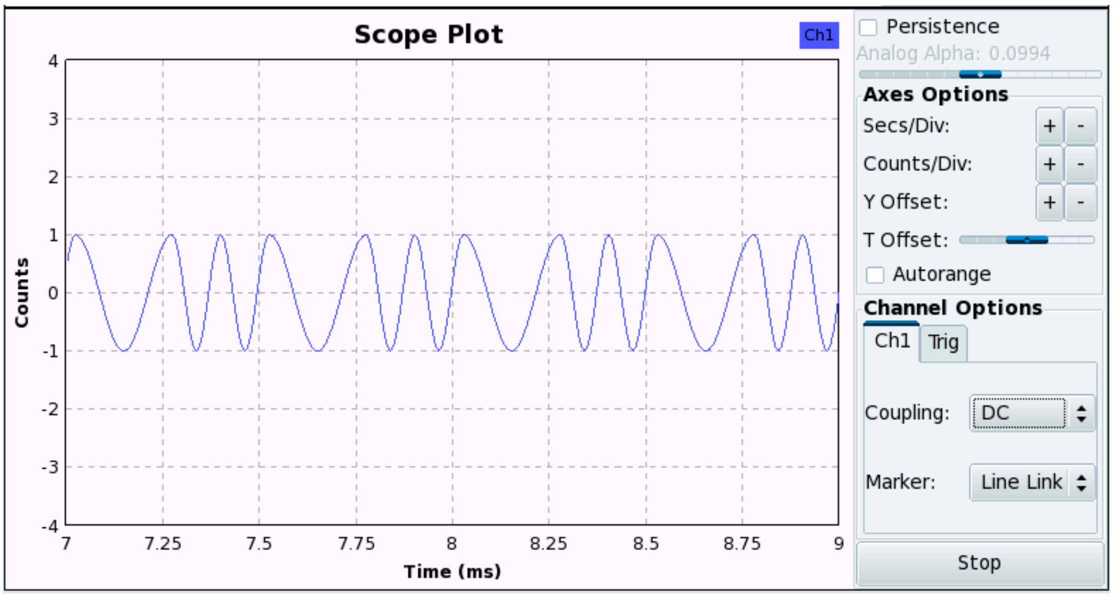


Figure 5.6.1- CPFSK signal in the time domain

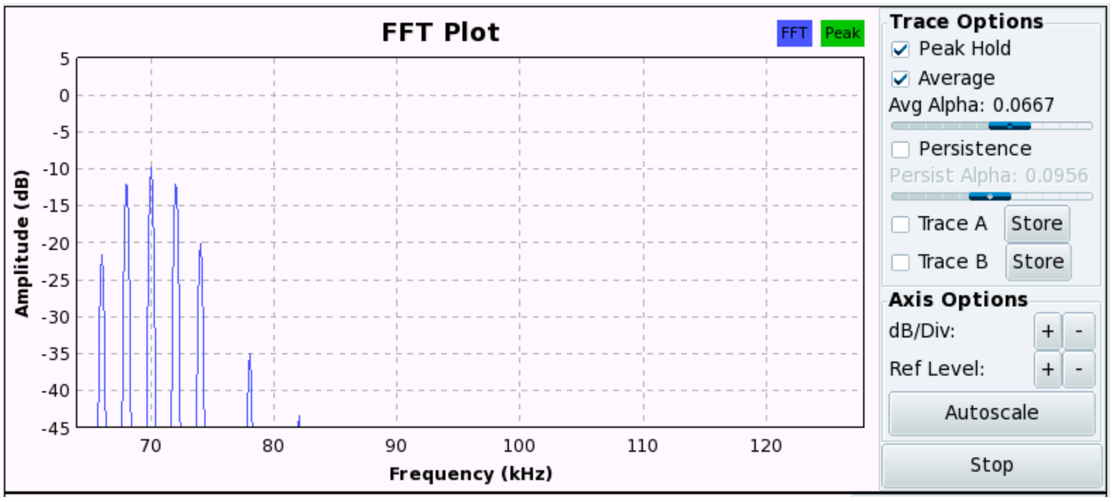


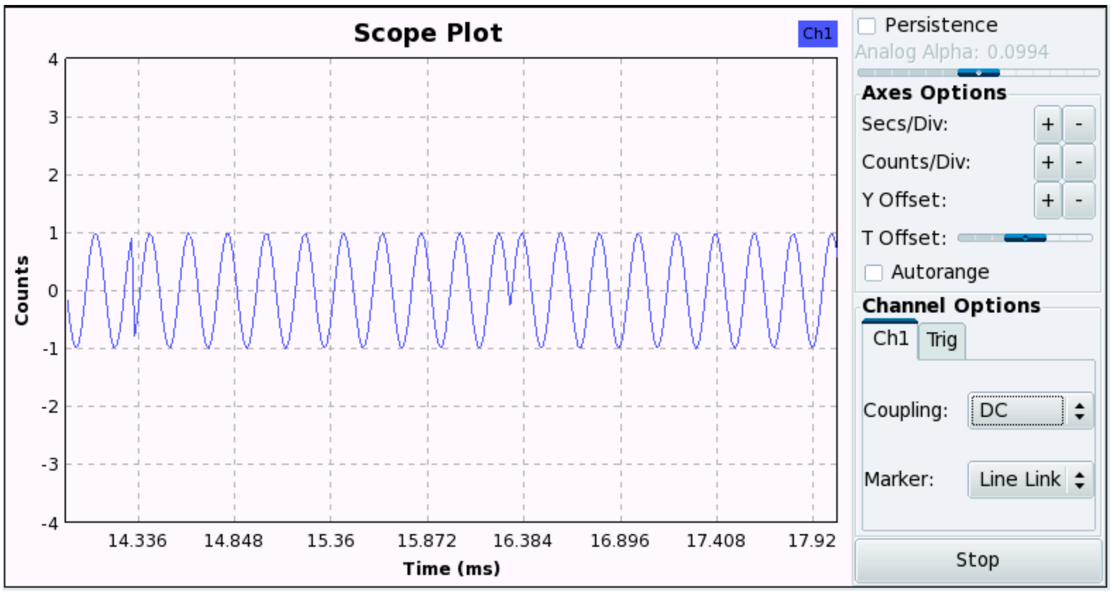
Figure 5.6.2- CPFSK signal in the frequency domain

Because the CPFSK waveform had no abrupt phase change and exhibited a constant envelope, it had better spectral efficiency than the non-coherent BFSK waveform (Proakis, 1989). CPFSK has improved spectral efficiency and improved spectral rolloff over BPSK and discontinuous phase FSK modulation techniques. This makes

CPFSK suitable for transmitting signals over a bandlimited channel. The constant amplitude of the CPFSK signal makes it suitable for use with nonlinear, multipath fading channels. CPFSK has better bit error performance than non-coherent FSK, but non-coherent FSK is easier to generate and is therefore used in many of the FSK transmissions.

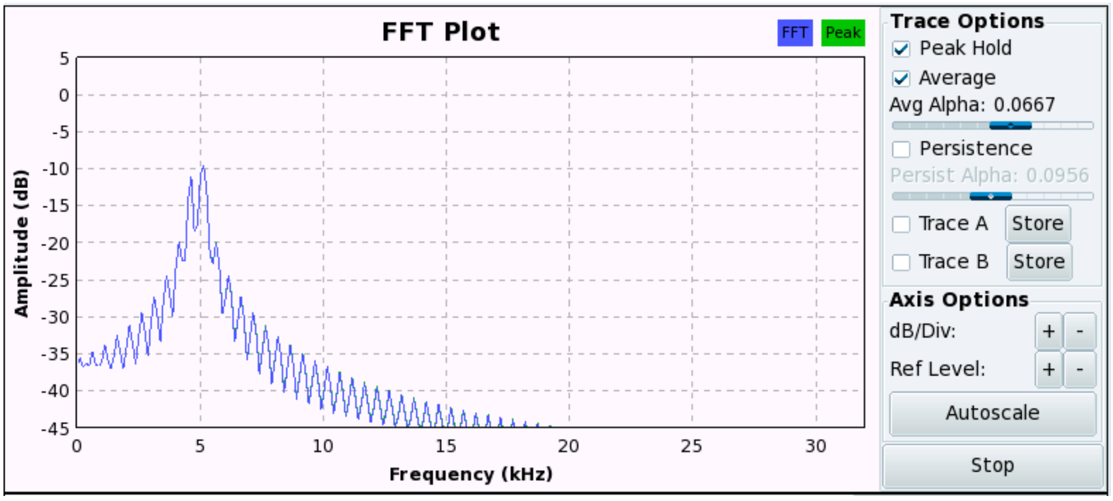
5.7 Binary Phase Shift Keying

The BPSK signal is shown in figures 5.7.1 and 5.7.2. Phase shifts at the bit edges of the data signal could be observed.



Persistence
Analog Alpha: 0.0994
Autorange
Coupling: DC
Line Link

Figure 5.7.1- BPSK signal in the time domain



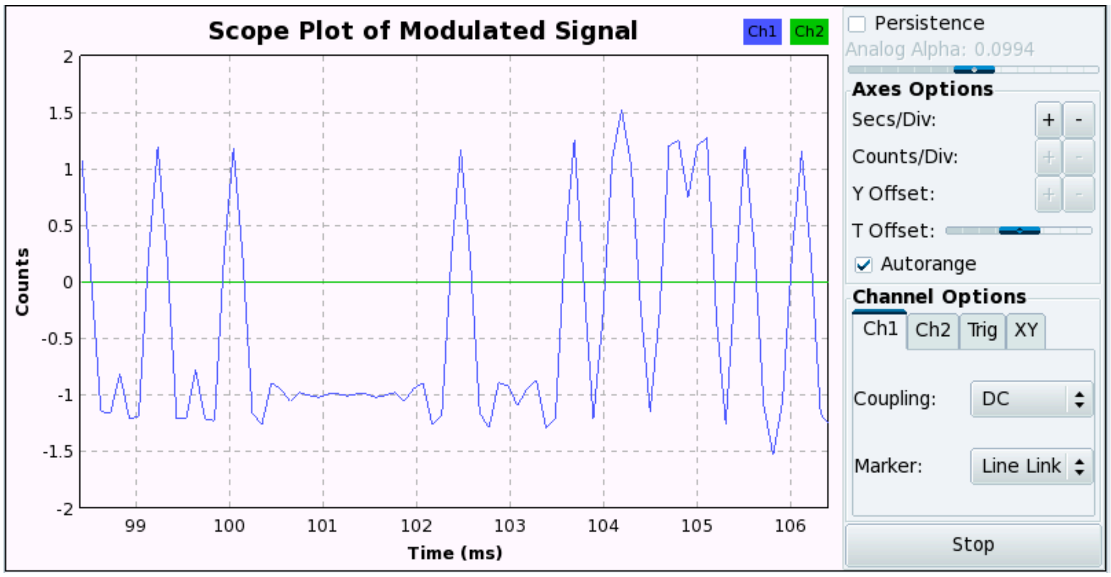
✓ Peak Hold
✓ Average
Avg Alpha: 0.0667
Persistence
Persist Alpha: 0.0956
Trace A Store
Trace B Store

Figure 5.7.2- BPSK signal in the frequency domain

The power and frequency efficiency was better when compared with BASK. The bandwidth between the first nulls around the carrier frequency was twice the bit rate.

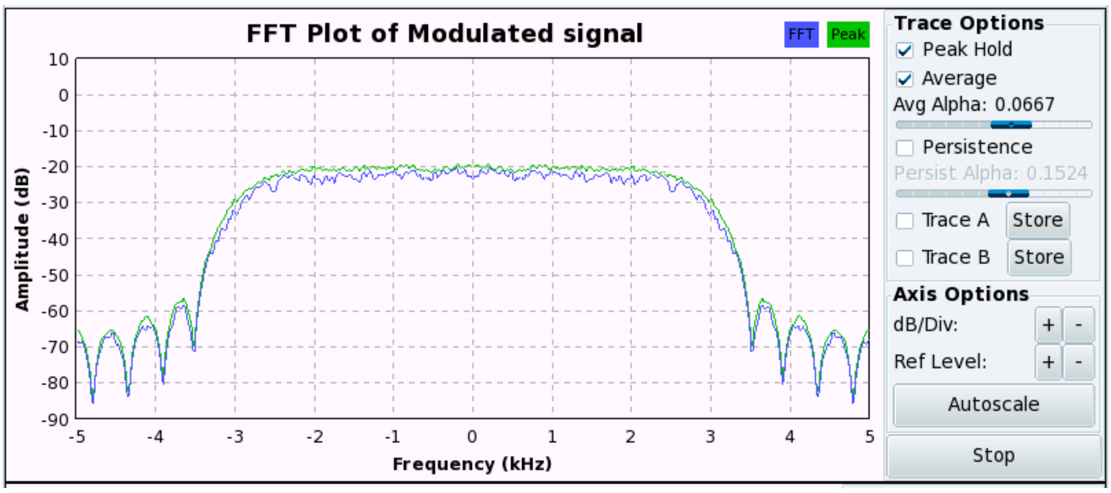
5.8 Differential Binary Phase Shift Keying Simulation

The simulations of DBPSK were consistent with theory. A plot of the DBPSK modulated signal is shown in figure 5.8.1. Figure 5.8.2 shows the spectrum display.



Persistence
Analog Alpha: 0.0994
Secs/Div:
Counts/Div:
Y Offset:
T Offset:
✓ Autorange
Coupling: DC
Marker: Line Link

Figure 5.8.1- Oscilloscope display of DBPSK signal



✓ Peak Hold
✓ Average
Avg Alpha: 0.0667
Persistence
Persist Alpha: 0.1524
Trace A
Trace B
dB/Div:
Ref Level:

Figure 5.8.2- Spectrum display of the DBPSK signal

The noise level at which some characters from the input test file were lost was 0.250 Volts. The noise level at which all of the characters were lost was 0.390 Volts. The differential phase encoded signal was demodulated reliably in the presence of high noise levels. Increasing the time-bandwidth product resulted in an increase in the bandwidth between the first nulls of the frequency spectrum.

5.9 Using the USRP2 to transmit Differential Phase Shift Keyed signals

Figure 5.9.1 shows the received signal after transmission of the 480 Hz tone. Figure 5.9.2 shows the spectrum of the received 480 Hz tone and figure 5.9.3 shows the constellation plot. The sine wave was recovered with no distortions as can be seen in figure 5.9.4. Figure 5.9.5 shows the frequency spectrum of the demodulated sine wave.

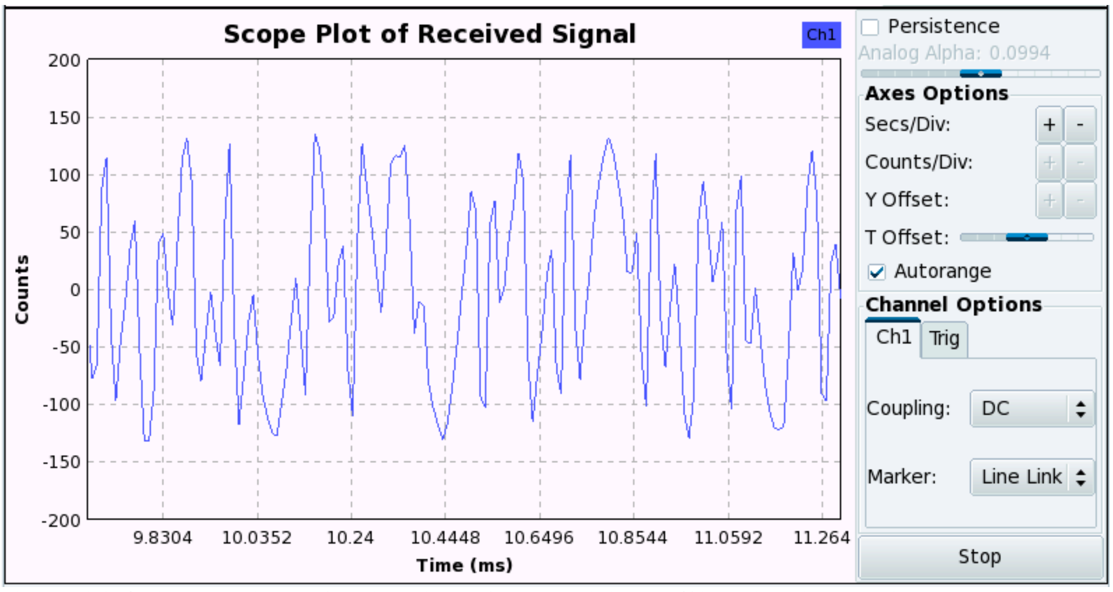


Figure 5.9.1- Oscilloscope display of received signal after transmitting the 480 Hz sinusoidal waveform

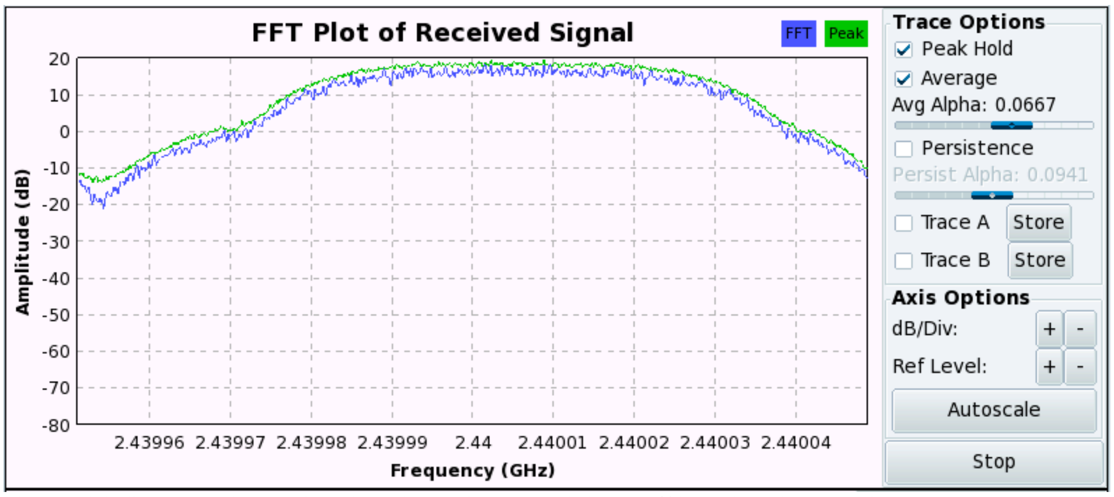


Figure 5.9.2- Spectrum display of received signal after transmitting the 480 Hz sinusoidal waveform

The gain of the transmit USRP2 and the receive USRP2 were varied between 0 dB and 20 dB to avoid signal attenuation in the air interface.

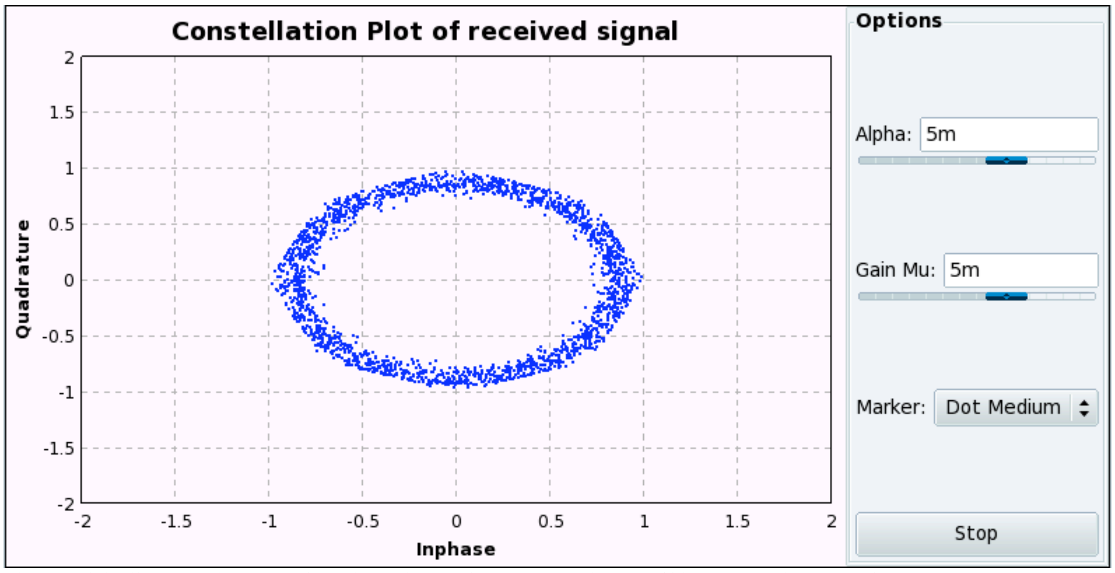


Figure 5.9.3- Constellation plot of received signal after transmitting the 480 Hz sinusoidal waveform

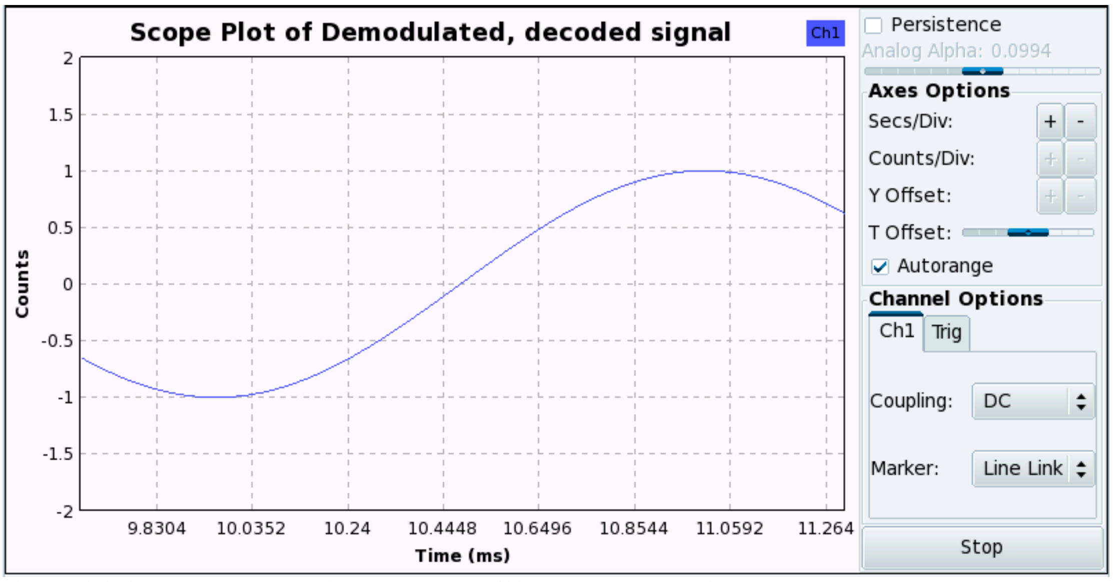


Figure 5.9.4- Oscilloscope display of the demodulated 480 Hz sinusoidal waveform

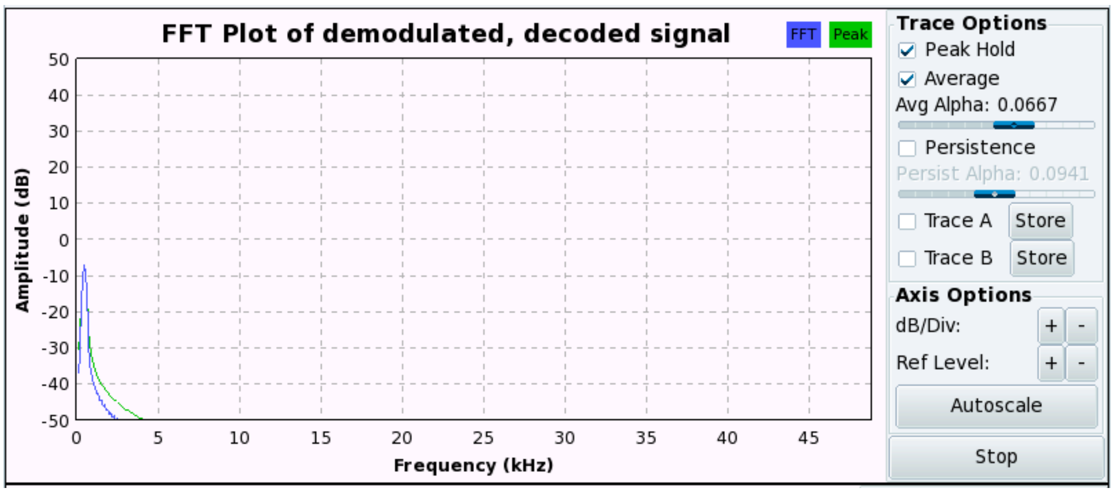
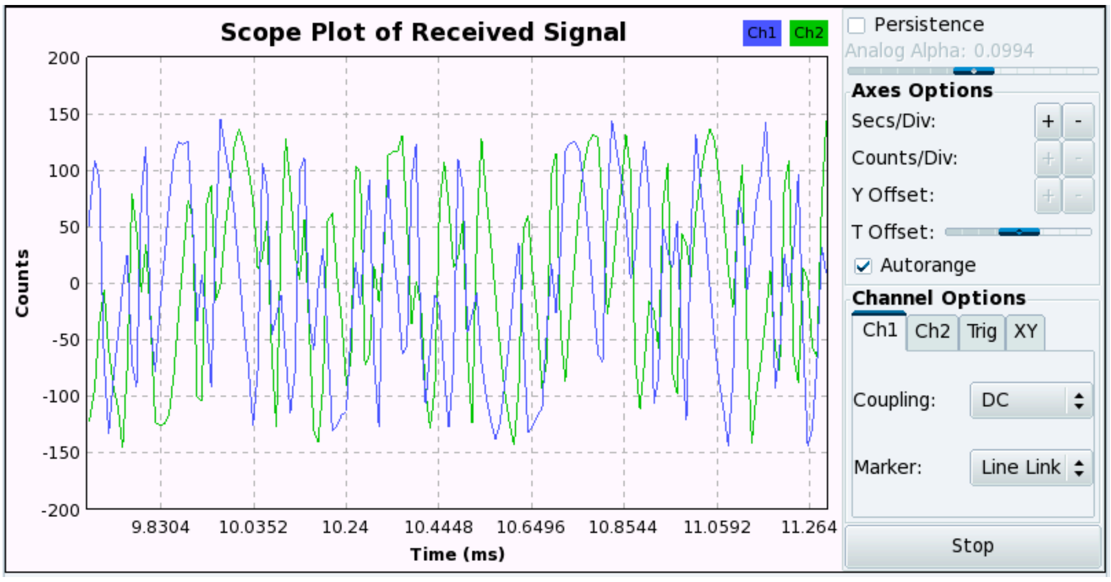


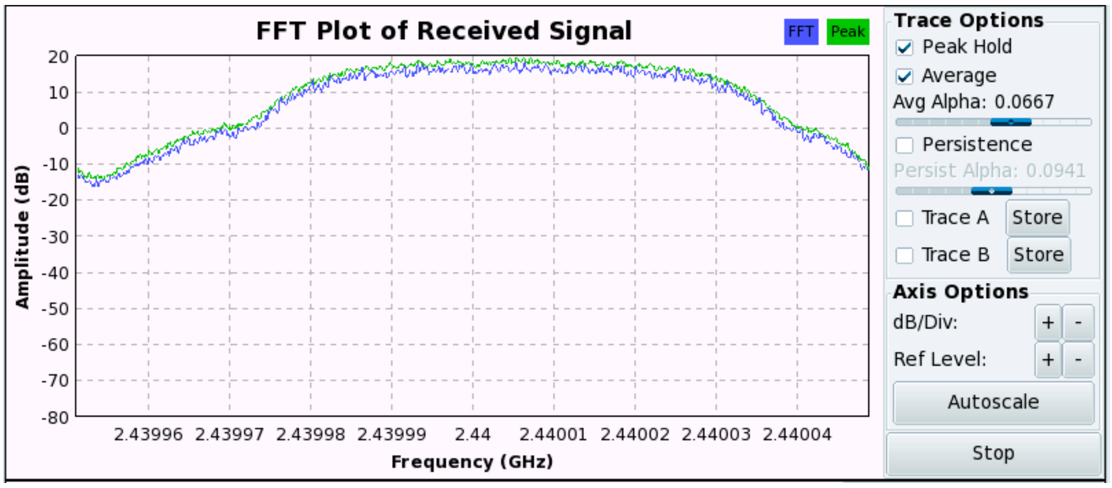
Figure 5.9.5- FFT display of the demodulated 480 Hz sinusoidal signal

The oscilloscope plot, FFT plot and constellation plot obtained from transmission and reception of the text file are shown in figures 5.9.6 to 5.9.8. During transmission of



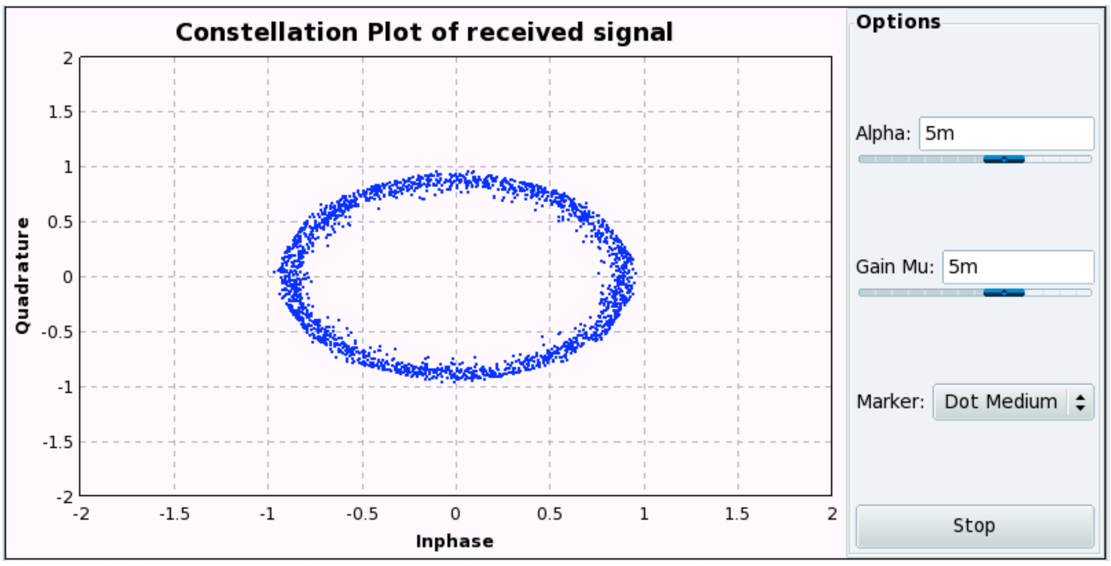
Persistence
Analog Alpha: 0.0994
Axes Options
✓ Autorange
Channel Options
Coupling: DC
Marker: Line Link

Figure 5.9.6- Oscilloscope display of received signal after transmitting the text file



Trace Options
✓ Peak Hold
✓ Average
Avg Alpha: 0.0667
Persistence
Persist Alpha: 0.0941
Trace A
Trace B
Axis Options

Figure 5.9.7- Spectrum display of received signal after transmitting the text file



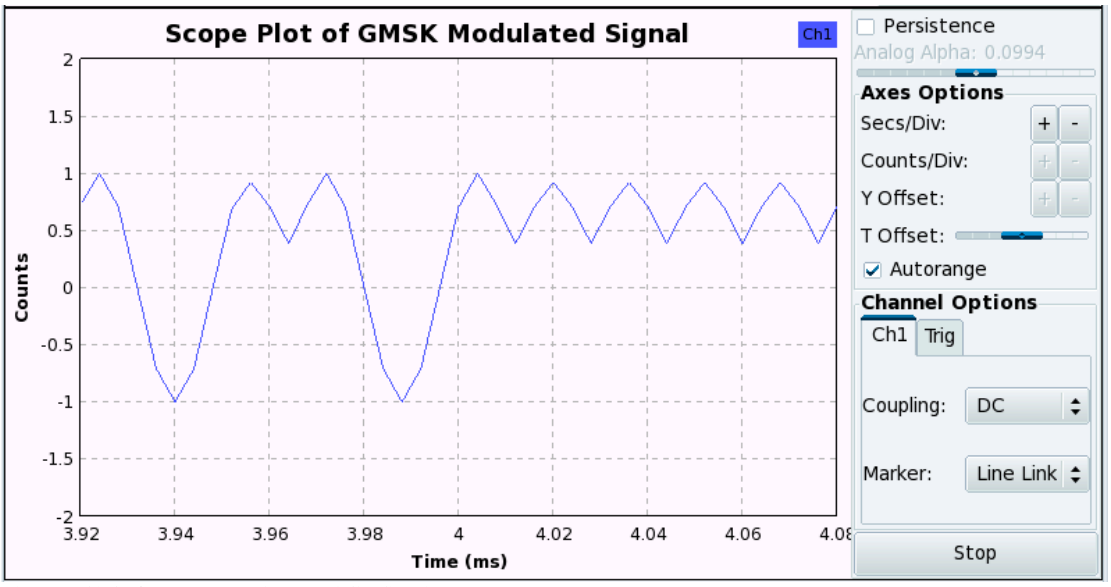
Options
Alpha: 5m
Gain Mu: 5m
Marker: Dot Medium

Figure 5.9.8- Constellation plot of received signal after transmitting the text file

the text file, it was observed that the first 14 characters of the file were not received. This was attributed to the noisy radio link.

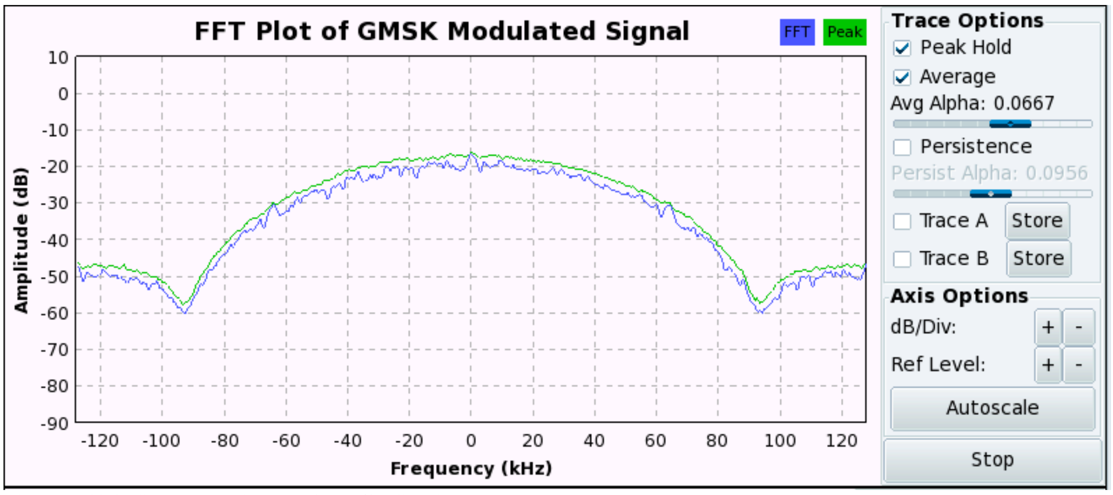
5.10 Gaussian Minimum Shift Keying Simulation

The simulations of GMSK were consistent with theory as shown in figures 5.10.1 and 5.10.2. The GMSK signal had a constant envelope and did not contain phase discontinuities.



Analog Alpha: 0.0994
☐ Persistence
✓ Autorange
Coupling: DC
Marker: Line Link

Figure 5.10.1- Oscilloscope display of GMSK signal



✓ Peak Hold
✓ Average
Avg Alpha: 0.0667
☐ Persistence
Persist Alpha: 0.0956
☐ Trace A
☐ Trace B

Figure 5.10.2- Spectrum display of the GMSK signal

The Gaussian LPF reduced the sideband power of the GMSK signal and exhibited higher spectral efficiency than DBPSK when comparing the spectrum of the data signal and the modulated signal. When the time-bandwidth product was less than unity, the transmit power was concentrated inside the passband and the power spectrum rolloff was faster. Higher frequency components of the GMSK signal were suppressed. The results supported the theory that GMSK reduces out-of-band interference between adjacent frequency channels. Decreasing the time-bandwidth product introduced inter-symbol interference because the time over which the Gaussian LPF was spread became longer. The output bit symbols from the Gaussian LPF had a longer duration than the bit interval of the original '.wav' signal. This made it difficult to distinguish between bit levels and the output voice signal became blurred (Haykin, 2001). The noise level at which there was no audio output was 0.175 Volts. The simulation showed that GMSK is less resilient to noise than DBPSK.

5.11 Using the USRP2 to transmit Gaussian Minimum Shift Keyed signals

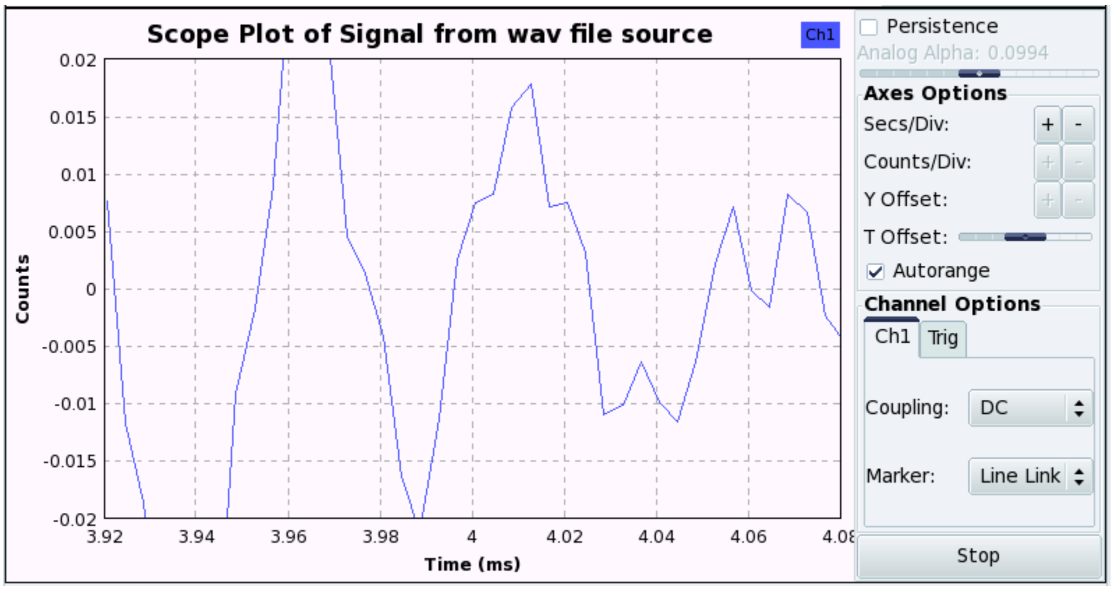


Figure 5.11.1- Oscilloscope plot of the repeated baseband '.wav' signal

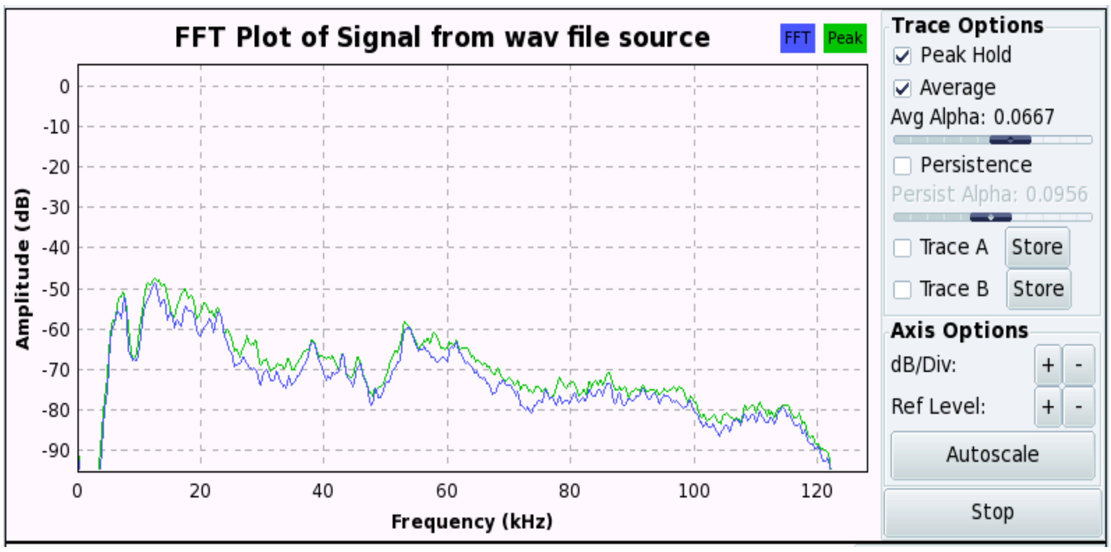
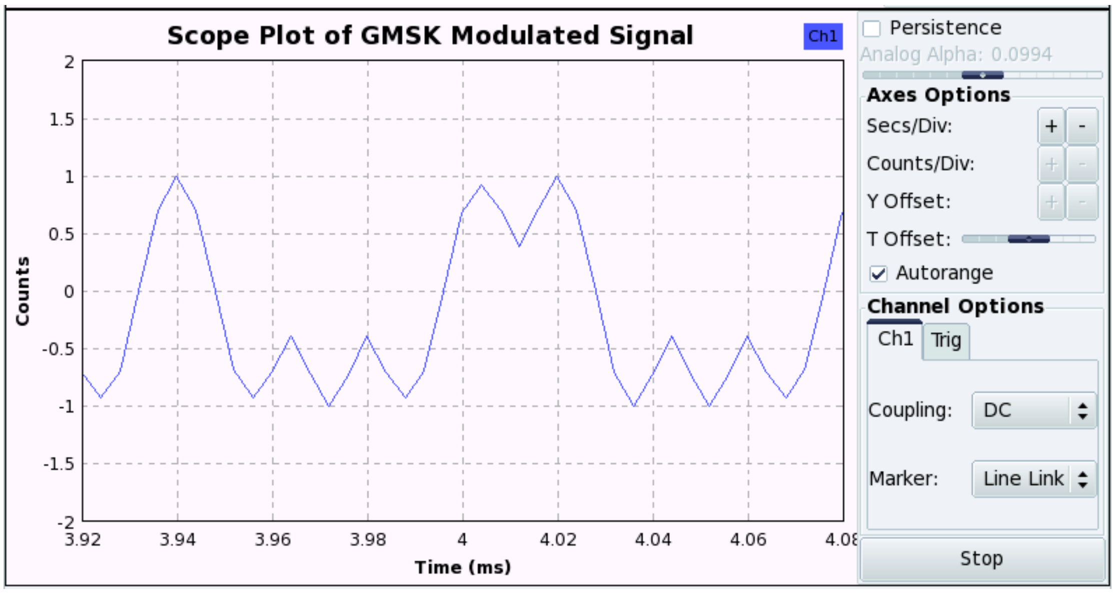


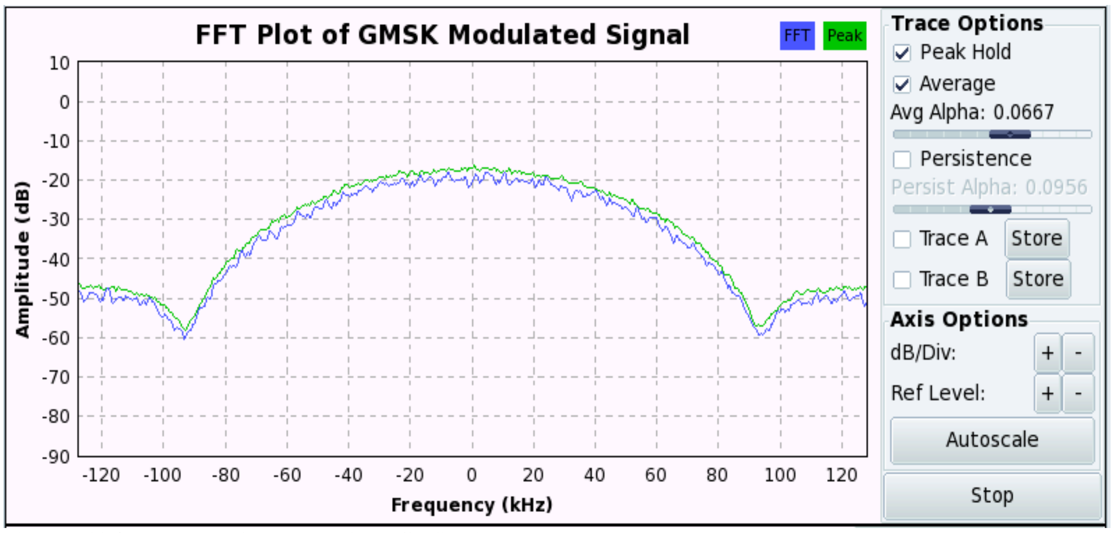
Figure 5.11.2- FFT plot of the repeated baseband '.wav' signal

Figure 5.11.1 and Figure 5.11.2 show the baseband repeated voice signal that was transmitted. Figure 5.11.3 and Figure 5.11.4 show the GMSK signal.



Analog Alpha: 0.0994
Autorange ✓
Coupling: DC
Marker: Line Link

Figure 5.11.3- Oscilloscope plot of the transmitted GMSK signal



Peak Hold ✓
Average ✓
Avg Alpha: 0.0667
Persist Alpha: 0.0956

Figure 5.11.4 - FFT plot of the transmitted GMSK signal

Figure 5.11.5 and Figure 5.11.6 show the attenuated GMSK signal that was received at the receiver end. Free space attenuation and noise lowered the received signal power. It was observed that GMSK required a lower power level than DBPSK in order to transmit the voice signal over the radio link reliably. The constellation diagram in figure 5.11.7 shows the graphical representation of each received symbol state. The constellation points were less scattered than those for DBPSK in figure 5.9.3 and 5.9.8 for the same transmission medium. A trade-off was made between spectral efficiency and the time domain performance in order to transmit the signal successfully. Figure 5.11.8 and Figure 5.11.9 show the demodulated voice signal. Errors arose from quantization of the sampled signal.

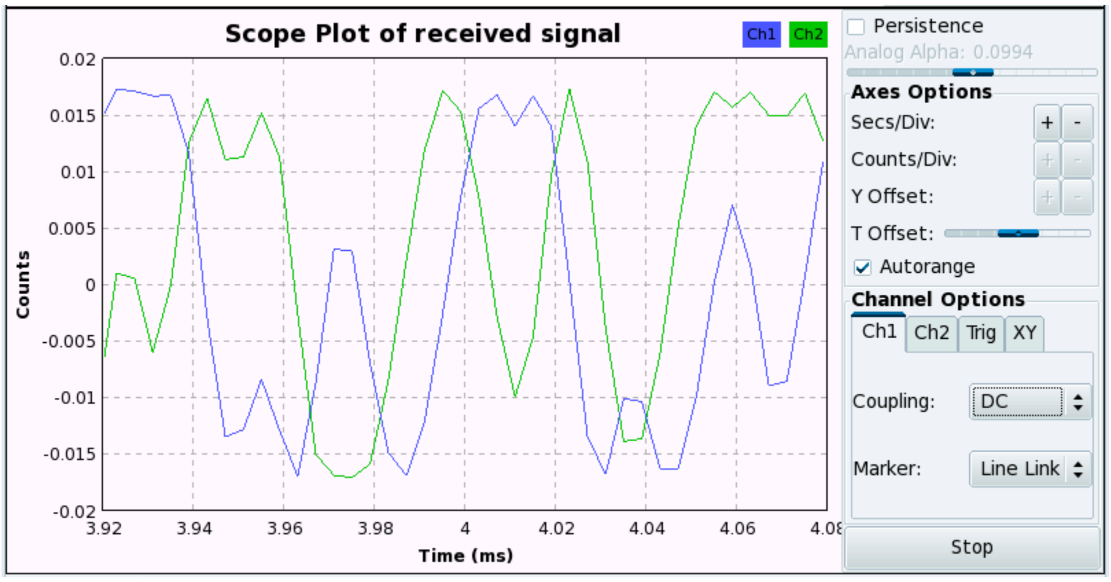


Figure 5.11.5- Oscilloscope display of received GMSK signal

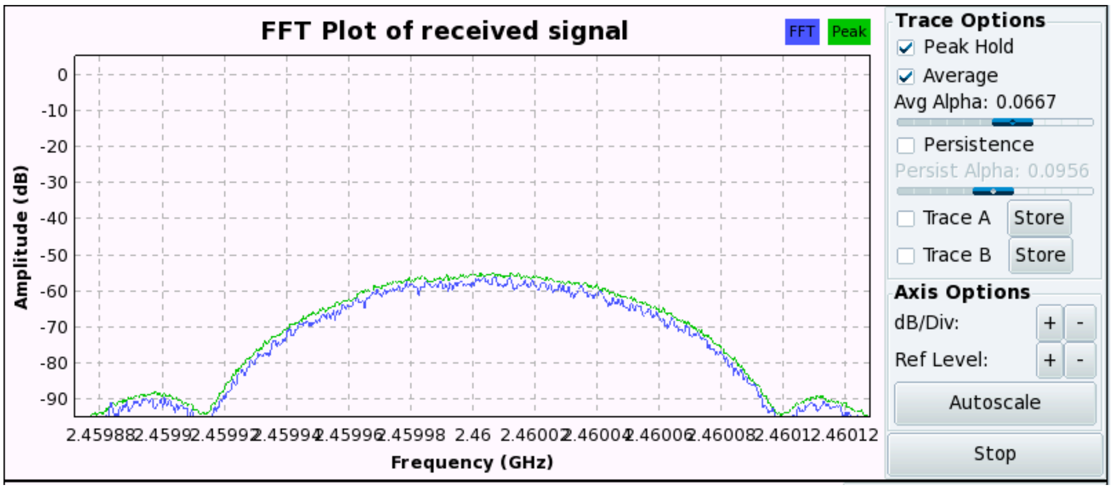


Figure 5.11.6- Spectrum display of received GMSK signal

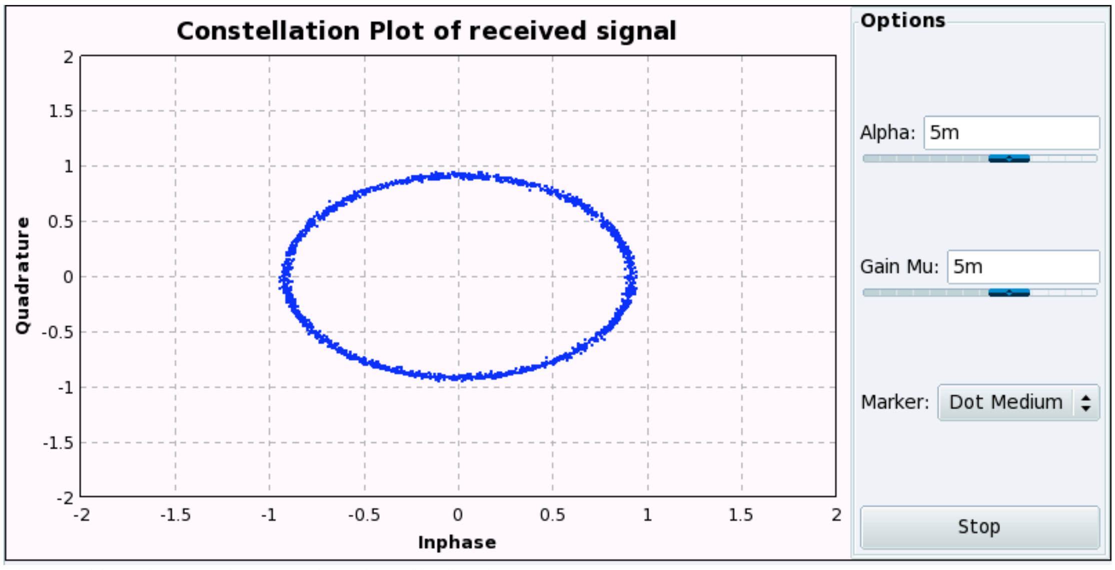


Figure 5.11.7- Constellation plot of received GMSK signal

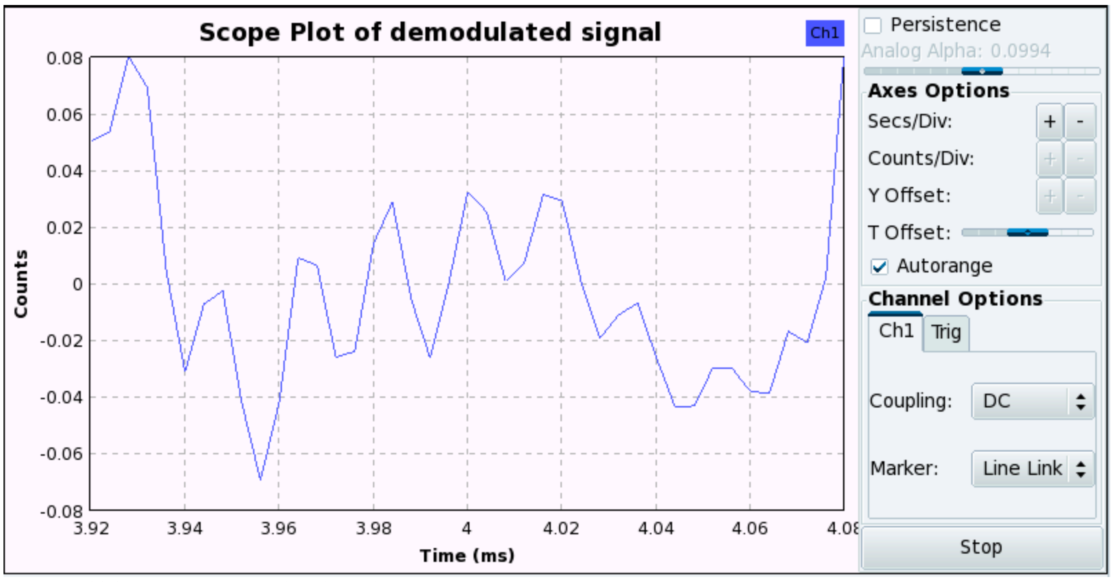


Figure 5.11.8- Demodulated voice signal in the time domain

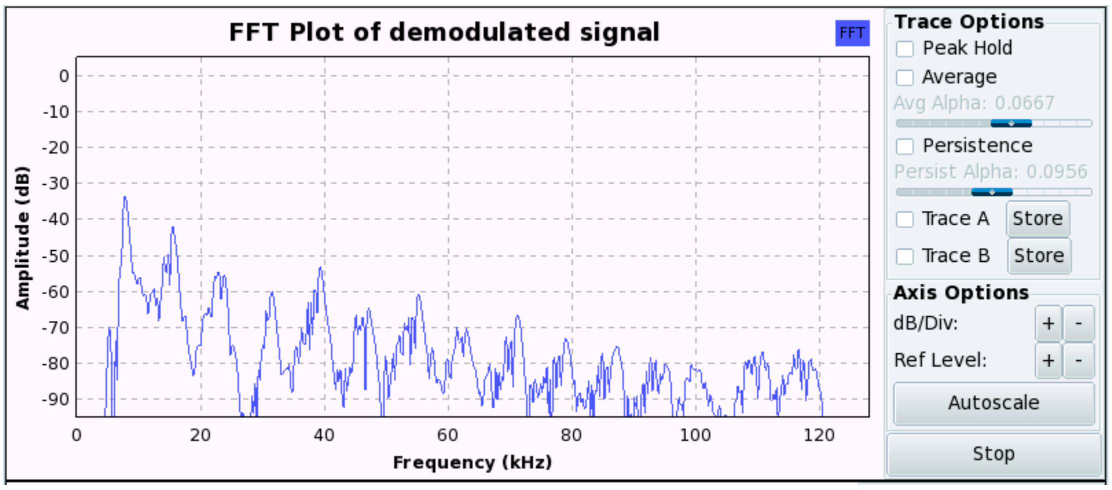


Figure 5.11.9- Demodulated voice signal in the frequency domain

Delays were introduced by the propagation of the wave between the antennas and frequency offset due to the two USRP2s not perfectly synchronized. That is why the FFT plot of the demodulated voice signal did not look exactly the same as it was before modulation.

5.12 Direct Sequence Spread Spectrum

Figures 5.12.1 to 5.12.11 show the waveforms obtained from the DSSS simulation with the repeated information bit sequence, 100011110001101. The transmitted signal took up much more bandwidth than the baseband information signal. Figures 5.12.2 and 5.12.11 show that the processing gain was about 14 dB. Although DSSS offers the best resistance to interference, its bandwidth usage is higher than the other modulation schemes described in the previous sections. Its complexity and computational load is also higher.

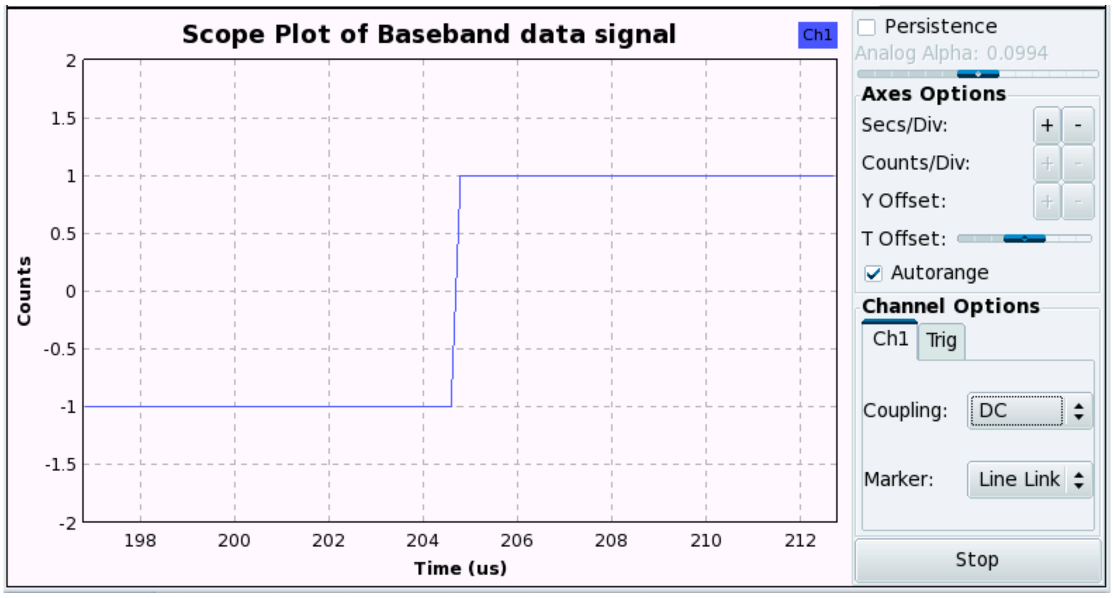


Figure 5.12.1 - Oscilloscope display of baseband signal before modulation

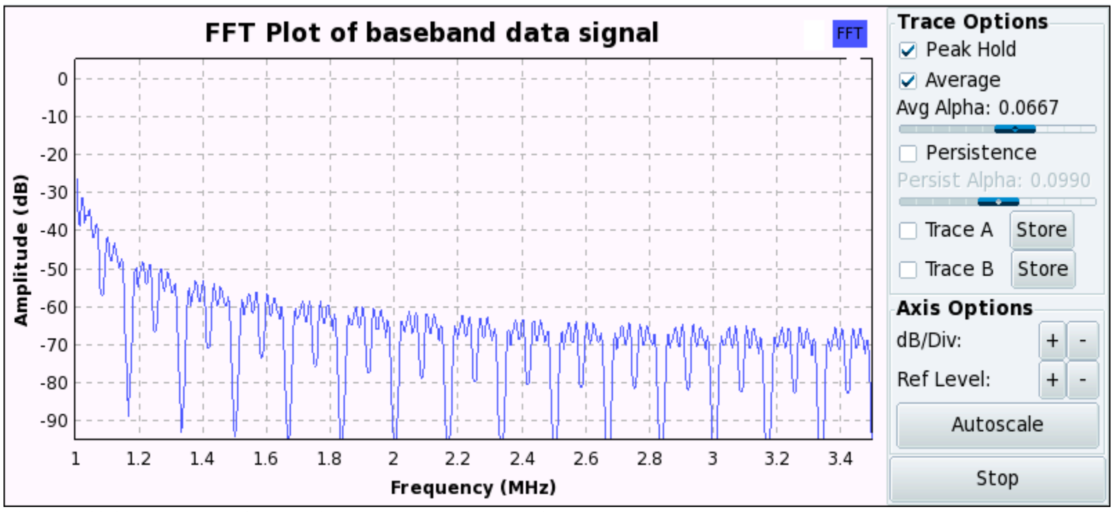


Figure 5.12.2 – FFT plot of transmitted baseband signal before modulation

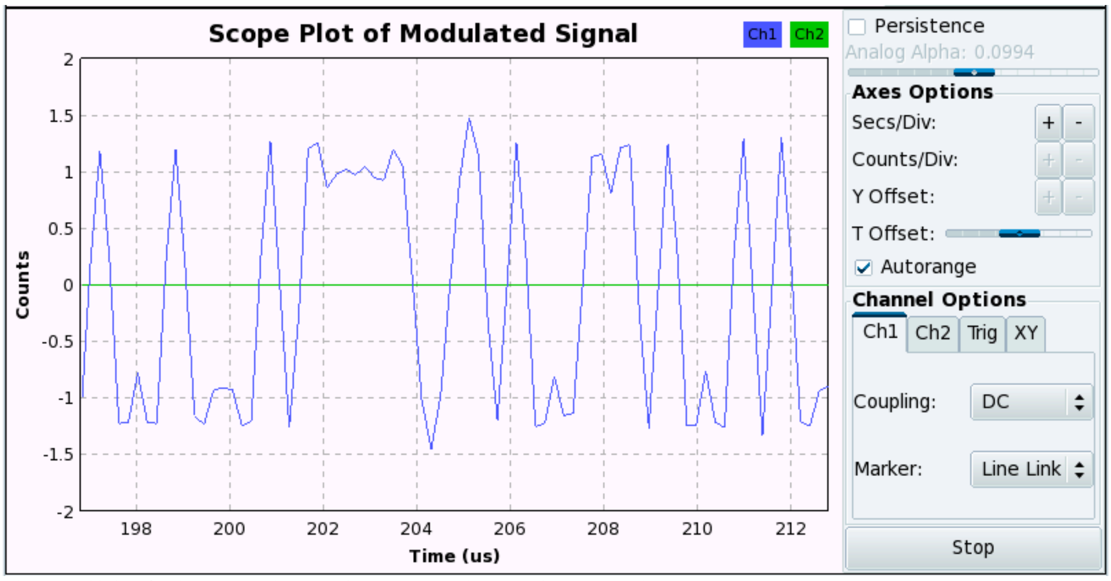
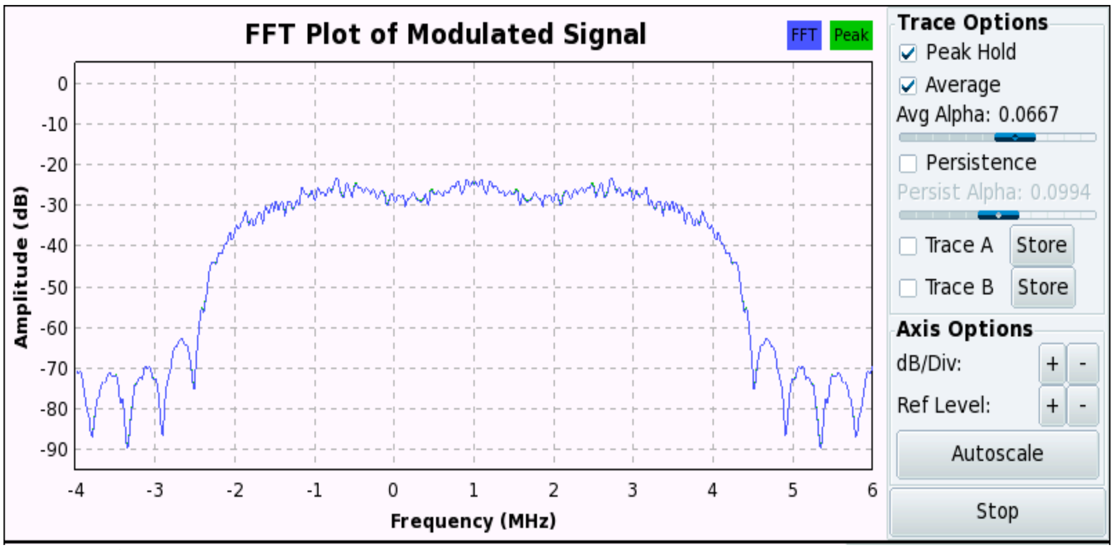
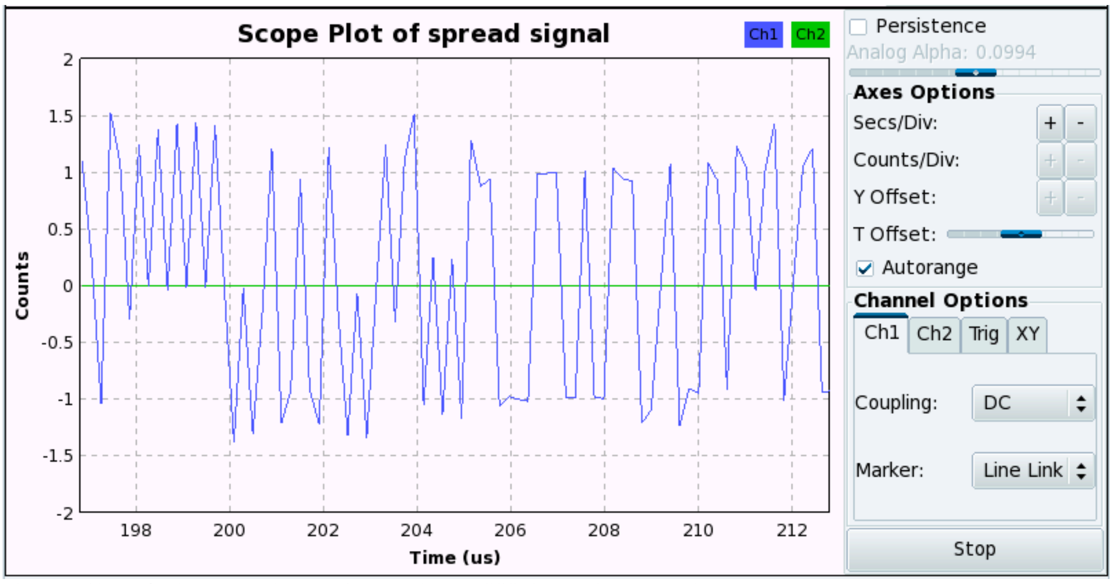


Figure 5.12.3 – Oscilloscope display of modulated signal before spreading



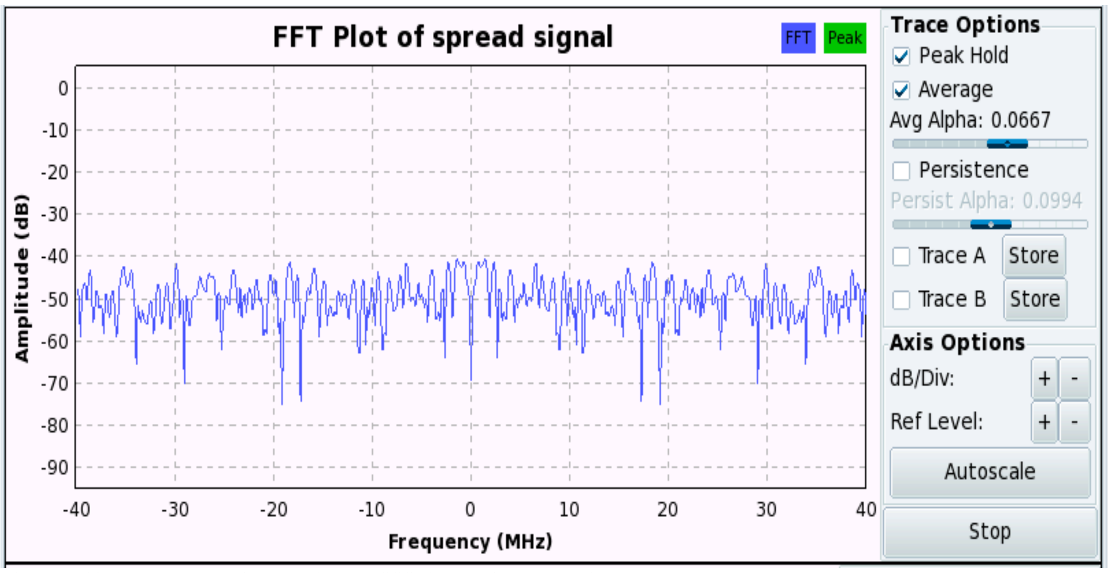
☑ Peak Hold
☑ Average
Avg Alpha: 0.0667
☐ Persistence
Persist Alpha: 0.0994
☐ Trace A
☐ Trace B

Figure 5.12.4 – FFT plot of modulated signal before spreading



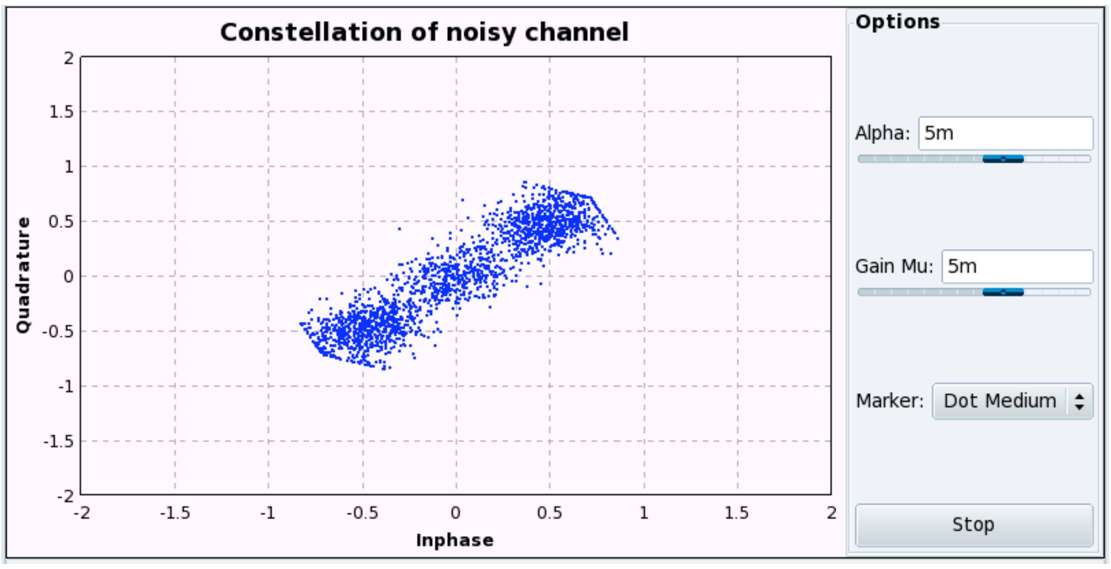
☐ Persistence
Analog Alpha: 0.0994
☑ Autorange
Coupling: DC
Marker: Line Link

Figure 5.12.5 – Oscilloscope display of transmitted DSSS signal after spreading



☑ Peak Hold
☑ Average
Avg Alpha: 0.0667
☐ Persistence
Persist Alpha: 0.0994
☐ Trace A
☐ Trace B

Figure 5.12.6 – FFT plot of transmitted DSSS signal after spreading

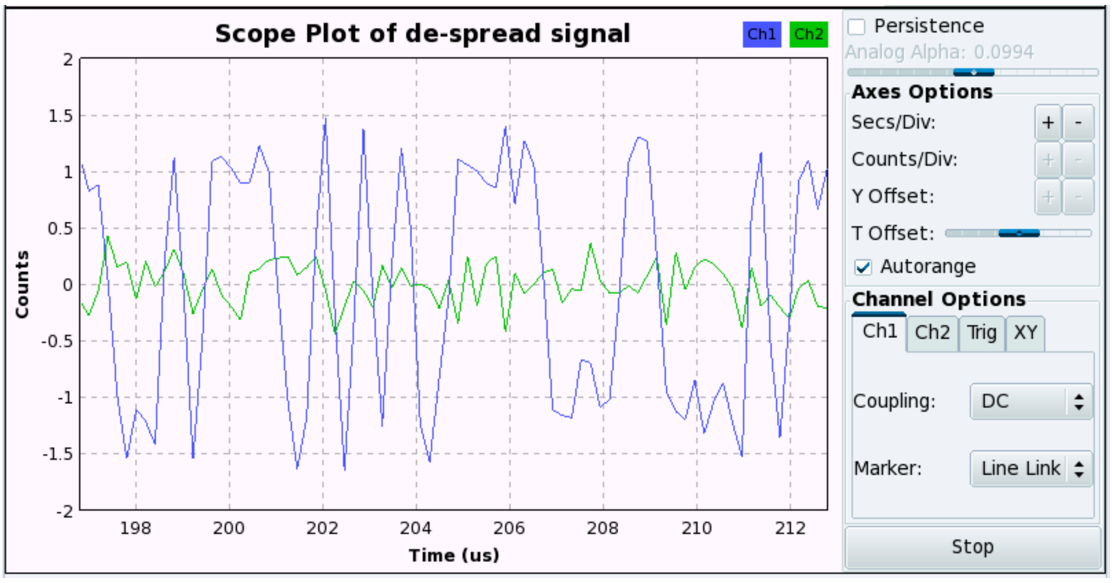


Alpha: 5m

Gain Mu: 5m

Marker: Dot Medium

Figure 5.12.7 – Constellation plot of noisy channel



☐ Persistence

Analog Alpha: 0.0994

Secs/Div:

Counts/Div:

Y Offset:

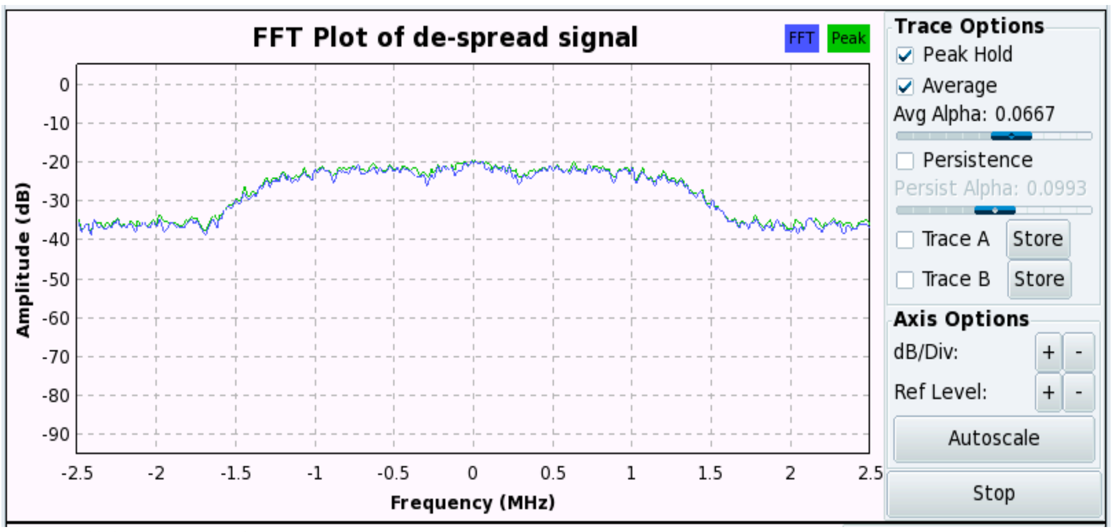
T Offset:

☑ Autorange

Coupling: DC

Marker: Line Link

Figure 5.12.8 – Oscilloscope display of de-spread signal



☑ Peak Hold

☑ Average

Avg Alpha: 0.0667

☐ Persistence

Persist Alpha: 0.0993

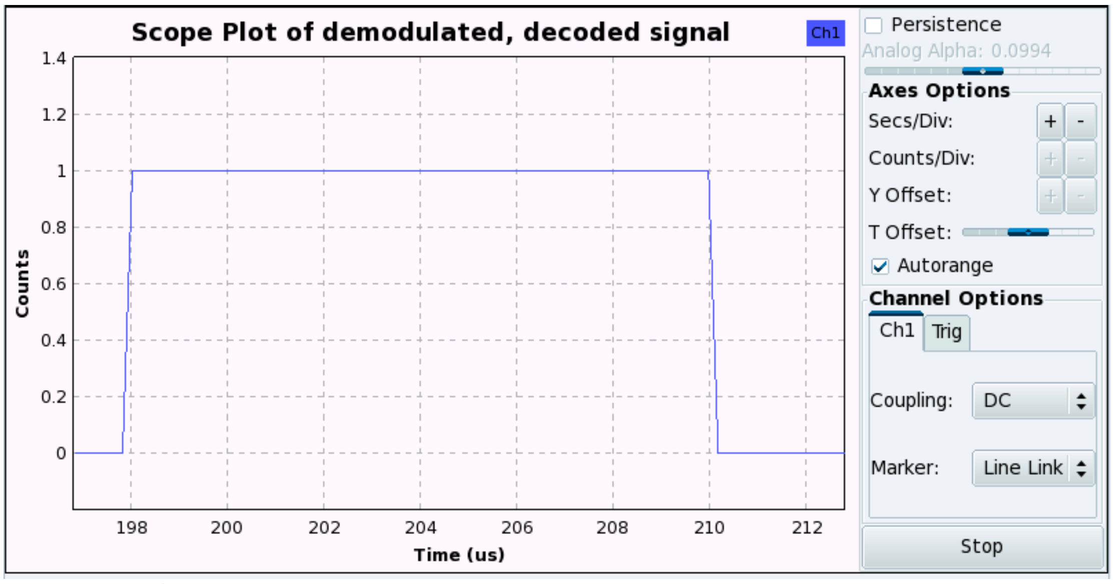
☐ Trace A

☐ Trace B

dB/Div:

Ref Level:

Figure 5.12.9 – FFT plot of de-spread signal



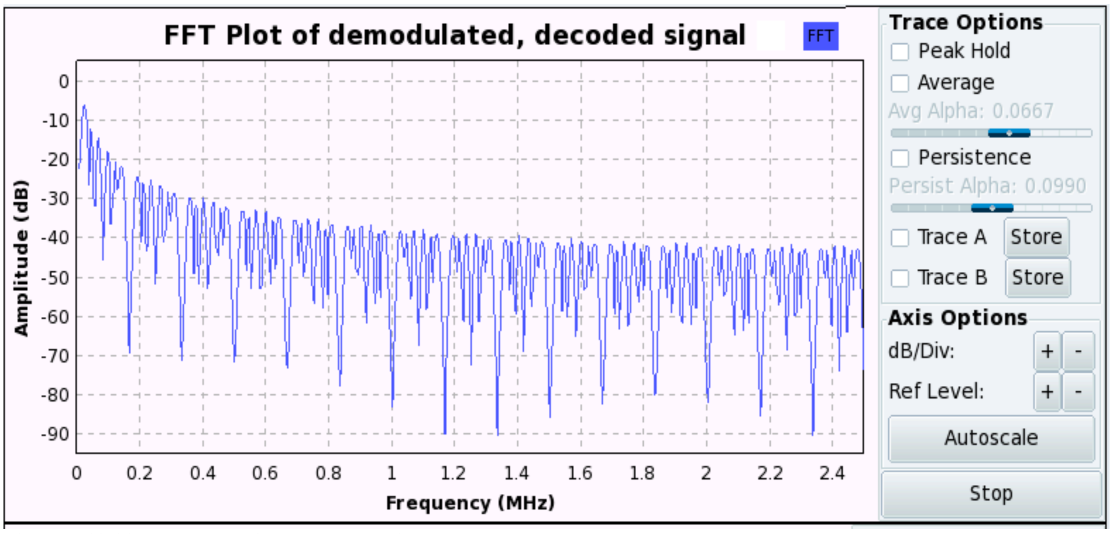
Analog Alpha: 0.0994

✓ Autorange

Coupling: DC

Marker: Line Link

Figure 5.12.10 – Oscilloscope display of demodulated signal



Avg Alpha: 0.0667

Persist Alpha: 0.0990

Figure 5.12.11 – FFT plot of demodulated signal

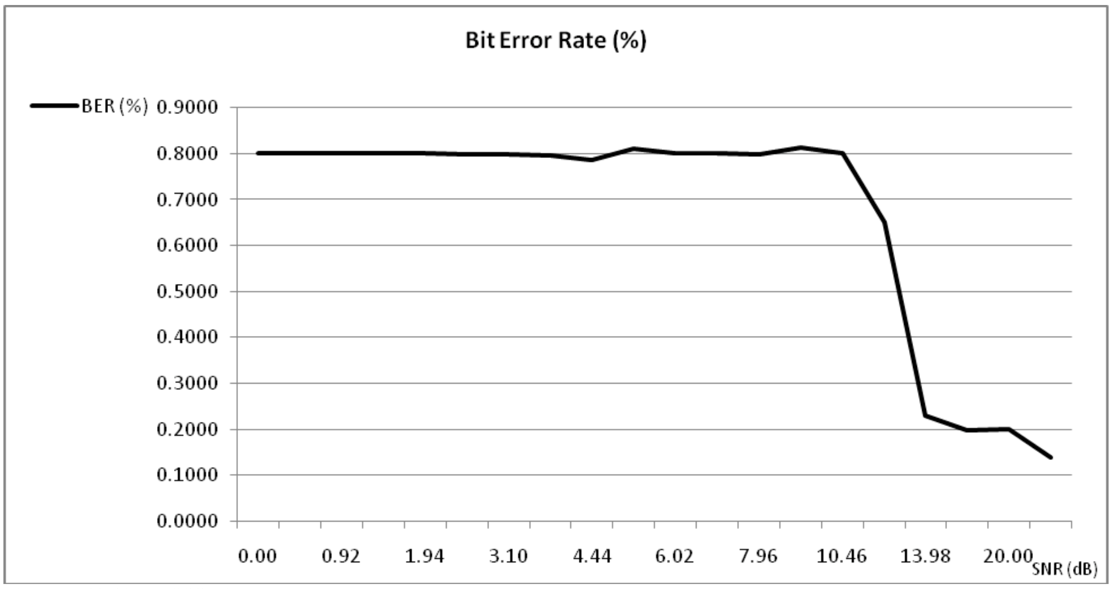


Figure 5.12.12 – DSSS Bit Error Rate curve

As the Gaussian noise of the channel model was increased the constellation points became more fuzzy and scattered as shown in figure 5.12.7. Signal attenuation due to noise made the points to move towards the centre.

The BER measurements obtained from the DSSS simulation are recorded in table 5.12.1. The BER increased sharply as the SNR decreased from 14 dB to 10 dB. Figure 5.12.12 shows the BER curve of DSSS.

Table 5.12.1-BER measurements of DSSS simulation

SNR	SNR (dB)	BER (%)
20.00	26.02	0.1383
10.00	20.00	0.1998
6.67	16.48	0.1990
5.00	13.98	0.2297
4.00	12.04	0.6497
3.33	10.46	0.7998
2.86	9.12	0.8121
2.50	7.96	0.7984
2.22	6.94	0.7998
2.00	6.02	0.7998
1.82	5.19	0.8111
1.67	4.44	0.7856
1.54	3.74	0.7966
1.43	3.10	0.7977
1.33	2.50	0.7981
1.25	1.94	0.7998
1.18	1.41	0.7998
1.11	0.92	0.7998
1.05	0.45	0.8010
1.00	0.00	0.7997

5.13 Discussion

In this thesis, various modulation schemes that are applied in wireless communication systems were designed in SDR. During experimentation, the buffering within the PYTHON flow graphs introduced latency throughout the simulated network. Frequency offsets due to the two USRP2s not perfectly synchronized also introduced delays. All the results obtained from the simulations and experimentation were consistent with theory except the DSSS BER measurements. Theory suggests that the BER should range from 0.1% to 10% for the same SNR range (Proakis, 1989). The discrepancy in BER values may have arisen from the limited window size of the ‘grc_blks2.error_rate’ function. Experimental results also showed that the value of the time-bandwidth product for the different modulation techniques offers a compromise between spectral efficiency and system performance.

The major challenges encountered during the course of the project were the late arrival of ordered equipment and the operating system security, which prevented timeous installation of the Mandriva Linux OS and GNU Radio dependency packages.

Chapter Six

6.1 Conclusions

An SDR platform, designed using the USRP2 and GNU Radio software, was implemented to support wireless communications research at the University of Hull. Design issues in implementing various modulation techniques were investigated. Various formats of speech and data signals were transmitted to demonstrate the concepts of SDR. In the author's estimation, the project was successful in achieving its intended goals. The software radio system satisfied its functional requirements. The system will be an effective practical training tool for simulation of standard radio systems, and will help students to retain the complex DSP theory learned.

6.2 Future Work

A number of areas can be pursued as future research. One area includes exploring error correction mechanisms that can be applied to the received signals in order to make them more resilient to noise and signal distortion, for example, timing recovery and carrier signal synchronization. Another area of future research is the implementation of Multiple-Input Multiple-Output (MIMO) systems to run simultaneous applications, including file transfer, image transfer, audio transfer and live speech. The work can also be extended to implementing M-ary techniques, employing multiple phases and amplitudes as different symbol states. Multi-carrier modulation schemes such as Orthogonal Frequency Division Multiplexing (OFDM) can also be investigated. The SDR system can help facilitate research in the areas of dynamic spectrum access, spectrum sensing and protection mechanisms that ensure secure and reliable operations of cognitive radios.

References

ABOULNASR, T., (1995), "One-shot cost conscious digital signal processing", *1995 International Conference on Acoustics, Speech, and Signal Processing*, ICASSP-95, vol.2, pp. 1121 - 1124.

ALLEBACH, J.P., ZOLTOWSKI, M.D., BOUMAN, C.A., (1994), “Digital signal processing with applications: a new and successful approach to undergraduate DSP education”, *1994 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP-94 vol. 5, pp. 49-52.

ANALOG DEVICES, (2006), "AD 9777 16-Bit, 160 MSPS 2x/4x/8x Interpolating Dual TxDAC+® D/A Converter", [Online], Analog Devices, Available: http://www.analog.com/static/imported-files/data_sheets/AD9777.pdf, [Accessed: 16 April 2010].

ANALOG DEVICES, (2004), "AD7912/AD7922 2-Channel, 2.35 V to 5.25 V, 1 MSPS, 10-/12-Bit ADCs", [Online], Analog Devices, Available: http://www.analog.com/static/imported-files/data_sheets/AD7912_7922.pdf, [Accessed: 16 April 2010].

ANALOG DEVICES, (2003), " AD5623R/AD5643R/AD5663R Dual 12-/14-/16-Bit nanoDAC® with 5 ppm/°C On-Chip Reference ", [Online], Analog Devices, Available: http://www.analog.com/static/imported-files/data_sheets/AD5623R_5643R_5663R.pdf, [Accessed: 16 April 2010].

ARSLAN, H., (2007), Cognitive radio, software defined radio, and adaptive wireless systems, Springer Publishers, Dordrecht, The Netherlands, ISBN 978-1-4020-5541-6.

BAEZ-LOPEZ, D., TRUEBA-MARCOS, E., RAMIREZ, J.M., JIMENEZ-LOPEZ, E., (2001), "DSP lab course experiments based on the Motorola DSP56K processor", *31st IEEE Annual Frontiers in Education Conference*, (FIE'01), vol. 1, pp. TIC-1 - TIC-4.

BARNWELL, T. P., MADISETTI, V. K., McGRATH, S. J. A., (1993), 'The Georgia Tech Digital Signal Multiprocessor', *IEEE Transactions on Signal Processing*, 41(7), pp. 2471-2487.

BOSE, V., ISMERT, M.,WELBORN, M. and GUTTAG, J., (1999), “Virtual radios,” *IEEE Journal on selected Areas in Communications*, 17(4), pp. 591 - 602.

BURNS, P. G., (2003), Software defined radio for 3G, Artech House Publishers, Norwood, Massachusetts, USA, ISBN 978-1-58-053347-8.

CETINER, B., JAFARKHANI, H., QIAN, J., YOO, H., GRAU, A., and De FLAVIIS, F., (2004), “Multifunctional reconfigurable mems integrated antennas for adaptive mimo systems,” *IEEE Communications Magazine*, 42(12), pp. 62–70.

CHEN, X., and HAN, Y., (2008), "Design and Implementation of Digital Signal Processing Soft Laboratory", *2008 International Conference on Computer Science and Software Engineering*, vol. 5, pp. 260 - 262.

CHASSAING, R., ANAKWA, W., RICHARDSON, A., (1993), "Real-time digital signal processing in education", *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP-93, vol. 1, pp. 28-31.

CHANDRAKASAN, A. P., SHENG, S., BRODERSEN, R. W., (1992), "Low-power CMOS digital design", *IEEE Journal of Solid-State Circuits*, 27(4), pp. 473 - 484.

CHANDRAKASAN, A. P., and BRODERSEN, R. W., (1995), "Minimizing power consumption in digital CMOS circuits", *Proceedings of the IEEE*, 83(4), pp. 498 - 523.

CHIANG, K. H., EVANS, B. L., HUANG, W. T., KOVAC, F., LEE , E. A., MESSERSCHMITT, D. G., REEKIE, H. J., SASTRY, S. S., (1996), "Real-time DSP for sophomores", *1996 IEEE International Conference Proceedings of the Acoustics, Speech, and Signal Processing*, ICASSP-96, vol. 2, pp. 1097-1100.

CLAUSEN, A., and SPANIAS, A., (1998), "An Internet-based computer laboratory for DSP courses", *28th Annual Frontiers in Education Conference*, (FIE-98), vol. 1, pp 206 - 210.

DeBRUNNER, V.E., DeBRUNNER, L.S., RADHAKRISHNAN, S., KHAN, A.K., (1996), "The Telecomputing laboratory: a multipurpose facility used in DSP education at the University of Oklahoma", *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP-96, vol. 2, pp. 1117-1120.

DICKERSON, J.A., (1998), "Design experiences in digital signal processing laboratories", *28th IEEE Annual Frontiers in Education Conference*, (FIE-98), vol. 3, pp. 1254.

DICKENS, M., DUNN, B., LANEMAN, N. J., (2008), "Design and Implementation of a Portable Software Radio", *IEEE Communications Magazine*, 46(8), pp. 58 – 66.

DILLINGER, M., MADANI, K., ALONISTIOTI, N., (2003), Software defined radio - Architectures, Systems and Functions, John Wiley and Sons Ltd, West Sussex, England, ISBN 978-0-47-085164-7.

ETTUS RESEARCH LLC, (2010), "USRP2 The Next Generation of Software Radio Systems", [Online], Ettus Research LLC, Available: http://www.ettus.com/downloads/ettus_ds_usrp2_v5.pdf, [Accessed:15 March 2010].

ETTUS RESEARCH LLC, (2010a), "Transceiver Daughterboards For the USRP Software Radio System", [Online], Ettus Research LLC, Available: http://www.ettus.com/downloads/ettus_ds_transceiver_dbrds_v6c.pdf, [Accessed:15 March 2010].

EYRE, J., and BIER, J., (2000), “The evolution of DSP processors”, *IEEE Signal Processing Magazine*, 17(2), pp. 43-51.

FARRELL, R., SANCHEZ, M., and CORLEY, G., (2009), “Software-Defined Radio Demonstrators: An Example and Future Trends”, *International Journal of Digital Multimedia Broadcasting*, vol. 2009, pp. 1-12.

FORNEY, G. D., (1973), "The Viterbi algorithm", *Proceedings of the IEEE*, 61(3), pp. 268–278.

FRANTZ, G., (2000), "Digital Signal Processor Trends", *IEEE Micro*, 20(6), pp. 52-59.

FRERKING, M. E., (1994), Digital signal processing in communication systems, Kluwer Academic Publishers, Massachusetts, USA, ISBN 978-0-44-201616-6.

FUCHIWAKI, Y., USUKI, N., ARAI, T., MURAHARA, Y., (2000), "The DSP experiments for under graduate students", *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP’00, vol. 6, pp. 3526 - 3529.

GADHIOK, M., HARDY, R., MURPHY, P., FRANTZ, J. P., CHOI, H., CAVALLARO, J. R., (2005), "An FPGA-Based Daughtercard for TI's C6000 family of DSKs", *2005 IEEE International Conference on Microelectronic Systems Education*, (MSE'05), pp. 85-86.

GAN, W. -S., CHONG, Y. -K., GONG, W., TAN, W. -T., (2000), “Rapid Prototyping System for Teaching Real-Time Digital Signal Processing,” *IEEE Transactions on Education*, 43(1), pp. 19–24.

GNU RADIO, (2010), GNU Radio Wiki page, [Online], GNU Radio, Available: http://www.gnuradio.org, [Accessed:30 March 2010].

GONZALEZ, C. R. A., DIETRICH, C. B., SAYED, S., VOLOS, H. I., GAEDDERT, J. D., ROBERT, P. M., REED, J. H., KRAGH, F. E., (2009), "Open-source SCA-based core framework and rapid development tools enable software-defined radio education and research", *IEEE Communications Magazine*, 47(10), pp 48 - 55.

GUNN, J. E., BARRON, K. S., RUCZCZYK, W., (1999), "A Low-Power DSP Core-Based Software Radio Architecture", *IEEE Journal on Selected Areas in Communications*, 17(4), pp. 574 - 590.

HAYKIN, S., (2001), Communications Systems, 4th Edition, John Wiley & Sons, New Jersey, ISBN 0-471-17869-1.

HEDDE, D., HORREIN, P. -H., PETROT, F., ROLLAND, R., ROUSSEAU, F., (2009), "A MPSoC Prototyping Platform for Flexible Radio Applications", *12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, DSD'09, pp. 559 - 566.

HEIDEMAN, M. T., JOHNSON, D. H., BURRUS, C. S., (1985), "Gauss and the history of the fast Fourier transform", *Archive for History of Exact Sciences*, 34(3), pp. 265-277.

HUEBER, G., ZOU, Y., DUFRENE, K., STUHLBERGER, R., VALKAMA, M., (2009), "Smart Front-End Signal Processing for Advanced Wireless Receivers", *IEEE Journal of Selected Topics in Signal Processing*, 3(3), pp. 472 - 487.

HWANG, J.-K., (2003), "Innovative communication design lab based on PC sound card and Matlab: a software-defined-radio OFDM modem example", *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP'03, vol. 3, pp. III-761 - III-764.

JACKSON, J., BARNWELL, T., WILLIAMS, D., HAYES, M., III, ANDERSON, D., SCHAFER, R., (2001), "DSP for practicing engineers: an online course for continuing DSP education", *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP '01, vol. 5, pp.2721-2724.

JEYALAKSHMI, V., and SANKARANARAYANAN, K., (2010), "Challenges in Technology and Reconfiguration of SDR - A Survey", *IETE Technical Review*, 25(1), pp. 19 - 28.

JOAQUIM, M.B., PEREIRA, J.C., de OLIVEIRA, V.A., (1998), "Course on DSP design using MATLAB", *28th IEEE Annual Frontiers in Education*, (FIE-98), vol. 2, pp. 685-690.

JONES, D. L., (2001), "Designing effective DSP laboratory courses", *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP'01, vol. 5, pp.2701-2704.

KANG, J., OLSON, O., FELZER, A., CHANDRA, R., OLDAK, S., (2007), "Simulink Based Real-Time Laboratory Course Development", *2007 IEEE International Conference on Microelectronic Systems Education*, (MSE'07), 0-7695-2849-X, pp. 15-16.

KATZ, S., and FLYNN, J., (2009), "Using software defined radio (SDR) to demonstrate concepts in communications and signal processing courses", *39th IEEE Annual Frontiers in Education Conference*, (FIE'09), pp. W2B-1-W2B-6.

KLINE, R., (1990), "An Overview of Twenty-Five Years of Electrical and Electronics Engineering in the Proceedings of the IEEE, 1963-1987", *Proceedings of the IEEE*, 78(3), pp. 469-485.

KUO, S. M., and MILLER, G. D., (1995), "An innovative course on real-time digital signal processing applications", *1995 Conference Record of the Twenty-Ninth Asilomar Conference on Signals, Systems and Computers*, ASILOMAR '95, vol. 1, pp. 88-92.

KURUGOLLU, F., PALAZ, H., GUMUKAYA, H., HARMANCI, E., ÖRENCIK, B., (1995), "Advanced educational parallel DSP system based on TMS320C25 processors", *Microprocessors and Microsystems*, 19(3), pp. 147-156.

LACKEY, R. I., and UPMAL, D. W., (1995), "Speakeasy: the military software radio", *IEEE Communications Magazine*, 33(5), pp. 56 - 61.

LEE, E. A., (1988), "Programmable DSP Architectures: Part I", *IEEE ASSP Magazine*, 5(4), pp. 4-19.

LEE, E. A., (1989), "Programmable DSP Architectures: Part II", *IEEE ASSP Magazine*, 6(1), pp. 4-14.

LINEAR TECHNOLOGY CORPORATION, (2006), "LTC2285 Dual 14-Bit, 100 Msps Low Power 3V ADC", [Online], Linear Technology Corporation, Available: http://www.linear.com/pc/productDetail.jsp?navId=H0,C1,C1155,C1001,C1150,P15833, [Accessed: 17 April 2010].

LIU, K. J. R., WU, A. –Y., RAGHUPATHY, A., CHEN, J., (1998), "Algorithm-Based Low-Power and High-Performance Multimedia Signal Processing", *Proceedings of the IEEE*, 86(6), pp. 1155-1202.

LIU, K. R., HSIEH, S. -F., YAO, K., (1992), "Systolic block Householder transformation for RLS algorithm with two-level pipelined implementation", *IEEE Transactions on Signal Processing*, 40(4), pp. 946 - 958.

LU, G., and LU, P., (2008), "A Software Defined Radio Receiver Architecture Based on Cluster Computing", *Seventh International Conference on Grid and Cooperative Computing*, GCC'08, pp. 612 - 619.

L-COM GLOBAL CONNECTIVITY, (2010), "HyperLink Wireless 2.4 GHz to 5.8 GHz 3dBi TNC-Male Tri-band Rubber Duck Antenna", [Online], L-Com Global Connectivity, Available: http://www.ettus.com/downloads/VERT2450.pdf, [Accessed:15 March 2010].

MADISETTI, V.K., McCLELLAN, J.H., BARNWELL, T.P., III, (1995), "DSP design education at Georgia Tech", *1995 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP-95, vol. 5, pp. 2869 - 2872.

McCLELLAN, J. H., SCHAFER, R. W., SCHODORF, J. B., YODER, M. A., (1996), "Multi-media and World Wide Web resources for teaching DSP", *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP-96, vol.2, pp. 1101-1104.

MENG, T. H. -Y., BRODERSEN, R., W., MESSERSCHMITT, D. G., (1991), "Asynchronous design for programmable digital signal processors", *IEEE Transactions on Signal Processing*, 39(4), pp. 939 - 952.

MELTON, D. E., FINELLI, C. J., RUST, L. M., (1999), "A Digital Signal Processing Laboratory with Style", *29th ASEE/IEEE Annual Frontiers in Education Conference*, (FIE-99), vol. 2, pp. 12b6-14 - 12b6-19.

MINDEN, G. J., EVANS, J. B., SEARL, L. S., DePARDO, D., RAJBANSHI, R., GUFFEY, J., CHEN, Q., NEWMAN, T. R., PETTY, V. R., WEIDLING, F., PECK, M., CORDILL, B., DATLA, D., BARKER, B., AGAH, A., (2007), "COGNITIVE RADIOS FOR DYNAMIC SPECTRUM ACCESS - An Agile Radio for Wireless Innovation", *IEEE Communications Magazine*, 45(5), pp. 113-121.

MITOLA, J., (1995), “The software radio architecture”, *IEEE Communications Magazine*, 33(5), pp. 26–38.

MOBASSERI, B.G., (1997), "Development of an experimental component for a senior communication systems course: a web-assisted approach", *27th IEEE Annual Frontiers in Education Conference*, (FIE-97), vol. 2, pp. 1066 - 1069.

MORROW, M. G., WELCH, T. B., WRIGHT, C. H. G., YORK, G. W. P., (2001), "Demonstration platform for real-time beamforming", *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP '01, vol. 5, pp. 2693 - 2696.

MOUSAVINEZHAD, S.H., and ABDEL-QADER, I. M., (2001), "Digital signal processing in theory and practice", *31st IEEE Annual Frontiers in Education Conference*, (FIE'01), vol. 1, pp. T2C-13-T2C-16.

NAGURNEY, L.S., (2009), "Software defined radio in the electrical and computer engineering curriculum", *39th ASEE/IEEE Annual Frontiers in Education Conference*, (FIE '09), pp. W2E-1-W2E-6.

NAKAJIMA, N., KOHNO, R., KUBOTA, S., (2001), "Research and Developments of Software-Defined Radio Technologies in Japan", *IEEE Communications Magazine*, 39(8), pp. 146 - 155.

NATIONAL SEMICONDUCTOR CORPORATION, (2004), "DP83865 Gig PHYTER® V 10/100/1000 Ethernet Physical Layer", [Online], National Semiconductor Corporation, Available: http://www.national.com/ds/DP/DP83865.pdf, [Accessed: 12 April 2010]

OH, S.-H., ABERLE, J. T., ANANTHARAMAN, S., ARAI, K., CHONG, H. L., and KOAY, S. C., (2005), “Electronically tunable antenna pair and novel rf front-end architecture for software-defined radios,” *EURASIP Journal on Advances in Signal Processing*, vol. 16, pp. 2701 – 2707.

OXTOBY, A. J. A., (1995), “Hardware and software tools and laboratory experiments for an undergraduate EET course in digital signal processing”, *25th ASEE/IEEE* Annual *Frontiers in Education Conference,* (FIE-95), vol. 2, pp. 4c2.14-4c2.19.

PARHI, K. K., (1994), “VLSI Digital Signal Processing Education”, *1994 Conference Record of the Twenty-Eigth Asilomar Conference on Signals, Systems and Computers*, ASILOMAR '94, vol. 2, pp. 1303 - 1308.

PEEBLES, P. Z., (1987), Digital Communications Systems, Prentice Hall, Englewood Cliffs, New Jersey, USA, ISBN 0-13-211970-6.

PEREZ-NEIRA, A., MESTRE, X., FONOLLOSA, J., (2001), “Smart antennas in software radio base stations”, *IEEE Communications Magazine*, 39(2), pp. 166–173.

PIERRE, J. W., KUBICHEK, R. F., HAMANN, J. C., (1999), "Reinforcing the understanding of signal processing concepts using audio exercises*", 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP-99, vol. 6, pp. 3577-3580.

PROAKIS, J. D., (1989), Digital Communications, 2nd Edition, McGraw Hill, Singapore, ISBN 0-07-100269-3.

PROAKIS, J. G., and MANOLAKIS, D. G., (1996), Digital Signal Processing: Principles, Algorithms and Applications, 3rd Edition, Prentice Hall, Upper Saddle River, New Jersey, USA, ISBN 978-0-13-394289-7.

RABAEY, J. M., GASS, W., BRODERSEN, R., NISHITANI, T., TSUHAN, C., (1998), "VLSI Design and Implementation Fuels the Signal-Processing Revolution", *IEEE Signal Processing Magazine*, 15(1), pp. 22–37.

RAMACHER, U., (2007), "Software-Defined Radio Prospects for Multi-standard Mobile Phones", *Computer*, 40(10), pp. 62-69.

REED, J. H., (2002), Software Radio: a modern approach to radio engineering, Prentice Hall, New Jersey, USA, ISBN 978-0-13-081158-5.

SMITH, S. W., (1997), The Scientist and Engineer's Guide to Digital Signal Processing, California Technical Publishing, California, USA, ISBN 978-0-96-601763-2.

SPANIAS, A., URBAN, S., CONSTANTINOU, A., TAMPI, M., CLAUSEN, A., ZHANG, X., FOUTZ, J., STYLIANOU, G., (2000), "Development and evaluation of a Web-based signal and speech processing laboratory for distance learning", *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP'00, vol. 6, pp. 3534-3537.

SPANIAS, A., and BIZUNEH, F., (2001), "Development of new functions and scripting capabilities in JavaA-DSP for easy creation and seamless integration of animated DSP simulations in Web courses", *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing* , ICASSP'01, vol. 5, pp. 2717-2720.

STEWART, R.W., (1993), "Practical DSP for scientists", *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP-93, vol. 1, pp. 32 - 35.

TAYLOR, F., MELLOTT, J.D., LEWIS, M., (1996), "Spectra- a hands-on DSP learning experience", 1996 *IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP-96, vol. 2, pp. 1125-1128.

TESSIER, R., and BURLESON, W., (2001), "Reconfigurable Computing for Digital Signal Processing: A Survey", *Journal of VLSI Signal Processing*, 28(1-2), 7–27.

TRUSSELL, H. J., and RAJALA, S. A., (1995), "A DSP laboratory designed for teaching fundamental concepts", *1995 International Conference on Acoustics, Speech, and Signal Processing*, ICASSP-95, vol. 2, pp. 1117-1120.

VERBAUWHEDE, I., TOURIGUIAN, M., GUPTA, K., MUWAFI, J., YICK, K., FETTWEIS, G., (1996), "A low power DSP engine for wireless communications", *Workshop on VLSI Signal Processing*, IX, pp. 471 - 480.

VERBAUWHEDE, I., and NICOL, C., (2000), "Low Power DSP's for Wireless Communications", *Proceedings of the 2000 international symposium on Low power electronics and design*, ISLPED '00, pp. 303-310.

VITERBI, A. J., (1967), "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm", *IEEE Transactions on Information Theory*, 13(2), pp. 260–269.

WANG, C. -L., (1994), "Bit-serial VLSI implementation of delayed LMS adaptive FIR filters", *IEEE Transactions on Signal Processing*, 42(8), pp. 2169 - 2175.

WELCH, T. B., WRIGHT, C. H. G., MORROW, M. G., (2007), "Teaching Rate Conversion using Hardware-Based DSP", *2007 IEEE International Conference on Acoustics, Speech, and Signal Processing*, ICASSP'07, vol. 3, pp. III-717 - III-720.

WRIGHT, C. H. G., WELCH, T. B., MORROW, M. G., (2003), "An inexpensive method to teach hands-on digital communications", *33rd IEEE Annual Frontiers in Education Conference*, (FIE'03), vol. 2, pp. F2E-20 - F2E-25.

XILINX, (2009), "Spartan-3 FPGA Family Data Sheet", [Online], Xilinx, Available: http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf, [Accessed: 29 March 2010].

XU, X., BOSISIO, R. G., WU, K., (2006), "Analysis and implementation of six-port software-defined radio receiver platform ", *IEEE Transactions on Microwave Theory and Techniques*, 54(7), pp. 2937 – 2943.

YASIN, M., KARAM, L. J., SPANIAS, A., (2003), "On-line laboratories for image and two-dimensional signal processing", *33rd IEEE Annual Frontiers in Education Conference*, (FIE'03), vol. 1, pp. T3E-19 - T3E-22.

YOUNGWOOK, K, DUMAN, T. M., SPANIAS, A., (2003), "On-line laboratory for communication systems using J-DSP", *33rd IEEE Annual Frontiers in Education Conference*, (FIE'03), vol. 1, pp. T3E-13 - T3E-18.

Appendix A

Linux Mandriva and GNU Radio Installation Procedure

The mandriva-seed.exe application was used to write the mandriva-linux-free-2010.0-i586 installation image onto a USB storage device. The mandriva-linux-free-2010.0-i586 image was downloaded from the mirror site 'http://www.mandriva.com/download'. During installation from the USB, two ordinary user accounts, 'user' and 'guest', were created without 'super user' rights. Once Mandriva installation was complete, the internet proxy settings were configured by clicking on the KDE Application Launcher menu located in the bottom left of the computer screen, and opening the subsequent submenus to access the 'Proxy settings' window:

> Application Launcher menu → Tools → System Tools → Configure your computer → Network and Internet → Proxy

The Proxy server was configured as 'http://slb-webcache.hull.ac.uk' and the port was configured as 3128. The media sources for installing and updating the GNU Radio dependency packages were then configured in the 'Configure media sources for install and update' window:

> Application Launcher menu → Tools → System Tools → Configure your computer → Software Management → Configure media sources for install and update

The Mandriva package installer, RPMdrake, was then used to install all of GNU Radio's package dependencies except Microblaze v4.1.1. RPMDrake was accessed through:

> Application Launcher menu →Install and Remove Software

When installation of the dependencies was complete, the following commands were executed at the command line interface as the 'root' user to remove all orphaned packages and to update the system:

```
[root@localhost user]# urpme --auto-orphans
[root@localhost user]# urpmi --auto-update
```

The Microblaze tarball, mb-gcc-4.1.1.gr2.i386.tar.gz, was downloaded from the GNU Radio website, http://gnuradio.org, and installed through the command line prompt using the following commands. Microblaze was required for the FPGA's firmware and had to be installed for successful installation of the GNU Radio component, usrp2-firmware.

```
[root@localhost user]# cd /home/user/Downloads
[root@localhost Downloads]# ls

mb-gcc-4.1.1.gr2.i386.tar.gz

[root@localhost Downloads]# sudo bash

gpg-agent[24979]: directory `/root/.gnupg' created
gpg-agent[24979]: directory `/root/.gnupg/private-keys-v1.d' created
gpg-agent[24980]: gpg-agent (GnuPG) 2.0.13 started
[root@localhost Downloads]# cd /opt
[root@localhost opt]# tar xzvf /home/user/Downloads/mb-gcc-4.1.1.gr2.i386.tar.gz

microblaze/
microblaze/info/
microblaze/info/bfd.info
microblaze/info/dir
microblaze/info/gccinstall.info
.
.
.
microblaze/libexec/gcc/microblaze-xilinx-elf/4.1.1/install-tools/fixincl
microblaze/libexec/gcc/microblaze-xilinx-elf/4.1.1/install-tools/fixinc.sh
microblaze/libexec/gcc/microblaze-xilinx-elf/4.1.1/collect2
microblaze/libexec/gcc/microblaze-xilinx-elf/4.1.1/cc1
[root@localhost opt]# ls /opt
```

After Microblaze installation, the following environment variables were configured in the /root/.bashrc and /home/user/.bash_profile files using the VI text editor:

1. LD_LIBRARY_PATH=/usr/local/lib:/lib:/usr:/usr/include:/usr/local/include:/usr/lib:/usr/sbin:/usr/lib/pkgconfig
   export LD_LIBRARY_PATH

2. CC=gcc-4.4.1
   export CC

3. PYTHONPATH=/usr/local/lib/python2.6/site-packages:/usr/lib/python2.6/sitepackages:/usr/lib/ooo/basis-link/program:/usr/lib/ooo/basis-link/ program:/usr/lib/ooo/basis-link/program
   export PYTHONPATH

4. PATH=/usr/share/colorgcc:/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/usr/local/sbin:/usr/lib/qt4/bin:/usr:/usr/include/cppunit:/usr/src/debug/cppunit1.12.1/include/cppunit:/usr/src/debug/cppunit-1.12.1/src/cppunit:/usr/src/debug/fftw-2.1.5/single/fftw:/usr/src/debug/fftw2.1.5/double/fftw:/usr/include:/usr/src:/usr/lib:/opt/microblaze/bin:/usr/libexec/sdcc:.
   export PATH

5. PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
   export PKG_CONFIG_PATH

The machine was rebooted before the following commands were used to download and configure the GNU Radio software components.

```
[root@localhost opt]# cd /home/user
[root@localhost user]# git clone http://gnuradio.org/git/gnuradio.git
[root@localhost user]# pwd
/home/user
[root@localhost user]# cd gnuradio
[root@localhost gnuradio (master)]# ls
[root@localhost gnuradio (master)]# ./bootstrap
[root@localhost gnuradio (master)]# ./configure --with-qwt-incdir=/usr/include --with-
qwtplot3d-incdir=/usr/include
```

All GNU Radio components were configured successfully. The following commands were used to compile and install the components.

```
[root@localhost gnuradio-3.2.2]# make
[root@localhost gnuradio-3.2.2]# make check
[root@localhost gnuradio-3.2.2]# make install
```

To allow ordinary users other than ‘root’ to use the USRP2 the following commands were executed.

```
[root@localhost user]# cd /home/user/gnuradio
[root@localhost gnuradio (master)]# sudo groupadd usrp
[root@localhost gnuradio (master)]# sudo usermod -G usrp user
 [root@localhost gnuradio (master)]#
[root@localhost gnuradio (master)]# echo 'ACTION=="add", BUS=="eth0",
SYSFS{idVendor}=="fffe", SYSFS{idProduct}=="0002", GROUP:="usrp",
MODE:="0660"' > tmpfile
[root@localhost gnuradio (master)]# ls -ltr tmpfile
-rw-r--r-- 1 root root 106 2010-07-02 13:53 tmpfile
[root@localhost gnuradio (master)]# more tmpfile
```

```
ACTION=="add", BUS=="eth0", SYSFS{idVendor}=="fffe", SYSFS{idProduct}=="0002",
GROUP:="usrp", MODE:="0660"
[root@localhost gnuradio (master)]# sudo chown root.root tmpfile
[root@localhost gnuradio (master)]# sudo mv tmpfile /etc/udev/rules.d/10-usrp.rules
[root@localhost gnuradio (master)]# sudo chmod ugo+s /usr/local/bin/usrp2_socket_opener
[root@localhost gnuradio (master)]# ls -ltr /usr/local/bin/usrp2_socket_opener
-rwsr-sr-x 1 root root 12319 2010-06-30 13:20 /usr/local/bin/usrp2_socket_opener*
```

Using the VI editor, the following line was added to the file /etc/security/limits.conf to allow real-time scheduling for non-root users.

```
@usrp  - rtprio 50
```

GNU Radio Companion was enabled through execution of the following command as the ordinary user, 'user':

```
[user@localhost user]# /usr/local/libexec/gnuradio/grc_setup_freedesktop install
Begin freedesktop install...
Install icon: 32x32
Install icon: 48x48
Install icon: 64x64
Install icon: 128x128
Install icon: 256x256
Install mime type
Install menu items
Done!
```

The equipment was set up as shown in Figure 4.1 before the following commands were used to locate the USRP2s.

```
[root@localhost user]# sudo find_usrps -e eth0
[root@localhost user]# sudo find_usrps -e eth1
```

The GNU radio executable files were installed to /usr/local/bin, the libraries were installed to /usr/local/lib and the configuration files were saved to /usr/local/etc. GNU Radio Companion was launched by typing 'grc' at the command prompt. GRC could also be accessed through the Mandriva KDE application launcher menu:

Application Launcher menu →Development →GRC.

Appendix B

GNU Radio Dependency packages

Dependency Package	Version	Description
gcc	4.4.1	GNU compiler Collection
sdcc	2.8.0	Small Device C compiler
autoconf	2.6.4	Tool for producing configure scripts for C/C++
automake	1.11	Programming tool to produce portable ‘makefiles’
libtool	2.2.6	Portable compiled libraries creator
swig	1.3.40	Interface compiler to connect C/C++ programs
flex	4.1.0	Web applications builder
bison	2.4.3	General-purpose parser generator
numarray	1.0.3	Array manipulator and numerical data processor
numpy	1.5.0	Package used for scientific computing with Python
cppunit	1.12.1	C++ port for JUnit framework
libcppunit-devel	1.12.1	Development files for cppunit
fftw	3.2.2	Subroutine library used to compute the Discrete Fourier Transform
libpython-devel	3.1.1	The libraries and header files needed for Python development
libboost1-devel	1.39.0	Boost development libraries and headers
wxPythonGTK	2.8.9.2	GUI toolkit for Python
guile	1.8.7	Programming interpreter
gsl	1.7.106	GNU Scientific library
python-cheetah	2.4.2.1	Code generation tool
qwt	5.2.1	2 dimensional plot widget
portaudio	18.1	Audio Input / Output library
jack	1.9.5	Audio Input / Output connector
lxml	2.2.7	Binder for using xml and html in Python
grc	2.1.0	Governance Risk Compliance tool
pygtk	2.4	Tool for controlling PYTHON graphical elements
gtk+1.2	1.2	GUL create toolkit
python-opengl	3.0.1	Binder to link Python and OpenGL
gnuplot	4.4.0	Scientific plotting package
libosip	2.2.0	Library for Session Initiation Protocol
libortp	0.15.0	Library for Real-time Transport Protocol
distcache	1.5.1	Tool for supporting single network-based sessions
id3lib	3.8.3	Control library for ID3v2 tags
libusb	0.1.12	Interface to USB ports
qt-devel	4.4.3	Header files for QT GUI toolkit
pthread	1.1.1	Application Programming Interface for multithreaded applications
libqt4-devel	4.7.0	Libraries for QT GUI toolkit
libalsa	1.0.23	Library to access audio devices
alsa-devel	1.0.23	Header files for accessing audio devices
libxslt	1.1.26	Libraries to support XSLT files
gsl-devel	1.7.106	GNU Scientific library header files and libraries
qt4-devel	4.7.0	Development files for QT GUI toolkit
python-scipy	0.8.0	The SciPy library
python-matplotlib	0.99.3	Library to export graphical plots
libcamd	2.2.0	Library to compute sparse matrices
libcolamd	2.2.1	Library to compute sparse matrices
libcxsparse	2.2.1	Library to compute sparse matrices

Dependency Package	**Version**	**Description**
hdf5-devel	1.8.5	Hierarchical Data Format (HDF5) development files
lapack-devel	3.0	Numerical linear algebra library
docbook	1.75.2	Application required for GNU Radio documentation
texlive-texmf	2007-36	Application required for GNU Radio documentation
octave	3.2.4	Interface to compute numerical linear and nonlinear problems
readline-devel	6.1	functions to allow editing of typed command lines
subversion	1.6	Binary package to support online downloads
comedilib	0.7.76	Data acquisition drivers, tools, and libraries
doxygen	1.7.1	Application required for GNU Radio documentation
xmlto	0.0.10	Application required for GNU Radio documentation
git	1.7.2.1	Global Information Tracker fast version control system
mb-gcc	4.1.1	Microblaze Soft processor core designed for Xilinx FPGAs

Appendix C

Source Code listings

C1 Sampling Theorem Demonstration

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: Sampling
# Author: Morine Sithole
# Description: Illustration of sampling theorem
# Generated: Thu Jul 1 14:18:17 2010
##################################################

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 scopesink2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class Sampling(grc_wxgui.top_block_gui):
        def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="Sampling")
        _icon_path =  "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

##################################################
# Blocks
##################################################
    # Multiply function
self.gr_multiply_xx_0 = gr.multiply_vff(1)
    # Signal generator
self.gr_sig_source_x_0 = gr.sig_source_f(128000, gr.GR_SIN_WAVE, 3200, 1, 0)
self.gr_sig_source_x_1 = gr.sig_source_f(128000, gr.GR_COS_WAVE, 32000, 1, 0)
    # Throttle block to avoid CPU congestion
self.gr_throttle_0 = gr.throttle(gr.sizeof_float*1, 128000)
self.gr_throttle_1 = gr.throttle(gr.sizeof_float*1, 128000)
   # FFT display
self.wxgui_fftsink2_0 = fftsink2.fft_sink_f(
        self.GetWin(),
        baseband_freq=0,
        y_per_div=20,
        y_divs=10,
        ref_level=0,
        ref_scale=10.0,
        sample_rate=128000,
        fft_size=1024,
        fft_rate=30,
        average=True,
        avg_alpha=None,
        title="FFT Plot",
        peak_hold=True,
        )
        self.Add(self.wxgui_fftsink2_0.win)
    # Scope display
self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
        self.GetWin(),
        title="Scope Plot",
        sample_rate=128000,
        v_scale=1,
        v_offset=0,
        t_scale=0.000256,
        ac_couple=False,
        xy_mode=False,
        num_inputs=1,
        )
        self.Add(self.wxgui_scopesink2_0.win)
####################################################
# Connections
####################################################
self.connect((self.gr_sig_source_x_0, 0), (self.gr_throttle_0, 0))
self.connect((self.gr_throttle_0, 0), (self.gr_multiply_xx_0, 0))
self.connect((self.gr_sig_source_x_1, 0), (self.gr_throttle_1, 0))
self.connect((self.gr_throttle_1, 0), (self.gr_multiply_xx_0, 1))
self.connect((self.gr_multiply_xx_0, 0), (self.wxgui_fftsink2_0, 0))
self.connect((self.gr_multiply_xx_0, 0), (self.wxgui_scopesink2_0, 0))

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

C2 Amplitude Modulation

```
#!/usr/bin/env python
####################################################
# Gnuradio Python Flow Graph
# Title: Amplitude Modulation Demo
# Author: Morine Sithole
# Description: Amplitude Modulation Demo
# Generated: Fri Jul  9 10:47:15 2010
####################################################

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 gnuradio.wxgui import scopesink2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class AM_top_block(grc_wxgui.top_block_gui):
        def _init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="Amplitude Modulation Demo")
        _icon_path = "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))
```

```
##################################################
# Variables
##################################################
        self.s_rate = s_rate = 200000
        self.amplitude = amplitude = 1
##################################################
# Controls
##################################################
        _amplitude_sizer = wx.BoxSizer(wx.VERTICAL)
        self._amplitude_text_box = forms.text_box(
                parent=self.GetWin(),
                sizer=_amplitude_sizer,
                value=self.amplitude,
                callback=self.set_amplitude,
                label="Amplitude",
                converter=forms.float_converter(),
                proportion=0,
                )
        self._amplitude_slider = forms.slider(
                parent=self.GetWin(),
                sizer=_amplitude_sizer,
                value=self.amplitude,
                callback=self.set_amplitude,
                minimum=0,
                maximum=1,
                num_steps=100,
                style=wx.SL_HORIZONTAL,
                cast=float,
                proportion=1,
                )
        self.Add(_amplitude_sizer)

##################################################
# Blocks
##################################################
        self.gr_add_xx_0 = gr.add_vff(1)
        self.gr_multiply_xx_0 = gr.multiply_vff(1)
        self.gr_sig_source_x_0 = gr.sig_source_f(s_rate, gr.GR_SIN_WAVE, 2000, amplitude, 0)
        self.gr_sig_source_x_1 = gr.sig_source_f(s_rate, gr.GR_COS_WAVE, 50000, amplitude, 0)
# Throttle block to avoid CPU congestion
        self.gr_throttle_0 = gr.throttle(gr.sizeof_float*1, s_rate)
        self.gr_throttle_1 = gr.throttle(gr.sizeof_float*1, s_rate)
        self.wxgui_fftsink2_0 = fftsink2.fft_sink_f(
                self.GetWin(),
                baseband_freq=2000,
                y_per_div=5,
                y_divs=10,
                ref_level=10,
                ref_scale=10.0,
                sample_rate=s_rate,
                fft_size=1024,
                fft_rate=30,
                average=True,
                avg_alpha=None,
                title="FFT Plot",
                peak_hold=True,
                )
        self.Add(self.wxgui_fftsink2_0.win)
        self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
                self.GetWin(),
                title="Scope Plot",
                sample_rate=s_rate,
                v_scale=1,
                v_offset=0,
                t_scale=0.000192,
                ac_couple=False,
                xy_mode=False,
                num_inputs=1,
                )
        self.Add(self.wxgui_scopesink2_0.win)

##################################################
# Connections
##################################################
self.connect((self.gr_multiply_xx_0, 0), (self.gr_add_xx_0, 0))
self.connect((self.gr_add_xx_0, 0), (self.wxgui_scopesink2_0, 0))
self.connect((self.gr_add_xx_0, 0), (self.wxgui_fftsink2_0, 0))
self.connect((self.gr_sig_source_x_0, 0), (self.gr_throttle_0, 0))
self.connect((self.gr_throttle_0, 0), (self.gr_multiply_xx_0, 0))
self.connect((self.gr_sig_source_x_1, 0), (self.gr_throttle_1, 0))
self.connect((self.gr_throttle_1, 0), (self.gr_multiply_xx_0, 1))
self.connect((self.gr_throttle_1, 0), (self.gr_add_xx_0, 1))

        def set_s_rate(self, s_rate):
                self.s_rate = s_rate
                self.wxgui_fftsink2_0.set_sample_rate(self.s_rate)
                self.wxgui_scopesink2_0.set_sample_rate(self.s_rate)
                self.gr_sig_source_x_1.set_sampling_freq(self.s_rate)
                self.gr_sig_source_x_0.set_sampling_freq(self.s_rate)


        def set_amplitude(self, amplitude):
                self.amplitude = amplitude
                self._amplitude_slider.set_value(self.amplitude)
                self._amplitude_text_box.set_value(self.amplitude)
                self.gr_sig_source_x_1.set_amplitude(self.amplitude)
                self.gr_sig_source_x_0.set_amplitude(self.amplitude)

##################################################
# The start up code
##################################################
if __name__ == '__main__':
        parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
        (options, args) = parser.parse_args()
        tb = AM_top_block()
        tb.Run(True)
```

C3 FIR and IIR Filtering

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: FIR and IIR filtering
# Author: Morine Sithole
# Description: FIR and IIR filtering Demo
# Generated: Fri Aug 13 15:31:02 2010
##################################################

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 gnuradio.wxgui import scopesink2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class FIRandIIR_top_block(grc_wxgui.top_block_gui):
        def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="FIR and IIR filtering")
        _icon_path = "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

##################################################
# Variables
##################################################
        self.s_rate = s_rate = 100000
        self.amplitude = amplitude = 1

##################################################
# Notebooks
##################################################
        self.notebook_0 = wx.Notebook(self.GetWin(), style=wx.NB_TOP)
        self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab0")
        self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab1")
        self.Add(self.notebook_0)

##################################################
# Controls
##################################################
_amplitude_sizer = wx.BoxSizer(wx.VERTICAL)
        self._amplitude_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_amplitude_sizer,
        value=self.amplitude,
        callback=self.set_amplitude,
        label="Amplitude",
        converter=forms.float_converter(),
        proportion=0,
        )
self._amplitude_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_amplitude_sizer,
        value=self.amplitude,
        callback=self.set_amplitude,
        minimum=0,
        maximum=1,
        num_steps=100,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
        )
        self.Add(_amplitude_sizer)

##################################################
# Blocks
```

```
##################################################
    # FIR filter block
self.gr_fir_filter_xxx_0 = gr.fir_filter_fff(1, (3, ))
    # IIR filter block
self.gr_iir_filter_ffd_0 = gr.iir_filter_ffd((3, ), (2, ))
   # Square wave signal source
self.gr_sig_source_x_0 = gr.sig_source_f(s_rate, gr.GR_SQR_WAVE, 1000, 1, 0)
self.gr_throttle_0 = gr.throttle(gr.sizeof_float*1, s_rate)
self.wxgui_fftsink2_0 = fftsink2.fft_sink_f(
            self.notebook_0.GetPage(0).GetWin(),
            baseband_freq=1000,
            y_per_div=5,
            y_divs=10,
            ref_level=5,
            ref_scale=10.0,
            sample_rate=s_rate,
            fft_size=1024,
            fft_rate=30,
            average=True,
            avg_alpha=None,
            title="FIR FFT Plot",
            peak_hold=True,
            )
            self.notebook_0.GetPage(0).Add(self.wxgui_fftsink2_0.win)
self.wxgui_fftsink2_1 = fftsink2.fft_sink_f(
            self.notebook_0.GetPage(1).GetWin(),
            baseband_freq=1000,
            y_per_div=5,
            y_divs=10,
            ref_level=5,
            ref_scale=10.0,
            sample_rate=s_rate,
            fft_size=1024,
            fft_rate=30,
            average=True,
            avg_alpha=None,
            title="IIR FFT Plot",
            peak_hold=True,
            )
            self.notebook_0.GetPage(1).Add(self.wxgui_fftsink2_1.win)
self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
            self.notebook_0.GetPage(0).GetWin(),
            title="FIR Scope Plot",
            sample_rate=s_rate,
            v_scale=1,
            v_offset=0,
            t_scale=0.000256,
            ac_couple=True,
            xy_mode=False,
            num_inputs=1,
            )
            self.notebook_0.GetPage(0).Add(self.wxgui_scopesink2_0.win)
self.wxgui_scopesink2_1 = scopesink2.scope_sink_f(
            self.notebook_0.GetPage(1).GetWin(),
            title="IIR Scope Plot",
            sample_rate=s_rate,
            v_scale=0,
            v_offset=0,
            t_scale=0.000256,
            ac_couple=False,
        xy_mode=False,
        num_inputs=1,
        )
        self.notebook_0.GetPage(1).Add(self.wxgui_scopesink2_1.win)

##################################################
# Connections
##################################################
self.connect((self.gr_throttle_0, 0), (self.gr_fir_filter_xxx_0, 0))
self.connect((self.gr_fir_filter_xxx_0, 0), (self.wxgui_fftsink2_0, 0))
self.connect((self.gr_fir_filter_xxx_0, 0), (self.wxgui_scopesink2_0, 0))
self.connect((self.gr_throttle_0, 0), (self.gr_iir_filter_ffd_0, 0))
self.connect((self.gr_iir_filter_ffd_0, 0), (self.wxgui_fftsink2_1, 0))
self.connect((self.gr_iir_filter_ffd_0, 0), (self.wxgui_scopesink2_1, 0))
self.connect((self.gr_sig_source_x_0, 0), (self.gr_throttle_0, 0))

        def set_s_rate(self, s_rate):
                self.s_rate = s_rate
                self.gr_sig_source_x_0.set_sampling_freq(self.s_rate)
                self.wxgui_fftsink2_0.set_sample_rate(self.s_rate)
                self.wxgui_scopesink2_0.set_sample_rate(self.s_rate)
                self.wxgui_fftsink2_1.set_sample_rate(self.s_rate)
                self.wxgui_scopesink2_1.set_sample_rate(self.s_rate)

        def set_amplitude(self, amplitude):
                self.amplitude = amplitude
                self._amplitude_slider.set_value(self.amplitude)
                self._amplitude_text_box.set_value(self.amplitude)

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

C4 Amplitude Shift Keying

C4.1 Binary Amplitude Shift Keying

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: Binary Amplitude Shift keying
# Author: Morine Sithole
# Description: Binary Amplitude Shift keying
# Generated: Fri Jul  9 14:54:49 2010
##################################################

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 scopesink2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
```

```
import wx

class bask(grc_wxgui.top_block_gui):
        def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="Binary Amplitude Shift keying")
        _icon_path = "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

##################################################
# Variables
##################################################
        self.s_rate = s_rate = 2000000
##################################################
# Blocks
##################################################
        self.gr_multiply_xx_0 = gr.multiply_vss(1)
   # Data type conversion from short to float
        self.gr_short_to_float_0 = gr.short_to_float()
   # Cosine signal source
        self.gr_sig_source_x_1 = gr.sig_source_s(s_rate, gr.GR_COS_WAVE, 1000000, 2, 0)
   # Throttle block to avoid CPU congestion
        self.gr_throttle_0 = gr.throttle(gr.sizeof_short*1, s_rate)
        self.gr_throttle_1 = gr.throttle(gr.sizeof_short*1, s_rate)
   # Source of unsigned characters
        self.gr_vector_source_x_0 = gr.vector_source_s(((1,)*32+(0,)*16+(1,)*8+(0,)*8+(0,)),True,1)
        self.wxgui_fftsink2_0 = fftsink2.fft_sink_f(
                self.GetWin(),
                baseband_freq=0,
                y_per_div=10,
                y_divs=10,
                ref_level=10,
                ref_scale=5.0,
                sample_rate=s_rate,
                fft_size=512,
                fft_rate=30,
                average=False,
                avg_alpha=None,
                title="FFT Plot",
                peak_hold=False,
                )
        self.Add(self.wxgui_fftsink2_0.win)
        self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
                self.GetWin(),
                title="Scope Plot",
                sample_rate=s_rate,
                v_scale=1,
                v_offset=0,
                t_scale=0.000010,
                ac_couple=False,
                xy_mode=False,
                num_inputs=1,
                )
        self.Add(self.wxgui_scopesink2_0.win)
##################################################
# Connections
##################################################
self.connect((self.gr_short_to_float_0, 0), (self.wxgui_scopesink2_0, 0))
self.connect((self.gr_short_to_float_0, 0), (self.wxgui_fftsink2_0, 0))
self.connect((self.gr_multiply_xx_0, 0), (self.gr_short_to_float_0, 0))
self.connect((self.gr_vector_source_x_0, 0), (self.gr_throttle_0, 0))
self.connect((self.gr_throttle_0, 0), (self.gr_multiply_xx_0, 0))
self.connect((self.gr_sig_source_x_1, 0), (self.gr_throttle_1, 0))
self.connect((self.gr_throttle_1, 0), (self.gr_multiply_xx_0, 1))

        def set_s_rate(self, s_rate):
                self.s_rate = s_rate
                self.gr_sig_source_x_1.set_sampling_freq(self.s_rate)
                self.wxgui_scopesink2_0.set_sample_rate(self.s_rate)
                self.wxgui_fftsink2_0.set_sample_rate(self.s_rate)

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

C4.2 4-Level Amplitude Shift Keying

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: 4-Level Amplitude Shift Keying
# Author: Morine Sithole
# Description: 4-Level Amplitude Shift Keying
# Generated: Thu Jul  15 15:52:45 2010
##################################################

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 scopesink2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class Mary_ASK(grc_wxgui.top_block_gui):

        def __init__(self):
                grc_wxgui.top_block_gui.__init__(self, title="4-Level Amplitude Shift Keying")
                _icon_path =  "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
                self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

##################################################
# Variables
##################################################
self.s_rate = s_rate = 4000000
##################################################
# Blocks
##################################################
 self.gr_multiply_xx_0 = gr.multiply_vss(1)
 self.gr_short_to_float_0 = gr.short_to_float()
 self.gr_sig_source_x_1 = gr.sig_source_s(s_rate, gr.GR_COS_WAVE, 2000000, 1, 0)
#Throttle block to avoid CPU congestion
 self.gr_throttle_0 = gr.throttle(gr.sizeof_short*1, s_rate)
 self.gr_throttle_1 = gr.throttle(gr.sizeof_short*1, s_rate)
```

```
# Vector source to provide stream of unsigned characters to represent the four binary levels 00, 01, 10 and 11
 self.gr_vector_source_x_0 = gr.vector_source_s(((-2,)*32+(-1,)*32+(1,)*32+(3,)*32+(3,)),True, 1)
 self.wxgui_fftsink2_0 = fftsink2.fft_sink_f(
			self.GetWin(),
			baseband_freq=125000,
			y_per_div=10,
			y_divs=10,
			ref_level=5,
			ref_scale=5.0,
			sample_rate=s_rate,
			fft_size=512,
			fft_rate=30,
			average=False,
			avg_alpha=None,
			title="FFT Plot",
			peak_hold=False,
			)
 self.Add(self.wxgui_fftsink2_0.win)
 self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
			self.GetWin(),
			title="Scope Plot",
			sample_rate=s_rate,
			v_scale=1,
			v_offset=0,
			t_scale=0.000004,
			ac_couple=False,
			xy_mode=False,
			num_inputs=1,
			)
 self.Add(self.wxgui_scopesink2_0.win)
##################################################
# Connections
##################################################
self.connect((self.gr_short_to_float_0, 0), (self.wxgui_scopesink2_0, 0))
self.connect((self.gr_short_to_float_0, 0), (self.wxgui_fftsink2_0, 0))
self.connect((self.gr_multiply_xx_0, 0), (self.gr_short_to_float_0, 0))
self.connect((self.gr_sig_source_x_1, 0), (self.gr_throttle_1, 0))
self.connect((self.gr_throttle_1, 0), (self.gr_multiply_xx_0, 1))
self.connect((self.gr_vector_source_x_0, 0), (self.gr_throttle_0, 0))
self.connect((self.gr_throttle_0, 0), (self.gr_multiply_xx_0, 0))

	def set_s_rate(self, s_rate):
		self.s_rate = s_rate
		self.gr_sig_source_x_1.set_sampling_freq(self.s_rate)
		self.wxgui_scopesink2_0.set_sample_rate(self.s_rate)
		self.wxgui_fftsink2_0.set_sample_rate(self.s_rate)
##################################################
# The startup code
##################################################
if __name__ == '__main__':
	parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
	(options, args) = parser.parse_args()
	tb = Mary_ASK()
	tb.Run(True)
```

C5 Non-coherent Binary Frequency Shift Keying

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: Binary Frequency Shift Keying
# Author: Morine Sithole
# Description: Binary Frequency Shift Keying
# Generated: Fri Jul  16 15:58:14 2010
##################################################

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 scopesink2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class bfsk(grc_wxgui.top_block_gui):
        def __init__(self):
                grc_wxgui.top_block_gui.__init__(self, title="Binary Frequency Shift Keying")
                _icon_path = "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
                self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

##################################################
# Variables
##################################################
self.s_rate = s_rate = 64000
##################################################
# Blocks
##################################################
self.gr_add_xx_0 = gr.add_vff(1)
self.gr_multiply_xx_0 = gr.multiply_vff(1)
self.gr_multiply_xx_1 = gr.multiply_vff(1)
self.gr_sig_source_x_0 = gr.sig_source_f(s_rate, gr.GR_COS_WAVE, 4882.8125, 1, 0)
self.gr_sig_source_x_1 = gr.sig_source_f(s_rate, gr.GR_COS_WAVE, 7324.21875, 1, 0)
# Throttle block to avoid CPU congestion
self.gr_throttle_0 = gr.throttle(gr.sizeof_float*1, s_rate)
self.gr_throttle_1 = gr.throttle(gr.sizeof_float*1, s_rate)
self.gr_throttle_2 = gr.throttle(gr.sizeof_float*1, s_rate)
self.gr_throttle_3 = gr.throttle(gr.sizeof_float*1, s_rate)
self.gr_vector_source_x_0 = gr.vector_source_f(((1,)*32+(0,)*32), True, 1)
self.gr_vector_source_x_1 = gr.vector_source_f(((0,)*32+(1,)*32), True, 1)
self.wxgui_fftsink2_0 = fftsink2.fft_sink_f(
        self.GetWin(),
        baseband_freq=0,
        y_per_div=5,
        y_divs=10,
        ref_level=5,
        ref_scale=2.0,
        sample_rate=s_rate,
        fft_size=256,
        fft_rate=30,
        average=True,
        avg_alpha=None,
        title="FFT Plot",
        peak_hold=True,
        )
self.Add(self.wxgui_fftsink2_0.win)
self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
        self.GetWin(),
        title="Scope Plot",
			sample_rate=s_rate,
			v_scale=1,
			v_offset=0,
			t_scale=0.001024,
			ac_couple=False,
			xy_mode=False,
			num_inputs=1,
			)
		self.Add(self.wxgui_scopesink2_0.win)
		##################################################
		# Connections
		##################################################
		self.connect((self.gr_throttle_0, 0), (self.gr_multiply_xx_0, 0))
		self.connect((self.gr_throttle_1, 0), (self.gr_multiply_xx_0, 1))
		self.connect((self.gr_throttle_2, 0), (self.gr_multiply_xx_1, 0))
		self.connect((self.gr_throttle_3, 0), (self.gr_multiply_xx_1, 1))
		self.connect((self.gr_sig_source_x_0, 0), (self.gr_throttle_3, 0))
		self.connect((self.gr_vector_source_x_1, 0), (self.gr_throttle_2, 0))
		self.connect((self.gr_sig_source_x_1, 0), (self.gr_throttle_1, 0))
		self.connect((self.gr_vector_source_x_0, 0), (self.gr_throttle_0, 0))
		self.connect((self.gr_add_xx_0, 0), (self.wxgui_scopesink2_0, 0))
		self.connect((self.gr_add_xx_0, 0), (self.wxgui_fftsink2_0, 0))
		self.connect((self.gr_multiply_xx_1, 0), (self.gr_add_xx_0, 1))
		self.connect((self.gr_multiply_xx_0, 0), (self.gr_add_xx_0, 0))


	def set_s_rate(self, s_rate):
		self.s_rate = s_rate
		self.gr_sig_source_x_1.set_sampling_freq(self.s_rate)
		self.gr_sig_source_x_0.set_sampling_freq(self.s_rate)
		self.wxgui_scopesink2_0.set_sample_rate(self.s_rate)
		self.wxgui_fftsink2_0.set_sample_rate(self.s_rate)

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

C6 Continuous Phase Frequency Shift Keying

C6.1 Binary Frequency Shift Keying using a Voltage controlled oscillator

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: BFSK using VCO
# Author: Morine Sithole
# Description: BFSK using Voltage controlled oscillator
# Generated: Mon Jul  19 10:57:17 2010
##################################################

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 scopesink2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import math
import wx

class BFSK_using_VCO(grc_wxgui.top_block_gui):
        def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="BFSK using VCO")
        _icon_path = "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

##################################################
# Variables
##################################################
self.s_rate = s_rate = 128000

##################################################
# Blocks
##################################################
    # Throttle block to avoid CPU congestion
self.gr_throttle_0 = gr.throttle(gr.sizeof_float*1, s_rate)
    # Voltage controlled oscillator with stream of control voltages of data type float
self.gr_vco_f_0 = gr.vco_f(s_rate, s_rate/16*math.pi, 1)
self.gr_vector_source_x_0 = gr.vector_source_f(((1,)*32+(2,)*32), True, 1)
self.wxgui_fftsink2_0 = fftsink2.fft_sink_f(
        self.GetWin(),
        baseband_freq=64000,
        y_per_div=5,
        y_divs=10,
        ref_level=5,
        ref_scale=2.0,
        sample_rate=s_rate,
        fft_size=1024,
        fft_rate=30,
        average=True,
        avg_alpha=None,
        title="FFT Plot",
        peak_hold=True,
        )
self.Add(self.wxgui_fftsink2_0.win)
self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
        self.GetWin(),
        title="Scope Plot",
        sample_rate=s_rate,
        v_scale=1,
        v_offset=0,
        t_scale=0.000250,
        ac_couple=False,
        xy_mode=False,
        num_inputs=1,
        )
self.Add(self.wxgui_scopesink2_0.win)
##################################################
# Connections
##################################################
self.connect((self.gr_throttle_0, 0), (self.gr_vco_f_0, 0))
self.connect((self.gr_vector_source_x_0, 0), (self.gr_throttle_0, 0))
self.connect((self.gr_vco_f_0, 0), (self.wxgui_fftsink2_0, 0))
self.connect((self.gr_vco_f_0, 0), (self.wxgui_scopesink2_0, 0))
```

```
        def set_s_rate(self, s_rate):
                self.s_rate = s_rate
                self.wxgui_scopesink2_0.set_sample_rate(self.s_rate)
                self.wxgui_fftsink2_0.set_sample_rate(self.s_rate)

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

C6.2 Binary Frequency Shift Keying using the CPFSK Block

```
#!/usr/bin/env python
####################################################
# Gnuradio Python Flow Graph
# Title: Binary Frequency Shift Keying using the CPFSK Block
# Author: Morine Sithole
# Description: Continuous Phase Frequency Shift Keying
# Generated: Mon Jul  19 16:09:15 2010
####################################################

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 scopesink2
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class cpfsk(grc_wxgui.top_block_gui):

def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title=" Binary Frequency Shift Keying using the CPFSK Block")
        _icon_path = "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

####################################################
# Variables
####################################################
        self.s_rate = s_rate = 128000
####################################################
# Blocks
####################################################
    # Data packet encoder
self.blks2_packet_encoder_0 = grc_blks2.packet_mod_f(grc_blks2.packet_encoder(
        samples_per_symbol=2,
        bits_per_symbol=1,
        access_code="",
        pad_for_usrp=True,
        ),
        payload_length=0,
        )
```

```
self.gr_complex_to_real_0 = gr.complex_to_real(1)
    # CPFSK modulator block
self.gr_cpfsk_bc_0 = gr.cpfsk_bc(0.35, 1, 2)
    # Throttle block to avoid CPU congestion
self.gr_throttle_0 = gr.throttle(gr.sizeof_float*1, s_rate)
self.gr_vector_source_x_1 = gr.vector_source_f(((1,)*32+(0,)*32), True, 1)
self.wxgui_fftsink2_0 = fftsink2.fft_sink_c(
        self.GetWin(),
        baseband_freq=0,
        y_per_div=10,
        y_divs=10,
        ref_level=10,
        ref_scale=2.0,
        sample_rate=s_rate,
        fft_size=512,
        fft_rate=30,
        average=True,
        avg_alpha=None,
        title="FFT Plot",
        peak_hold=True,
        )
self.Add(self.wxgui_fftsink2_0.win)
self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
        self.GetWin(),
        title="Scope Plot",
        sample_rate=s_rate,
        v_scale=1,
        v_offset=0,
        t_scale=0,
        ac_couple=False,
        xy_mode=False,
        num_inputs=1,
        )
self.Add(self.wxgui_scopesink2_0.win)

##################################################
# Connections
##################################################
self.connect((self.gr_cpfsk_bc_0, 0), (self.wxgui_fftsink2_0, 0))
self.connect((self.blks2_packet_encoder_0, 0), (self.gr_cpfsk_bc_0, 0))
self.connect((self.gr_vector_source_x_1, 0), (self.gr_throttle_0, 0))
self.connect((self.gr_throttle_0, 0), (self.blks2_packet_encoder_0, 0))
self.connect((self.gr_cpfsk_bc_0, 0), (self.gr_complex_to_real_0, 0))
self.connect((self.gr_complex_to_real_0, 0), (self.wxgui_scopesink2_0, 0))

        def set_s_rate(self, s_rate):
                self.s_rate = s_rate
                self.wxgui_scopesink2_0.set_sample_rate(self.s_rate)
                self.wxgui_fftsink2_0.set_sample_rate(self.s_rate)

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

C7 Binary Phase Shift Keying

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: Binary Phase Shift Keying
# Author: Morine Sithole
# Description: Binary Phase Shift Keying
# Generated: Thu Jul  15 16:14:32 2010
##################################################

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 scopesink2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class bpsk(grc_wxgui.top_block_gui):
        def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="Binary Phase Shift Keying")
        _icon_path = "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

##################################################
# Variables
##################################################
        self.s_rate = s_rate = 64000
        self.delay = delay = 32000
##################################################
# Blocks
##################################################
self.gr_add_xx_0 = gr.add_vff(1)
   # Phase delay block
self.gr_delay_0 = gr.delay(gr.sizeof_float*1, delay)
   # Multiply blocks
self.gr_multiply_xx_0 = gr.multiply_vff(1)
self.gr_multiply_xx_1 = gr.multiply_vff(1)
    # Carrier signal blocks
self.gr_sig_source_x_0 = gr.sig_source_f(s_rate, gr.GR_COS_WAVE, 4882.8125, 1, 0)
self.gr_sig_source_x_1 = gr.sig_source_f(s_rate, gr.GR_COS_WAVE, 4882.8125, 1, 0)
   # Throttle block to avoid CPU congestion
self.gr_throttle_0 = gr.throttle(gr.sizeof_float*1, s_rate)
self.gr_throttle_1 = gr.throttle(gr.sizeof_float*1, s_rate)
self.gr_throttle_2 = gr.throttle(gr.sizeof_float*1, s_rate)
self.gr_throttle_3 = gr.throttle(gr.sizeof_float*1, s_rate)
self.gr_vector_source_x_0 = gr.vector_source_f(((1,)*128+(0,)*128), True, 1)
self.gr_vector_source_x_1 = gr.vector_source_f(((0,)*128+(1,)*128), True, 1)
self.wxgui_fftsink2_0 = fftsink2.fft_sink_f(
                        self.GetWin(),
                        baseband_freq=0,
                        y_per_div=5,
                        y_divs=10,
                        ref_level=5,
                        ref_scale=2.0,
                        sample_rate=s_rate,
                        fft_size=512,
                        fft_rate=30,
                        average=True,
                        avg_alpha=None,
                        title="FFT Plot",
                        peak_hold=True,
                        )
self.Add(self.wxgui_fftsink2_0.win)
self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
                        self.GetWin(),
                        title="Scope Plot",
                        sample_rate=s_rate,
                        v_scale=1,
                        v_offset=0,
                        t_scale=0.000512,
                        ac_couple=False,
                        xy_mode=False,
                        num_inputs=1,
                        )
self.Add(self.wxgui_scopesink2_0.win)

##################################################
# Connections
##################################################
self.connect((self.gr_throttle_0, 0), (self.gr_multiply_xx_0, 0))
self.connect((self.gr_throttle_1, 0), (self.gr_multiply_xx_0, 1))
self.connect((self.gr_throttle_2, 0), (self.gr_multiply_xx_1, 0))
self.connect((self.gr_delay_0, 0), (self.gr_multiply_xx_1, 1))
self.connect((self.gr_throttle_3, 0), (self.gr_delay_0, 0))
self.connect((self.gr_sig_source_x_0, 0), (self.gr_throttle_3, 0))
self.connect((self.gr_vector_source_x_1, 0), (self.gr_throttle_2, 0))
self.connect((self.gr_sig_source_x_1, 0), (self.gr_throttle_1, 0))
self.connect((self.gr_vector_source_x_0, 0), (self.gr_throttle_0, 0))
self.connect((self.gr_add_xx_0, 0), (self.wxgui_scopesink2_0, 0))
self.connect((self.gr_add_xx_0, 0), (self.wxgui_fftsink2_0, 0))
self.connect((self.gr_multiply_xx_1, 0), (self.gr_add_xx_0, 1))
self.connect((self.gr_multiply_xx_0, 0), (self.gr_add_xx_0, 0))


        def set_s_rate(self, s_rate):
                self.s_rate = s_rate
                self.gr_sig_source_x_1.set_sampling_freq(self.s_rate)
                self.gr_sig_source_x_0.set_sampling_freq(self.s_rate)
                self.wxgui_scopesink2_0.set_sample_rate(self.s_rate)
                self.wxgui_fftsink2_0.set_sample_rate(self.s_rate)

        def set_delay(self, delay):
                self.delay = delay
                self.gr_delay_0.set_delay(self.delay)

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

C8 Differential Phase Shift Keying Simulation

```
#!/usr/bin/env python
```

```
##################################################
# Gnuradio Python Flow Graph
# Title: DPSK
# Author: Morine Sithole
# Description: Differential Phase Shift Keying Simulation
# Generated: Fri Jul  23 11:40:51 2010
##################################################

from gnuradio import blks2
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 scopesink2
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class dpsk(grc_wxgui.top_block_gui):
        def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="DPSK")
        _icon_path = "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

##################################################
# Variables
##################################################
self.samp_per_sym = samp_per_sym = 2
self.s_rate = s_rate = 10000
self.noise = noise = 0.100
self.excess_bw = excess_bw = 0.35

##################################################
# Notebooks
##################################################
self.notebook_0 = wx.Notebook(self.GetWin(), style=wx.NB_TOP)
self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab0")
self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab1")
self.Add(self.notebook_0)

##################################################
# Blocks
##################################################
    #   DPBSK demodulator block
self.blks2_dxpsk_demod_0 = blks2.dbpsk_demod(
        samples_per_symbol=samp_per_sym,
        excess_bw=0.35,
        costas_alpha=0.175,
        gain_mu=0.175,
        mu=0.5,
        omega_relative_limit=0.005,
        gray_code=True,
        verbose=False,
        log=False,
        )
    #   DPBSK modulator block
self.blks2_dxpsk_mod_0 = blks2.dbpsk_mod(
        samples_per_symbol=samp_per_sym,
        excess_bw=excess_bw,
        gray_code=True,
        verbose=False,
        log=False,
        )
    #     Packet decoder block
self.blks2_packet_decoder_0 = rc_blks2.packet_demod_f(
        grc_blks2.packet_decoder(
        access_code="",
        threshold=-1,
        callback=lambda ok, payload: self.blks2_packet_decoder_0.recv_pkt(ok,  payload),
        ),
        )
    #     Packet encoder block
self.blks2_packet_encoder_0 = grc_blks2.packet_mod_f(grc_blks2.packet_encoder(
        samples_per_symbol=samp_per_sym,
        bits_per_symbol=1,
        access_code="",
        pad_for_usrp=True,
        ),
        payload_length=0,
        )
self.gr_add_xx_0 = gr.add_vcc(1)
    #     File output block
self.gr_file_sink_0 = gr.file_sink(gr.sizeof_float*1, "/home/user/Documents/DPSK/output")
    #     File input block
self.gr_file_source_0 = gr.file_source(gr.sizeof_float*1, "/home/user/Documents/DPSK/test", True)
    #     Additive White Gaussian Noise Source
self.gr_noise_source_x_0 = gr.noise_source_c(gr.GR_GAUSSIAN, noise, 42)
self.gr_throttle_0_0 = gr.throttle(gr.sizeof_float*1, s_rate)
self.wxgui_fftsink2_0 = fftsink2.fft_sink_c(
        self.notebook_0.GetPage(1).GetWin(),
        baseband_freq=0,
        y_per_div=10,
        y_divs=10,
        ref_level=10,
        ref_scale=2.0,
        sample_rate=s_rate,
        fft_size=512,
        fft_rate=30,
        average=True,
        avg_alpha=None,
        title="FFT Plot of Modulated signal",
        peak_hold=True,
        )
self.notebook_0.GetPage(1).Add(self.wxgui_fftsink2_0.win)
self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
        self.notebook_0.GetPage(0).GetWin(),
        title="Scope Plot of demodulated, decoded signal",
        sample_rate=s_rate,
        v_scale=0.000000001,
        v_offset=0,
        t_scale=20/s_rate,
        ac_couple=False,
        xy_mode=False,
        num_inputs=1,
        )
self.notebook_0.GetPage(0).Add(self.wxgui_scopesink2_0.win)
self.wxgui_scopesink2_1 = scopesink2.scope_sink_f(
        self.notebook_0.GetPage(0).GetWin(),
        title="Scope Plot of Waveform from File Source",
        sample_rate=s_rate,
        v_scale=0.000000001,
        v_offset=0,
        t_scale=20/s_rate,
        ac_couple=False,
        xy_mode=False,
        num_inputs=1,
        )
self.notebook_0.GetPage(0).Add(self.wxgui_scopesink2_1.win)
self.wxgui_scopesink2_2 = scopesink2.scope_sink_c(
        self.notebook_0.GetPage(1).GetWin(),
        title="Scope Plot of Modulated Signal",
        sample_rate=s_rate,
        v_scale=0,
        v_offset=0,
        t_scale=20/s_rate,
        ac_couple=False,
        xy_mode=False,
        num_inputs=1,
        )
self.notebook_0.GetPage(1).Add(self.wxgui_scopesink2_2.win)

##################################################
# Connections
##################################################
self.connect((self.gr_add_xx_0, 0), (self.blks2_dxpsk_demod_0, 0))
self.connect((self.gr_noise_source_x_0, 0), (self.gr_add_xx_0, 1))
self.connect((self.blks2_dxpsk_mod_0, 0), (self.gr_add_xx_0, 0))
self.connect((self.gr_throttle_0_0, 0), (self.wxgui_scopesink2_1, 0))
self.connect((self.blks2_dxpsk_mod_0, 0), (self.wxgui_scopesink2_2, 0))
self.connect((self.blks2_dxpsk_mod_0, 0), (self.wxgui_fftsink2_0, 0))
self.connect((self.gr_file_source_0, 0), (self.gr_throttle_0_0, 0))
self.connect((self.blks2_packet_decoder_0, 0), (self.gr_file_sink_0, 0))
self.connect((self.blks2_packet_decoder_0, 0), (self.wxgui_scopesink2_0, 0))
self.connect((self.gr_throttle_0_0, 0), (self.blks2_packet_encoder_0, 0))
self.connect((self.blks2_packet_encoder_0, 0), (self.blks2_dxpsk_mod_0, 0))
self.connect((self.blks2_dxpsk_demod_0, 0), (self.blks2_packet_decoder_0, 0))

        def set_samp_per_sym(self, samp_per_sym):
                        self.samp_per_sym = samp_per_sym

        def set_s_rate(self, s_rate):
                        self.s_rate = s_rate
                        self.wxgui_scopesink2_2.set_sample_rate(self.s_rate)
                        self.wxgui_fftsink2_0.set_sample_rate(self.s_rate)
                        self.wxgui_scopesink2_0.set_sample_rate(self.s_rate)
                        self.wxgui_scopesink2_1.set_sample_rate(self.s_rate)

        def set_noise(self, noise):
                        self.noise = noise
                        self.gr_noise_source_x_0.set_amplitude(self.noise)

        def set_excess_bw(self, excess_bw):
                        self.excess_bw = excess_bw

##################################################
# start up
##################################################
```

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

C9 Differential Phase Shift Keying using the USRP2

C9.1 Transmitter

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: DPSK TX over USRP2
# Author: Morine Sithole
# Description: gnuradio DPSK flow graph using USRP2
# Generated: Fri Aug 13 11:12:43 2010
##################################################

from gnuradio import blks2
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import usrp2
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 scopesink2
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class dpsk_using_USRP2(grc_wxgui.top_block_gui):
	def __init__(self):
	grc_wxgui.top_block_gui.__init__(self, title="DPSK TX over USRP2")
	_icon_path = "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
	self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

##################################################
# Variables
##################################################
	self.samp_per_sym = samp_per_sym = 2
	self.s_rate = s_rate = 97656.25
	self.interpol = interpol = 512
	self.excess_bw = excess_bw = 0.35
	self.decim = decim = 512

##################################################
# Notebooks
##################################################
self.notebook_0 = wx.Notebook(self.GetWin(), style=wx.NB_TOP)
self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab0")
self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab1")
self.Add(self.notebook_0)

##################################################
# Blocks
##################################################
self.blks2_dxpsk_mod_0 = blks2.dbpsk_mod(
	samples_per_symbol=samp_per_sym,
            excess_bw=excess_bw,
            gray_code=True,
            verbose=False,
            log=False,
            )
self.blks2_packet_encoder_0 = grc_blks2.packet_mod_f(grc_blks2.packet_encoder(
            samples_per_symbol=samp_per_sym,
            bits_per_symbol=1,
            access_code="",
            pad_for_usrp=True,
            ),
            payload_length=0,
            )
self.gr_file_source_0 = gr.file_source(gr.sizeof_float*1, "/home/user/Documents/DPSK/test", True)
self.gr_throttle_0_0 = gr.throttle(gr.sizeof_float*1, s_rate)
            # USRP2 interface
self.usrp2_sink_xxxx_0 = usrp2.sink_32fc("eth1")
self.usrp2_sink_xxxx_0.set_interp(interpol)
self.usrp2_sink_xxxx_0.set_center_freq(2440000000)
self.usrp2_sink_xxxx_0.set_gain(20)
self.usrp2_sink_xxxx_0.config_mimo(usrp2.MC_WE_DONT_LOCK)
self.wxgui_fftsink2_0 = fftsink2.fft_sink_c(
            self.notebook_0.GetPage(1).GetWin(),
            baseband_freq=2440000000,
            y_per_div=10,
            y_divs=10,
            ref_level=10,
            ref_scale=2.0,
            sample_rate=s_rate,
            fft_size=512,
            fft_rate=30,
            average=True,
            avg_alpha=None,
            title="FFT Plot of Modulated Signal",
            peak_hold=True,
            )
            self.notebook_0.GetPage(1).Add(self.wxgui_fftsink2_0.win)
self.wxgui_fftsink2_3 = fftsink2.fft_sink_f(
            self.notebook_0.GetPage(0).GetWin(),
            baseband_freq=0,
            y_per_div=20,
            y_divs=10,
            ref_level=10,
            ref_scale=2.0,
            sample_rate=s_rate,
            fft_size=1024,
            fft_rate=30,
            average=True,
            avg_alpha=None,
            title="FFT Plot of Waveform from File Source",
            peak_hold=True,
            )
            self.notebook_0.GetPage(0).Add(self.wxgui_fftsink2_3.win)
self.wxgui_scopesink2_1 = scopesink2.scope_sink_f(
            self.notebook_0.GetPage(0).GetWin(),
            title="Scope Plot of Waveform from File Source",
            sample_rate=s_rate,
            v_scale=0,
            v_offset=0,
            t_scale=20/s_rate,
		ac_couple=False,
		xy_mode=False,
		num_inputs=1,
		trig_mode=gr.gr_TRIG_MODE_AUTO,
		)
		self.notebook_0.GetPage(0).Add(self.wxgui_scopesink2_1.win)
	self.wxgui_scopesink2_2 = scopesink2.scope_sink_c(
		self.notebook_0.GetPage(1).GetWin(),
		title="Scope Plot of modulated signal",
		sample_rate=s_rate,
		v_scale=0,
		v_offset=0,
		t_scale=20/s_rate,
		ac_couple=False,
		xy_mode=False,
		num_inputs=1,
		trig_mode=gr.gr_TRIG_MODE_AUTO,
		)
		self.notebook_0.GetPage(1).Add(self.wxgui_scopesink2_2.win)
	##################################################
	# Connections
	##################################################
	self.connect((self.gr_file_source_0, 0), (self.wxgui_scopesink2_1, 0))
	self.connect((self.gr_file_source_0, 0), (self.gr_throttle_0_0, 0))
	self.connect((self.blks2_packet_encoder_0, 0), (self.blks2_dxpsk_mod_0, 0))
	self.connect((self.gr_file_source_0, 0), (self.wxgui_fftsink2_3, 0))
	self.connect((self.blks2_dxpsk_mod_0, 0), (self.wxgui_fftsink2_0, 0))
	self.connect((self.blks2_dxpsk_mod_0, 0), (self.wxgui_scopesink2_2, 0))
	self.connect((self.gr_throttle_0_0, 0), (self.blks2_packet_encoder_0, 0))
	self.connect((self.blks2_dxpsk_mod_0, 0), (self.usrp2_sink_xxxx_0, 0))

		def set_samp_per_sym(self, samp_per_sym):
			self.samp_per_sym = samp_per_sym

		def set_s_rate(self, s_rate):
			self.s_rate = s_rate
			self.wxgui_scopesink2_1.set_sample_rate(self.s_rate)
			self.wxgui_fftsink2_3.set_sample_rate(self.s_rate)
			self.wxgui_fftsink2_0.set_sample_rate(self.s_rate)
			self.wxgui_scopesink2_2.set_sample_rate(self.s_rate)

		def set_interpol(self, interpol):
			self.interpol = interpol
			self.usrp2_sink_xxxx_0.set_interp(self.interpol)

		def set_excess_bw(self, excess_bw):
			self.excess_bw = excess_bw

		def set_decim(self, decim):
			self.decim = decim
	##################################################
	# start up code
	##################################################
	if __name__ == '__main__':
		parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
		(options, args) = parser.parse_args()
		tb = dpsk_using_USRP2()
		tb.Run(True)
```

C9.2 Receiver

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: DPSK RX over USRP2
# Author: Morine Sithole
# Description: gnuradio DPSK flow graph using USRP2
# Generated: Fri Aug 13 10:59:05 2010
##################################################

from gnuradio import blks2
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import usrp2
from gnuradio import window
from gnuradio.eng_option import eng_option
from gnuradio.gr import firdes
from gnuradio.wxgui import constsink_gl
from gnuradio.wxgui import fftsink2
from gnuradio.wxgui import scopesink2
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class dpsk_using_USRP2(grc_wxgui.top_block_gui):
        def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="DPSK RX over USRP2")
        _icon_path = "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

##################################################
# Variables
##################################################
        self.samp_per_sym = samp_per_sym = 2
        self.s_rate = s_rate = 97656.25
        self.interpol = interpol = 512
        self.excess_bw = excess_bw = 0.35
        self.decim = decim = 512

##################################################
# Notebooks
##################################################
        self.notebook_0 = wx.Notebook(self.GetWin(), style=wx.NB_TOP)
        self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab0")
        self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab1")
        self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab2")
        self.Add(self.notebook_0)

##################################################
# Blocks
##################################################
self.blks2_dxpsk_demod_0 = blks2.dbpsk_demod(
        samples_per_symbol=samp_per_sym,
        excess_bw=0.35,
        costas_alpha=0.175,
        gain_mu=0.175,
        mu=0.5,
        omega_relative_limit=0.005,
        gray_code=True,
        verbose=False,
        log=False,
        )
self.blks2_packet_decoder_0 = grc_blks2.packet_demod_f(grc_blks2.packet_decoder(
        access_code="",
        threshold=-1,
        callback=lambda ok, payload: self.blks2_packet_decoder_0.recv_pkt(ok, payload),
        ),
        )
self.gr_file_sink_0 = gr.file_sink(gr.sizeof_float*1, "/home/user/Documents/DPSK/output")
self.gr_multiply_const_vxx_0 = gr.multiply_const_vcc((1000, ))
self.gr_throttle_1 = gr.throttle(gr.sizeof_gr_complex*1, s_rate)
        # USRP2 interface
self.usrp2_source_xxxx_0 = usrp2.source_32fc("eth0")
self.usrp2_source_xxxx_0.set_decim(decim)
self.usrp2_source_xxxx_0.set_center_freq(2440000000)
self.usrp2_source_xxxx_0.set_gain(20)
self.usrp2_source_xxxx_0.config_mimo(usrp2.MC_WE_DONT_LOCK)
self.wxgui_constellationsink2_0 = constsink_gl.const_sink_c(
        self.notebook_0.GetPage(2).GetWin(),
        title="Constellation Plot of received signal",
        sample_rate=s_rate,
        frame_rate=5,
        const_size=2048,
        M=4,
        theta=0,
        alpha=0.005,
        fmax=0.06,
        mu=0.5,
        gain_mu=0.005,
        symbol_rate=s_rate/samp_per_sym,
        omega_limit=0.005,
        )
        self.notebook_0.GetPage(2).Add(self.wxgui_constellationsink2_0.win)
self.wxgui_fftsink2_1 = fftsink2.fft_sink_c(
        self.notebook_0.GetPage(1).GetWin(),
        baseband_freq=2440000000,
        y_per_div=10,
        y_divs=10,
        ref_level=20,
        ref_scale=2.0,
        sample_rate=s_rate,
        fft_size=1024,
        fft_rate=30,
        average=True,
        avg_alpha=None,
        title="FFT Plot of Received Signal",
        peak_hold=True,
        )
        self.notebook_0.GetPage(1).Add(self.wxgui_fftsink2_1.win)
self.wxgui_fftsink2_2 = fftsink2.fft_sink_f(
        self.notebook_0.GetPage(0).GetWin(),
        baseband_freq=0,
        y_per_div=10,
        y_divs=10,
        ref_level=50,
        ref_scale=2.0,
        sample_rate=s_rate,
        fft_size=1024,
        fft_rate=30,
		average=True,
		avg_alpha=None,
		title="FFT Plot of demodulated, decoded signal",
		peak_hold=True,
		)
		self.notebook_0.GetPage(0).Add(self.wxgui_fftsink2_2.win)
	self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
		self.notebook_0.GetPage(0).GetWin(),
		title="Scope Plot of Demodulated, decoded signal",
		sample_rate=s_rate,
		v_scale=0,
		v_offset=0,
		t_scale=20/s_rate,
		ac_couple=False,
		xy_mode=False,
		num_inputs=1,
		)
		self.notebook_0.GetPage(0).Add(self.wxgui_scopesink2_0.win)
	self.wxgui_scopesink2_3 = scopesink2.scope_sink_c(
		self.notebook_0.GetPage(1).GetWin(),
		title="Scope Plot of Received Signal",
		sample_rate=s_rate,
		v_scale=0,
		v_offset=0,
		t_scale=20/s_rate,
		ac_couple=False,
		xy_mode=False,
		num_inputs=1,
		)
		self.notebook_0.GetPage(1).Add(self.wxgui_scopesink2_3.win)

	##################################################
	# Connections
	##################################################
	self.connect((self.usrp2_source_xxxx_0, 0), (self.gr_throttle_1, 0))
	self.connect((self.blks2_dxpsk_demod_0, 0), (self.blks2_packet_decoder_0, 0))
	self.connect((self.blks2_packet_decoder_0, 0), (self.wxgui_scopesink2_0, 0))
	self.connect((self.blks2_packet_decoder_0, 0), (self.gr_file_sink_0, 0))
	self.connect((self.blks2_packet_decoder_0, 0), (self.wxgui_fftsink2_2, 0))
	self.connect((self.gr_throttle_1, 0), (self.wxgui_constellationsink2_0, 0))
	self.connect((self.gr_throttle_1, 0), (self.gr_multiply_const_vxx_0, 0))
	self.connect((self.gr_multiply_const_vxx_0, 0), (self.wxgui_fftsink2_1, 0))
	self.connect((self.gr_multiply_const_vxx_0, 0), (self.wxgui_scopesink2_3, 0))
	self.connect((self.gr_multiply_const_vxx_0, 0), (self.blks2_dxpsk_demod_0, 0))

		def set_samp_per_sym(self, samp_per_sym):
			self.samp_per_sym = samp_per_sym

		def set_s_rate(self, s_rate):
			self.s_rate = s_rate
			self.wxgui_constellationsink2_0.set_sample_rate(self.s_rate)
			self.wxgui_scopesink2_0.set_sample_rate(self.s_rate)
			self.wxgui_fftsink2_2.set_sample_rate(self.s_rate)
			self.wxgui_scopesink2_3.set_sample_rate(self.s_rate)
			self.wxgui_fftsink2_1.set_sample_rate(self.s_rate)

		def set_interpol(self, interpol):
			self.interpol = interpol

		def set_excess_bw(self, excess_bw):
```

```
                self.excess_bw = excess_bw

        def set_decim(self, decim):
                self.decim = decim
                self.usrp2_source_xxxx_0.set_decim(self.decim)

##################################################
# Start up
##################################################
if __name__ == '__main__':
        parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
        (options, args) = parser.parse_args()
        tb = dpsk_using_USRP2()
        tb.Run(True)
```

C10 Gaussian Minimum Shift Keying Simulation

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: Gaussian Minimum Shift Keying Simulation
# Author: Morine Sithole
# Description: Gaussian Minimum Shift Keying Simulation
# Generated: Fri Jul  30 12:15:13 2010
##################################################

from gnuradio import audio
from gnuradio import blks2
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 scopesink2
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class gmsk(grc_wxgui.top_block_gui):
        def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="Gaussian Minimum Shift Keying Simulation")
        _icon_path = "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

##################################################
# Variables
##################################################
        self.time_bw = time_bw = 0.35
        self.samp_per_sym = samp_per_sym = 2
        self.s_rate = s_rate = 256000
        self.noise = noise = 0.100

##################################################
# Notebooks
##################################################
self.notebook_0 = wx.Notebook(self.GetWin(), style=wx.NB_TOP)
self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab0")
self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab1")
self.Add(self.notebook_0)
```

```
##################################################
# Blocks
##################################################
self.audio_sink_0 = audio.sink(44100, "", True)
self.blks2_gmsk_demod_0 = blks2.gmsk_demod(
        samples_per_symbol=samp_per_sym,
        gain_mu=0.5,
        mu=0.5,
        omega_relative_limit=0.005,
        freq_error=0.0,
        verbose=False,
        log=False,
        )
self.blks2_gmsk_mod_0 = blks2.gmsk_mod(
        samples_per_symbol=samp_per_sym,
        bt=time_bw,
        verbose=False,
        log=False,
        )
self.blks2_packet_decoder_0 = grc_blks2.packet_demod_f(grc_blks2.packet_decoder(
        access_code="",
        threshold=-1,
        callback=lambda ok, payload: self.blks2_packet_decoder_0.recv_pkt(ok, payload),
        ),
        )
self.blks2_packet_encoder_0 = grc_blks2.packet_mod_f(grc_blks2.packet_encoder(
        samples_per_symbol=samp_per_sym,
        bits_per_symbol=1,
        access_code="",
        pad_for_usrp=True,
        ),
        payload_length=0,
        )
self.blks2_rational_resampler_xxx_0 = blks2.rational_resampler_fff(
        interpolation=2560,
        decimation=441,
        taps=None,
        fractional_bw=None,
        )
self.gr_channel_model_0 = gr.channel_model(
        noise_voltage=noise,
        frequency_offset=0.0,
        epsilon=1.0,
        taps=(1.0 + 1.0j,
        ),
        noise_seed=42,
        )
self.gr_complex_to_real_0 = gr.complex_to_real(1)
# Throttle block to avoid CPU congestion
self.gr_throttle_0 = gr.throttle(gr.sizeof_float*1, s_rate)
self.gr_wavfile_source_0 = gr.wavfile_source("/home/user/408.wav", True)
self.wxgui_fftsink2_0 = fftsink2.fft_sink_c(
        self.notebook_0.GetPage(1).GetWin(),
        baseband_freq=0,
        y_per_div=10,
        y_divs=10,
        ref_level=10,
        ref_scale=2.0,
        sample_rate=s_rate,
        fft_size=512,
        fft_rate=30,
        average=True,
        avg_alpha=None,
        title="FFT Plot of GMSK Modulated Signal",
        peak_hold=True,
        )
self.notebook_0.GetPage(1).Add(self.wxgui_fftsink2_0.win)
self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
        self.notebook_0.GetPage(0).GetWin(),
        title="Demodulated, decoded signal",
        sample_rate=s_rate,
        v_scale=0,
        v_offset=0,
        t_scale=0.000020,
        ac_couple=False,
        xy_mode=False,
        num_inputs=1,
        )
self.notebook_0.GetPage(0).Add(self.wxgui_scopesink2_0.win)
self.wxgui_scopesink2_1 = scopesink2.scope_sink_f(
        self.notebook_0.GetPage(0).GetWin(),
        title="Signal from wav file source",
        sample_rate=s_rate,
        v_scale=0,
        v_offset=0,
        t_scale=0.000020,
        ac_couple=False,
        xy_mode=False,
        num_inputs=1,
        )
self.notebook_0.GetPage(0).Add(self.wxgui_scopesink2_1.win)
self.wxgui_scopesink2_2 = scopesink2.scope_sink_f(
        self.notebook_0.GetPage(1).GetWin(),
        title="Scope Plot of GMSK Modulated Signal",
        sample_rate=s_rate,
        v_scale=0,
        v_offset=0,
        t_scale=0.000020,
        ac_couple=False,
        xy_mode=False,
        num_inputs=1,
        )
self.notebook_0.GetPage(1).Add(self.wxgui_scopesink2_2.win)

##################################################
# Connections
##################################################
self.connect((self.blks2_gmsk_mod_0, 0), (self.wxgui_fftsink2_0, 0))
self.connect((self.blks2_packet_encoder_0, 0), (self.blks2_gmsk_mod_0, 0))
self.connect((self.blks2_gmsk_demod_0, 0), (self.blks2_packet_decoder_0, 0))
self.connect((self.blks2_packet_decoder_0, 0), (self.wxgui_scopesink2_0, 0))
self.connect((self.gr_wavfile_source_0, 0), (self.wxgui_scopesink2_1, 0))
self.connect((self.blks2_gmsk_mod_0, 0), (self.gr_channel_model_0, 0))
self.connect((self.gr_channel_model_0, 0), (self.blks2_gmsk_demod_0, 0))
self.connect((self.blks2_packet_decoder_0, 0), (self.blks2_rational_resampler_xxx_0, 0))
self.connect((self.blks2_rational_resampler_xxx_0, 0), (self.audio_sink_0, 0))
self.connect((self.gr_wavfile_source_0, 0), (self.gr_throttle_0, 0))
self.connect((self.gr_throttle_0, 0), (self.blks2_packet_encoder_0, 0))
self.connect((self.gr_complex_to_real_0, 0), (self.wxgui_scopesink2_2, 0))
self.connect((self.blks2_gmsk_mod_0, 0), (self.gr_complex_to_real_0, 0))

        def set_time_bw(self, time_bw):
                self.time_bw = time_bw

        def set_samp_per_sym(self, samp_per_sym):
                self.samp_per_sym = samp_per_sym

        def set_s_rate(self, s_rate):
                self.s_rate = s_rate
                self.wxgui_fftsink2_0.set_sample_rate(self.s_rate)
                self.wxgui_scopesink2_0.set_sample_rate(self.s_rate)
                self.wxgui_scopesink2_1.set_sample_rate(self.s_rate)
                self.wxgui_scopesink2_2.set_sample_rate(self.s_rate)

        def set_noise(self, noise):
                self.noise = noise
                self.gr_channel_model_0.set_noise_voltage(self.noise)

##################################################
# Start up
##################################################

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

C11 Gaussian Minimum Shift Keying using the USRP2

C11.1 Transmitter

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: GMSK TX over USRP2
# Author: Morine Sithole
# Description: Transmitting and receiving a Microsoft *.wav file
# Generated: Fri Aug 13 11:26:12 2010
##################################################

from gnuradio import blks2
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import usrp2
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 scopesink2
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class GMSK_using_USRP(grc_wxgui.top_block_gui):
        def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="GMSK TX over USRP2")
        _icon_path =
"/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
		self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

##################################################
# Variables
##################################################
		self.samp_per_sym = samp_per_sym = 2
		self.s_rate = s_rate = 256000
		self.interpol = interpol = 196
		self.excess_bw = excess_bw = 0.35
		self.decim = decim = 196

##################################################
# Notebooks
##################################################
self.notebook_0 = wx.Notebook(self.GetWin(), style=wx.NB_TOP)
self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab0")
self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab1")
self.Add(self.notebook_0)

##################################################
# Blocks
##################################################
self.blks2_gmsk_mod_0 = blks2.gmsk_mod(
		samples_per_symbol=samp_per_sym,
		bt=excess_bw,
		verbose=False,
		log=False,
		)
self.blks2_packet_encoder_0 = grc20_blks2.packet_mod_f(grc_blks2.packet_encoder(
		samples_per_symbol=samp_per_sym,
		bits_per_symbol=1,
		access_code="",
		pad_for_usrp=True,
		),
		payload_length=0,
		)
self.gr_complex_to_real_0 = gr.complex_to_real(1)
self.gr_throttle_0 = gr.throttle(gr.sizeof_float*1, s_rate)
self.gr_wavfile_source_0 = gr.wavfile_source("/home/user/408.wav", True)
self.usrp2_sink_xxxx_0 = usrp2.sink_32fc("eth1")
self.usrp2_sink_xxxx_0.set_interp(interpol)
self.usrp2_sink_xxxx_0.set_center_freq(2460000000)
self.usrp2_sink_xxxx_0.set_gain(20)
self.usrp2_sink_xxxx_0.config_mimo(usrp2.MC_WE_DONT_LOCK)
self.wxgui_fftsink2_0 = fftsink2.fft_sink_c(
		self.notebook_0.GetPage(1).GetWin(),
		baseband_freq=0,
		y_per_div=10,
		y_divs=10,
		ref_level=10,
		ref_scale=2.0,
		sample_rate=s_rate,
		fft_size=512,
		fft_rate=30,
		average=True,
		avg_alpha=None,
		title="FFT Plot of GMSK Modulated Signal",
		peak_hold=True,
		)
		self.notebook_0.GetPage(1).Add(self.wxgui_fftsink2_0.win)
	self.wxgui_fftsink2_3 = fftsink2.fft_sink_f(
		self.notebook_0.GetPage(0).GetWin(),
		baseband_freq=0,
		y_per_div=10,
		y_divs=10,
		ref_level=5,
		ref_scale=2.0,
		sample_rate=s_rate,
		fft_size=1024,
		fft_rate=30,
		average=True,
		avg_alpha=None,
		title="FFT Plot of Signal from wav file source",
		peak_hold=True,
		)
		self.notebook_0.GetPage(0).Add(self.wxgui_fftsink2_3.win)
	self.wxgui_scopesink2_1 = scopesink2.scope_sink_f(
		self.notebook_0.GetPage(0).GetWin(),
		title="Scope Plot of Signal from wav file source",
		sample_rate=s_rate,
		v_scale=0,
		v_offset=0,
		t_scale=0.000020,
		ac_couple=False,
		xy_mode=False,
		num_inputs=1,
		trig_mode=gr.gr_TRIG_MODE_AUTO,
		)
	self.notebook_0.GetPage(0).Add(self.wxgui_scopesink2_1.win)
	self.wxgui_scopesink2_2 = scopesink2.scope_sink_f(
		self.notebook_0.GetPage(1).GetWin(),
		title="Scope Plot of GMSK Modulated Signal",
		sample_rate=s_rate,
		v_scale=0,
		v_offset=0,
		t_scale=0.000020,
		ac_couple=False,
		xy_mode=False,
		num_inputs=1,
		trig_mode=gr.gr_TRIG_MODE_AUTO,
		)
		self.notebook_0.GetPage(1).Add(self.wxgui_scopesink2_2.win)

	##################################################
	# Connections
	##################################################
	self.connect((self.gr_wavfile_source_0, 0), (self.wxgui_scopesink2_1, 0))
	self.connect((self.gr_wavfile_source_0, 0), (self.gr_throttle_0, 0))
	self.connect((self.gr_throttle_0, 0), (self.blks2_packet_encoder_0, 0))
	self.connect((self.blks2_packet_encoder_0, 0), (self.blks2_gmsk_mod_0, 0))
	self.connect((self.gr_wavfile_source_0, 0), (self.wxgui_fftsink2_3, 0))
	self.connect((self.gr_complex_to_real_0, 0), (self.wxgui_scopesink2_2, 0))
	self.connect((self.blks2_gmsk_mod_0, 0), (self.gr_complex_to_real_0, 0))
	self.connect(self.blks2_gmsk_mod_0, 0), (self.usrp2_sink_xxxx_0, 0))
	self.connect((self.blks2_gmsk_mod_0, 0), (self.wxgui_fftsink2_0, 0))

		def set_samp_per_sym(self, samp_per_sym):
			self.samp_per_sym = samp_per_sym
	def set_s_rate(self, s_rate):
		self.s_rate = s_rate
		self.wxgui_scopesink2_1.set_sample_rate(self.s_rate)
		self.wxgui_fftsink2_3.set_sample_rate(self.s_rate)
		self.wxgui_scopesink2_2.set_sample_rate(self.s_rate)
		self.wxgui_fftsink2_0.set_sample_rate(self.s_rate)

	def set_interpol(self, interpol):
		self.interpol = interpol
		self.usrp2_sink_xxxx_0.set_interp(self.interpol)

	def set_excess_bw(self, excess_bw):
		self.excess_bw = excess_bw

	def set_decim(self, decim):
		self.decim = decim

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

C11.2 Receiver

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: GMSK RX over USRP2
# Author: Morine Sithole
# Description: Transmitting and receiving a Microsoft *.wav file
# Generated: Fri Aug 20 11:12:28 2010
##################################################

from gnuradio import audio
from gnuradio import blks2
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import usrp2
from gnuradio import window
from gnuradio.eng_option import eng_option
from gnuradio.gr import firdes
from gnuradio.wxgui import constsink_gl
from gnuradio.wxgui import fftsink2
from gnuradio.wxgui import scopesink2
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import wx

class GMSK_using_USRP(grc_wxgui.top_block_gui):
	def __init__(self):
	grc_wxgui.top_block_gui.__init__(self, title="GMSK RX over USRP2")
	_icon_path = "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
	self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

##################################################
# Variables
```

```
##################################################
            self.samp_per_sym = samp_per_sym = 2
            self.s_rate = s_rate = 256000
            self.interpol = interpol = 196
            self.excess_bw = excess_bw = 0.35
            self.decim = decim = 196

##################################################
# Notebooks
##################################################
            self.notebook_0 = wx.Notebook(self.GetWin(), style=wx.NB_TOP)
            self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab0")
            self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab1")
            self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab2")
            self.Add(self.notebook_0)

##################################################
# Blocks
##################################################
self.audio_sink_0 = audio.sink(44100, "", True)
self.blks2_gmsk_demod_0 = blks2.gmsk_demod(
            samples_per_symbol=samp_per_sym,
            gain_mu=0.5,
            mu=0.5,
            omega_relative_limit=0.005,
            freq_error=0.0,
            verbose=False,
            log=False,
            )
self.blks2_packet_decoder_0 = grc_blks2.packet_demod_f(grc_blks2.packet_decoder(
            access_code="",
            threshold=-1,
            callback=lambda ok, payload: self.blks2_packet_decoder_0.recv_pkt(ok, payload),
            ),
            )
self.blks2_rational_resampler_xxx_0 = blks2.rational_resampler_fff(
            interpolation=2560,
            decimation=441,
            taps=None,
            fractional_bw=None,
            )
self.gr_throttle_2 = gr.throttle(gr.sizeof_gr_complex*1, s_rate)
self.usrp2_source_xxxx_0 = usrp2.source_32fc()
self.usrp2_source_xxxx_0.set_decim(decim)
self.usrp2_source_xxxx_0.set_center_freq(2460000000)
self.usrp2_source_xxxx_0.set_gain
self.usrp2_source_xxxx_0.config_mimo(usrp2.MC_WE_DONT_LOCK)
self.wxgui_constellationsink2_0 = constsink_gl.const_sink_c(
            self.notebook_0.GetPage(2).GetWin(),
            title="Constellation Plot of received signal",
            sample_rate=s_rate,
            frame_rate=5,
            const_size=2048,
            M=4,
            theta=0,
            alpha=0.005,
            fmax=0.06,
            mu=0.5,
            gain_mu=0.005,
            symbol_rate=s_rate/samp_per_sym,
		omega_limit=0.005,
		)
		self.notebook_0.GetPage(2).Add(self.wxgui_constellationsink2_0.win)
	self.wxgui_fftsink2_1 = fftsink2.fft_sink_f(
		self.notebook_0.GetPage(1).GetWin(),
		baseband_freq=0,
		y_per_div=10,
		y_divs=10,
		ref_level=5,
		ref_scale=2.0,
		sample_rate=s_rate,
		fft_size=1024,
		fft_rate=30,
		average=False,
		avg_alpha=None,
		title="FFT Plot of demodulated signal",
		peak_hold=False,
		)
		self.notebook_0.GetPage(1).Add(self.wxgui_fftsink2_1.win)
	self.wxgui_fftsink2_2 = fftsink2.fft_sink_c(
		self.notebook_0.GetPage(0).GetWin(),
		baseband_freq=2460000000,
		y_per_div=10,
		y_divs=10,
		ref_level=5,
		ref_scale=2.0,
		sample_rate=s_rate,
		fft_size=1024,
		fft_rate=30,
		average=True,
		avg_alpha=None,
		title="FFT Plot of received signal",
		peak_hold=True,
		)
		self.notebook_0.GetPage(0).Add(self.wxgui_fftsink2_2.win)
	self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
		self.notebook_0.GetPage(1).GetWin(),
		title="Scope Plot of demodulated signal",
		sample_rate=s_rate,
		v_scale=0,
		v_offset=0,
		t_scale=0.000020,
		ac_couple=False,
		xy_mode=False,
		num_inputs=1,
		)
		self.notebook_0.GetPage(1).Add(self.wxgui_scopesink2_0.win)
	self.wxgui_scopesink2_3 = scopesink2.scope_sink_c(
		self.notebook_0.GetPage(0).GetWin(),
		title="Scope Plot of received signal",
		sample_rate=s_rate,
		v_scale=0,
		v_offset=0,
		t_scale=0.000020,
		ac_couple=False,
		xy_mode=False,
		num_inputs=1,
		)
		self.notebook_0.GetPage(0).Add(self.wxgui_scopesink2_3.win)
```

```
##################################################
# Connections
##################################################
self.connect((self.blks2_rational_resampler_xxx_0, 0), (self.audio_sink_0, 0))
self.connect((self.blks2_packet_decoder_0, 0), (self.blks2_rational_resampler_xxx_0, 0))
self.connect((self.blks2_packet_decoder_0, 0), (self.wxgui_fftsink2_1, 0))
self.connect((self.blks2_packet_decoder_0, 0), (self.wxgui_scopesink2_0, 0))
self.connect((self.blks2_gmsk_demod_0, 0), (self.blks2_packet_decoder_0, 0))
self.connect((self.usrp2_source_xxxx_0, 0), (self.gr_throttle_2, 0))
self.connect((self.gr_throttle_2, 0), (self.blks2_gmsk_demod_0, 0))
self.connect((self.gr_throttle_2, 0), (self.wxgui_scopesink2_3, 0))
self.connect((self.gr_throttle_2, 0), (self.wxgui_fftsink2_2, 0))
self.connect((self.usrp2_source_xxxx_0, 0), (self.wxgui_constellationsink2_0, 0))

        def set_samp_per_sym(self, samp_per_sym):
                self.samp_per_sym = samp_per_sym

        def set_s_rate(self, s_rate):
                self.s_rate = s_rate
                self.wxgui_scopesink2_3.set_sample_rate(self.s_rate)
                self.wxgui_fftsink2_2.set_sample_rate(self.s_rate)
                self.wxgui_fftsink2_1.set_sample_rate(self.s_rate)
                self.wxgui_scopesink2_0.set_sample_rate(self.s_rate)
                self.wxgui_constellationsink2_0.set_sample_rate(self.s_rate)

        def set_interpol(self, interpol):
                self.interpol = interpol

        def set_excess_bw(self, excess_bw):
                self.excess_bw = excess_bw

        def set_decim(self, decim):
                self.decim = decim
                self.usrp2_source_xxxx_0.set_decim(self.decim)
##################################################
# START UP
##################################################
if __name__ == '__main__':
        parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
        (options, args) = parser.parse_args()
        tb = GMSK_using_USRP()
        tb.Run(True)
```

C12 Direct Sequence Spread Spectrum

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: Direct Sequence Spread Spectrum
# Author: Morine Sithole
# Description: Direct Sequence Spread Spectrum
# Generated: Thu Jul 22 14:30:59 2010
##################################################

from gnuradio import blks2
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 numbersink2
from gnuradio.wxgui import scopesink2
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import math
import wx

class DSSS(grc_wxgui.top_block_gui):
        def __init__(self):
        grc_wxgui.top_block_gui.__init__(self, title="Direct Sequence Spread Spectrum")
        _icon_path = "/home/user/.local/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

##################################################
# Variables
##################################################
self.samp_per_sym = samp_per_sym = 2
self.s_rate = s_rate = 5000000
self.noise = noise = 0.050
self.excess_bw = excess_bw = 0.350
self.const = const = 1+1j, 1+0j, 1-1j, 0-1j, -1-1j, -1+0j, -1+1j, 0+1j,

##################################################
# Notebooks
##################################################
self.notebook_0 = wx.Notebook(self.GetWin(), style=wx.NB_TOP)
self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab0")
self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab1")
self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab2")
self.notebook_0.AddPage(grc_wxgui.Panel(self.notebook_0), "tab3")
self.Add(self.notebook_0)

##################################################
# Blocks
##################################################
self.blks2_dxpsk_demod_0 = blks2.dbpsk_demod(
        samples_per_symbol=samp_per_sym,
        excess_bw=0.35,
        costas_alpha=0.175,
        gain_mu=0.175,
        mu=0.5,
        omega_relative_limit=0.005,
        gray_code=True,
        verbose=False,
        log=False,
        )
self.blks2_dxpsk_mod_0 = blks2.dbpsk_mod(
        samples_per_symbol=samp_per_sym,
        excess_bw=excess_bw,
        gray_code=True,
        verbose=False,
        log=False,
        )
   # Bit error rate function
self.blks2_error_rate_0 = grc_blks2.error_rate(
        type='BER',
        win_size=1000000,
        bits_per_symbol=int(math.log(len(const))/math.log(2)),
        )
self.blks2_packet_decoder_0 = grc_blks2.packet_demod_f(grc_blks2.packet_decoder(
        access_code="",
        threshold=-1,
        callback=lambda ok, payload: self.blks2_packet_decoder_0.recv_pkt(ok, payload),
        ),
        )
self.blks2_packet_encoder_0 = grc_blks2.packet_mod_f(grc_blks2.packet_encoder(
        samples_per_symbol=samp_per_sym,
        bits_per_symbol=1,
        access_code="",
        pad_for_usrp=True,
        ),
        payload_length=0,
        )
self.gr_add_xx_0 = gr.add_vcc(1)
self.gr_chunks_to_symbols_xx_0 = gr.chunks_to_symbols_bc((const), 1)
self.gr_float_to_complex_0 = gr.float_to_complex(1)
self.gr_float_to_complex_1 = gr.float_to_complex(1)
self.gr_multiply_xx_0 = gr.multiply_vcc(1)
self.gr_multiply_xx_1 = gr.multiply_vcc(1)
self.gr_noise_source_x_0 = gr.noise_source_c(gr.GR_GAUSSIAN, noise, 42)
  # Throttle block to avoid CPU congestion
self.gr_throttle_0 = gr.throttle(gr.sizeof_float*1, s_rate)
self.gr_throttle_0_0 = gr.throttle(gr.sizeof_float*1, s_rate)
self.gr_throttle_1 = gr.throttle(gr.sizeof_float*1, s_rate)
self.gr_throttle_2 = gr.throttle(gr.sizeof_char*1, s_rate)
self.gr_throttle_3 = gr.throttle(gr.sizeof_char*1, s_rate)
  # LFSR Pseudo random sequence generator
self.gr_glfsr_source_x_0 = gr.glfsr_source_f(4, True, 1, 14)
self.gr_glfsr_source_x_1 = gr.glfsr_source_f(4, True, 1, 14)
self.gr_vector_source_x_0 = gr.vector_source_f(((1,)*30+(-1,)*60), True, 1)
self.gr_vector_source_x_1 = gr.vector_source_f((1,-1,-1,-1,1,1,1,1,-1,-1,-1,1,1,-1,1,), True, 1)
self.gr_vector_source_x_2 = gr.vector_source_f((1,-1,-1,-1,1,1,1,1,-1,-1,-1,1,1,-1,1,), True, 1)
self.gr_vector_source_x_3 = gr.vector_source_f(((1,)*30+(-1,)*60), True, 1)
self.wxgui_fftsink2_0 = fftsink2.fft_sink_f(
        self.notebook_0.GetPage(2).GetWin(),
        baseband_freq=200000,
        y_per_div=10,
        y_divs=10,
        ref_level=5,
        ref_scale=2.0,
        sample_rate=s_rate,
        fft_size=1024,
        fft_rate=30,
        average=True,
        avg_alpha=None,
        title="FFT Plot of demodulated, decoded signal",
        peak_hold=True,
        win=window.blackmanharris,
        )
        self.notebook_0.GetPage(2).Add(self.wxgui_fftsink2_0.win)
self.wxgui_fftsink2_1 = fftsink2.fft_sink_c(
        self.notebook_0.GetPage(0).GetWin(),
        baseband_freq=200000,
        y_per_div=10,
        y_divs=10,
        ref_level=5,
        ref_scale=2.0,
        sample_rate=s_rate*2,
            fft_size=512,
            fft_rate=30,
            average=True,
            avg_alpha=None,
            title="FFT Plot of Modulated Signal",
            peak_hold=True,
            win=window.blackmanharris,
            )
            self.notebook_0.GetPage(0).Add(self.wxgui_fftsink2_1.win)
self.wxgui_fftsink2_2 = fftsink2.fft_sink_c(
            self.notebook_0.GetPage(1).GetWin(),
            baseband_freq=0,
            y_per_div=10,
            y_divs=10,
            ref_level=5,
            ref_scale=2.0,
            sample_rate=s_rate*16,
            fft_size=512,
            fft_rate=30,
            average=True,
            avg_alpha=None,
            title="FFT Plot of spread signal",
            peak_hold=True,
            win=window.blackmanharris,
            )
            self.notebook_0.GetPage(1).Add(self.wxgui_fftsink2_2.win)
    # BER display
self.wxgui_numbersink2_0 = numbersink2.number_sink_f(
            self.notebook_0.GetPage(3).GetWin(),
            unit="%",
            minval=-100,
            maxval=100,
            factor=100.0,
            decimal_places=10,
            ref_level=0,
            sample_rate=s_rate/100,
            number_rate=15,
            average=False,
            avg_alpha=None,
            label="Bit error rate",
            peak_hold=False,
            show_gauge=False,
            )
            self.notebook_0.GetPage(3).Add(self.wxgui_numbersink2_0.win)
self.wxgui_scopesink2_0 = scopesink2.scope_sink_f(
            self.notebook_0.GetPage(2).GetWin(),
            title="Scope Plot of demodulated, decoded signal",
            sample_rate=s_rate,
            v_scale=0,
            v_offset=0,
            t_scale=0,
            ac_couple=False,
            xy_mode=False,
            num_inputs=1,
            )
            self.notebook_0.GetPage(2).Add(self.wxgui_scopesink2_0.win)
self.wxgui_scopesink2_1 = scopesink2.scope_sink_c(
            self.notebook_0.GetPage(0).GetWin(),
            title="Scope Plot of Modulated Signal",
            sample_rate=s_rate,
			v_scale=0,
			v_offset=0,
			t_scale=0,
			ac_couple=False,
			xy_mode=False,
			num_inputs=1,
			)
			self.notebook_0.GetPage(0).Add(self.wxgui_scopesink2_1.win)
		self.wxgui_scopesink2_2 = scopesink2.scope_sink_c(
			self.notebook_0.GetPage(1).GetWin(),
			title="Scope Plot of spread signal",
			sample_rate=s_rate,
			v_scale=0,
			v_offset=0,
			t_scale=0,
			ac_couple=False,
			xy_mode=False,
			num_inputs=1,
			)
			self.notebook_0.GetPage(1).Add(self.wxgui_scopesink2_2.win)
		self.wxgui_scopesink2_3 = scopesink2.scope_sink_c(
			self.notebook_0.GetPage(3).GetWin(),
			title="Constellation of demodulated signal",
			sample_rate=s_rate,
			v_scale=0,
			v_offset=0,
			t_scale=0,
			ac_couple=False,
			xy_mode=True,
			num_inputs=1,
			)
			self.notebook_0.GetPage(3).Add(self.wxgui_scopesink2_3.win)
		self.wxgui_scopesink2_4 = scopesink2.scope_sink_c(
			self.notebook_0.GetPage(3).GetWin(),
			title="Constellation of noisy channel",
			sample_rate=s_rate,
			v_scale=0,
			v_offset=0,
			t_scale=0,
			ac_couple=False,
			xy_mode=True,
			num_inputs=1,
			)
			self.notebook_0.GetPage(3).Add(self.wxgui_scopesink2_4.win)

		##################################################
		# Connections
		##################################################
		self.connect((self.blks2_dxpsk_demod_0, 0), (self.blks2_packet_decoder_0, 0))
		self.connect((self.gr_throttle_0, 0), (self.gr_float_to_complex_0, 0))
		self.connect((self.gr_float_to_complex_0, 0), (self.gr_multiply_xx_0, 1))
		self.connect((self.blks2_dxpsk_mod_0, 0), (self.gr_multiply_xx_0, 0))
		self.connect((self.gr_multiply_xx_0, 0), (self.wxgui_scopesink2_2, 0))
		self.connect((self.gr_throttle_1, 0), (self.gr_float_to_complex_1, 0))
		self.connect((self.gr_float_to_complex_1, 0), (self.gr_multiply_xx_1, 1))
		self.connect((self.gr_vector_source_x_1, 0), (self.gr_throttle_0, 0))
		self.connect((self.gr_glfsr_source_x_0, 0), (self.gr_throttle_0, 0))
		self.connect((self.gr_vector_source_x_2, 0), (self.gr_throttle_1, 0))
		self.connect((self.gr_glfsr_source_x_1, 0), (self.gr_throttle_1, 0))
		self.connect((self.gr_multiply_xx_0, 0), (self.gr_add_xx_0, 0))
```

```
self.connect((self.gr_noise_source_x_0, 0), (self.gr_add_xx_0, 1))
self.connect((self.gr_add_xx_0, 0), (self.gr_multiply_xx_1, 0))
self.connect((self.blks2_error_rate_0, 0), (self.wxgui_numbersink2_0, 0))
self.connect((self.gr_vector_source_x_0, 0), (self.gr_throttle_0_0, 0))
self.connect((self.gr_chunks_to_symbols_xx_0, 0), (self.wxgui_scopesink2_3, 0))
self.connect((self.gr_multiply_xx_1, 0), (self.blks2_dxpsk_demod_0, 0))
self.connect((self.blks2_dxpsk_mod_0, 0), (self.wxgui_scopesink2_1, 0))
self.connect((self.blks2_dxpsk_mod_0, 0), (self.wxgui_fftsink2_1, 0))
self.connect((self.blks2_packet_encoder_0, 0), (self.blks2_dxpsk_mod_0, 0))
self.connect((self.gr_multiply_xx_0, 0), (self.wxgui_fftsink2_2, 0))
self.connect((self.gr_vector_source_x_3, 0), (self.gr_throttle_2, 0))
self.connect((self.gr_throttle_2, 0), (self.blks2_error_rate_0, 0))
self.connect((self.gr_throttle_0_0, 0), (self.blks2_packet_encoder_0, 0))
self.connect((self.gr_add_xx_0, 0), (self.wxgui_scopesink2_4, 0))
self.connect((self.blks2_dxpsk_demod_0, 0), (self.gr_chunks_to_symbols_xx_0, 0))
self.connect((self.gr_throttle_3, 0), (self.blks2_error_rate_0, 1))
self.connect((self.blks2_dxpsk_demod_0, 0), (self.gr_throttle_3, 0))
self.connect((self.blks2_packet_decoder_0, 0), (self.wxgui_scopesink2_0, 0))
self.connect((self.blks2_packet_decoder_0, 0), (self.wxgui_fftsink2_0, 0))

        def set_samp_per_sym(self, samp_per_sym):
                self.samp_per_sym = samp_per_sym

        def set_s_rate(self, s_rate):
                self.s_rate = s_rate
                self.wxgui_scopesink2_1.set_sample_rate(self.s_rate)
                self.wxgui_scopesink2_2.set_sample_rate(self.s_rate)
                self.wxgui_scopesink2_3.set_sample_rate(self.s_rate)
                self.wxgui_scopesink2_4.set_sample_rate(self.s_rate)
                self.wxgui_scopesink2_0.set_sample_rate(self.s_rate)
                self.wxgui_fftsink2_1.set_sample_rate(self.s_rate*2)
                self.wxgui_fftsink2_2.set_sample_rate(self.s_rate*16)
                self.wxgui_fftsink2_0.set_sample_rate(self.s_rate)

        def set_noise(self, noise):
                self.noise = noise
                self.gr_noise_source_x_0.set_amplitude(self.noise)

        def set_excess_bw(self, excess_bw):
                self.excess_bw = excess_bw

        def set_const(self, const):
                self.const = const
##################################################
# start up code
##################################################
if __name__ == '__main__':
        parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
        (options, args) = parser.parse_args()
        tb = DSSS()
        tb.Run(True)
```

Appendix D

Laboratory Exercise Documentation

D1 Objectives

The objectives of this laboratory exercise are

- to learn the digital signal processing methods used in Software Defined Radio. A software defined radio is a device implemented partially or entirely using software.
- to use the GNU Radio Companion GUI to simulate digital signal processing algorithms
- to gain experience in PYTHON programming
- to transmit and receive voice and data over the 2.4 - 2.483 GHz ISM band using various modulation and demodulation techniques.
- to evaluate the system performance of an SDR system.

D2 The Equipment

The laboratory consists of two sets of standard PCs each connected to a USRP2 (Universal Software Radio Peripheral) via Gigabit Ethernet. The USRP2 houses a Xilinx Spartan 3-2000 FPGA, two 16-bit digital-to-analogue converters and two 14-bit analogue-to-digital converters. The RFX2400 transceiver mounted on the USRP2 motherboard is the RF frontend operating in the frequency range 2.4 - 2.9 GHz. The RFX2400 transmits and receives via a VERT2450 tri-band antenna. One of the PCs has a sound card and a set of loud speakers for signal output. The hardware is used with the GNU Radio software, which provides the development environment for creating software radios. The GNU Radio software runs on Linux Mandriva operating system. GNU Radio has a GUI called GNU Radio Companion (GRC) which is used to create PYTHON flow graphs. Digital Signal processing functions, implemented as blocks of C++ programs, are linked together and callable from PYTHON scripts.

Note: To avoid the fuse on the USRP2 motherboard from blowing up, ensure the power supply is off when plugging the RFX2400 transceiver in and out.

D3 Creating and running a flow graph in GNU Radio

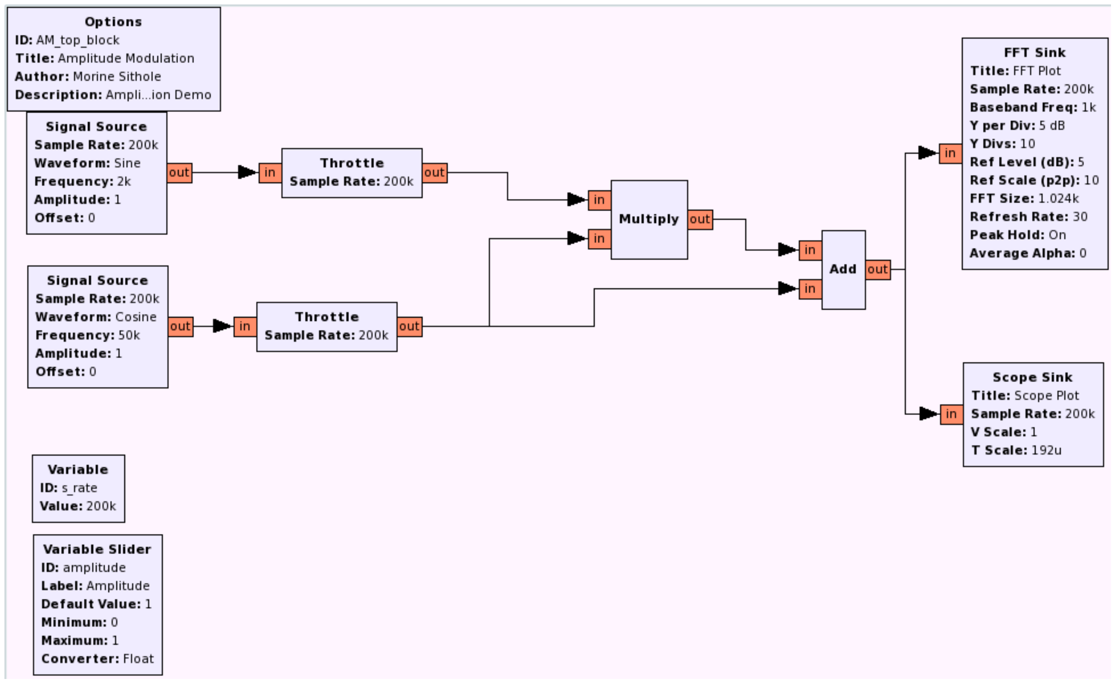
Open the GRC application through the Mandriva application launcher menu:

Application Launcher menu →Development →GRC

Create the flow graph shown in Figure D1. The flow graph is an implementation of the basic equation which defines an amplitude modulated signal:

$$s(t)=A_c\left(1+h\,m(t)\right)\cos(2\pi f_c t)$$

where *s(t)* is the modulated signal, A_c is the carrier amplitude, f_c is the carrier frequency, *h* is the modulation index and *m(t)* is the modulating signal.



Options
ID: AM_top_block
Title: Amplitude Modulation
Author: Morine Sithole
Description: Ampli...ion Demo

Signal Source
Sample Rate: 200k
Waveform: Sine
Frequency: 2k
Amplitude: 1
Offset: 0

Signal Source
Sample Rate: 200k
Waveform: Cosine
Frequency: 50k
Amplitude: 1
Offset: 0

Throttle
Sample Rate: 200k

Throttle
Sample Rate: 200k

Multiply

Add

FFT Sink
Title: FFT Plot
Sample Rate: 200k
Baseband Freq: 1k
Y per Div: 5 dB
Y Divs: 10
Ref Level (dB): 5
Ref Scale (p2p): 10
FFT Size: 1.024k
Refresh Rate: 30
Peak Hold: On
Average Alpha: 0

Scope Sink
Title: Scope Plot
Sample Rate: 200k
V Scale: 1
T Scale: 192u

Variable
ID: s_rate
Value: 200k

Variable Slider
ID: amplitude
Label: Amplitude
Default Value: 1
Minimum: 0
Maximum: 1
Converter: Float

Figure D1-Screenshot of the Amplitude Modulation Signal Flow Diagram

The modulating signal is a sine waveform of 2 kHz and the carrier is 50 kHz. The sampling rate is 200 kHz. To add a block to the flow graph, click on the triangle next to the block's group in the "Signal Blocks" tab of the block selection window. A list of the blocks in the group will be displayed. Drag and drop the required block from the block selection window into the main window. To connect blocks, click on the output port of one block then click on the input port of the target block. To delete a connection, click on the connection to highlight it then press the 'delete' keyboard button. To edit the parameters of a block, right click it, then select 'edit' from the drop-down menu.

The throttle blocks in figure D1 are required in the flow graphs to buffer the rate of streaming data and to prevent high CPU utilization during runtime. The signal source blocks and graphical display blocks must have the same sampling rate at which they are sending or receiving data. In GRC, the sampling rate is defined by the parameters of the graphical displays used in the flow graph. To save the flow graph, click on the File menu, and click Save. Save the file in your home directory. A python program is generated and saved in the same location where the GRC flow graph was saved. To run the flow graph, click the 'Green Gear' toolbar button. Observe the amplitude modulated signal in the time domain on the Scope sink. Observe the amplitude modulated signal in the frequency domain on the FFT sink. To stop the program, click the red 'Stop' toolbar button.

D4 Exercises

Using the information acquired in the above preparatory exercise carry out the following tasks.

1. Adjust the sampling rate in the flow graph you created to a value less than twice the frequency of the modulating signal and observe the resulting waveform in the time and frequency domains.
2. Add a low pass filter to the flow graph you created in section D3 to obtain the lower sideband of the amplitude modulated signal. Observe the resulting waveform in the time and frequency domains.
3. The two main classes of digital filters are
   - Finite impulse response (FIR) filters and
   - Infinite impulse response (IIR) filters

The impulse response of an FIR filter is of finite duration, while the impulse response of an IIR filter is of infinite duration. Filters are used in digital communications systems to separate signals that have been mixed for transmission over the same communication channel and to reduce the effects of noise on the desired signal. Generate a 1 kHz square wave and pass it through a 3-tap finite impulse response (FIR) filter and a 3-tap infinite impulse (IIR) filter in parallel. Observe the time domain signal and spectrum at each stage of the flow graph. Determine the difference between the IIR filter and the FIR filter.

4. Implement the following modulation schemes for the binary sequence 000000011100001110001111111000111:
    i. Binary Amplitude Shift Keying
    ii. Non-coherent Binary Frequency Shift Keying
    iii. Binary Phase Shift Keying
    iv. Four-Level Amplitude Shift Keying
    v. Continuous Phase Frequency Shift Keying. Use the VCO (Voltage Controlled Oscillator) block or the CPFSK (Continuous Phase Frequency Shift Keying) block. What are the differences between Non-coherent BFSK and CPFSK?
5. Design a DBPSK transmitter and receiver to transmit a 1 kHz sinusoidal signal from one host PC to the other over a carrier of 2.44 GHz.
6. Design a DBPSK transmitter and receiver to transmit a text file from one PC to another PC over a carrier of 2.44 GHz.
7. Design a GMSK transmitter and receiver to transmit the recorded voice signal 408.wav from one host PC to a second PC. The 408.wav file is located in the /home/user directory. The data rate of the 408.wav file is 128 kbps. The signal must be transmitted over a carrier frequency of 2.6 GHz.
8. Simulate a DSSS (Direct Sequence Spread Spectrum) system on one PC. Measure the bit error rate for various SNR (Signal to Noise Ratios) for an AWGN channel model.

D5 References

1. REED, J. H., (2002), Software Radio: a modern approach to radio engineering, Prentice Hall, New Jersey, USA, ISBN 978-0-13-081158-5.
2. http://www.ettus.com
3. http://www.gnuradio.org
4. http://www.python.org/
5. HAYKIN, S., (2001), Communications Systems, 4th Edition, John Wiley & Sons, New Jersey, ISBN 0-471-17869-1.
6. PEEBLES, P. Z., (1987), Digital Communications Systems, Prentice Hall, Englewood Cliffs, New Jersey, USA, ISBN 0-13-211970-6.
7. PROAKIS, J. D., (1989), Digital Communications, 2nd Edition, McGraw Hill, Singapore, ISBN 0-07-100269-3.

8. PROAKIS, J. G., and MANOLAKIS, D. G., (1996), Digital Signal Processing: Principles, Algorithms and Applications, 3rd Edition, Prentice Hall, Upper Saddle River, New Jersey, USA, ISBN 978-0-13-394289-7.

Appendix E

Project Gantt chart

Project Activity	Start date	End date	2010						Status
			March	April	May	June	July	August	
Literature review	01-Mar	26-Apr							Complete
Project Plan	15-Mar	15-Mar							Complete
Risk Assessment	18-Mar	18-Mar							Complete
Equipment Selection and purchase	22-Mar	02-Apr							Complete
Assembly of equipment	05-Apr	09-Apr							Complete
Function Design and verification									
· Sampling theorem(Simulation and testing)	10-Apr	17-April							Complete
· FFT spectral analysis (Simulation, debugging and testing)	17-April	24-May							Complete
· Digital filtering, encoding and decoding (Simulation, debugging and testing)	24-May	31-May							Complete
· ASK, FSK and PSK modulation techniques (Simulation, debugging and testing)	31-May	06-Jun							Complete
· DBPSK, DQPSK, GMSK modulation and demodulation (Simulation, debugging and testing)	06-Jun	13-Jun							Complete
· Matched Filtering, BER and SNR (Simulation, debugging and testing)	13-Jun	20-Jun							Complete
· Direct sequence spread spectrum (Simulation, debugging and testing)	20-Jun	27-Jun							Complete
· Additive white Gaussian noise (AWGN) channel (Simulation, debugging and testing)	27-Jun	04-Jul							Complete
Combine overall code into overall model and test	04-Jul	11-Jul							Complete
Prepare Documentation for the lab exercises	01-Jul	31-Aug							Complete
Project Dissertation	01-Jul	31-Aug							Complete