USRP2 IMPLEMENTATION OF COMPRESSIVE SENSING BASED CHANNEL ESTIMATION IN OFDM

A Thesis
Presented to
the Faculty of the Electrical and Computer Engineering Department
University of Houston

in Partial Fulfillment
of the Requirements for the Degree
Master of Science
in Electrical Engineering

by

Tina Jayce Mathews

December 2012

USRP2 IMPLEMENTATION OF COMPRESSIVE SENSING
BASED CHANNEL ESTIMATION IN OFDM

Tina Jayce Mathews

Approved:

Chair of the Committee
Dr. Zhu Han, Associate Professor
Electrical and Computer Engineering

Committee Members:

Dr. Rong Zheng, Associate Professor
Electrical and Computer Engineering

Dr. E. Joe Charlson, Professor
Electrical and Computer Engineering

Dr. Suresh K. Khator, Associate Dean,
Cullen College of Engineering

Dr. Badrinath Roysam, Professor and Chairman,
Electrical and Computer Engineering

Acknowledgements

I am grateful to God, who always has better plans for me than anyone else. I have to thank Mummy for giving me the confidence to get back into academia and supported me everyday. I have to thank Daddy for instilling the love of science and the light of knowledge as a child and showing me the importance of being sincere in all your endeavours, however small or big it may be. Most importantly, I have to thank my husband, Jayce Mathews who held me through thick and thin, fall and spring, ups and downs, and who reiterated the importance of a good education. I also would like to thank my sisters and my in-laws for praying for me and believing in me.

I owe the deepest gratitude to my advisor, Dr. Zhu Han who accepted me as his student and supported me all throughout the thesis with his patience and knowledge. I am thankful for his constant efforts to keep me focussed and also allowing me to think on my own and approach the problem in different directions. I cherish the opportunity to have done research under him and learning from him.

I would also like to thank Dr. E. J. Charlson and Dr. Rong Zheng for being a part of my thesis committee. I have to thank Dr. Zheng for accommodating me in the Wiser Lab. I have to mention the faculty in COT for supporting me financially during my research semesters. I would like to thank my friends, my colleagues in the Wireless Networking Lab in the ECE department and also Wiser Lab in the Computer Science department, especially Mr. Guanbo Zheng with whom I had numerous discussions about the wireless platform and accelerated my learning curve to get familiarized with the USRP2 platform. This was a time when being in a research group provided me with a platform to interact with various researchers and learn from them.

In every big and small way that everyone has touched me in the past two years, I am grateful for everything you have given me.

USRP2 IMPLEMENTATION OF COMPRESSIVE SENSING BASED CHANNEL ESTIMATION IN OFDM

An Abstract
of a
Thesis
Presented to
the Faculty of the Electrical and Computer Engineering Department
University of Houston

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in Electrical Engineering

by
Tina Jayce Mathews

December 2012

Abstract

Radio channel impairment is a major concern in any wireless system. Channel estimation is performed at the receiver to obtain the channel response in order to calculate the multipath channel effects. However, the traditional way of using pilots for channel estimation has a tradeoff between spectral efficiency and estimation accuracy. An increasing amount of research is being done on a novel signal processing technique called compressive sensing and its applications in the modern day wireless systems for channel estimation. In this thesis, we can exploit the sparsity of the time domain channel by choosing the pilots randomly and building a random projection measurement matrix. This approach improves the channel estimation accuracy by conserving the bandwidth. This thesis investigates various modulation schemes in an open source software based radio development kit, GNU Radio for a wireless system and build a compressed sensing based channel estimator for the OFDM module on a Universal Software Radio Peripheral 2 (USRP2). Simulations for compressed channel sensing are conducted to prove the effectiveness over traditional channel estimation. The time domain based compressed channel estimator is implemented as a signal processing package in GNU Radio, and performance studies are done in the real-time system.

Table of Contents

Acknowledgements **v**

Abstract **vii**

Table of Contents **viii**

List of Figures **xi**

List of Algorithms **xiv**

1 Introduction **1**

1.1 Channel Estimation in Wireless Channels 2

1.2 Contributions of This Thesis 4

1.3 Organization of This Thesis 5

2 USRP2 and GNU Radio **7**

2.1 Hardware Platform: Universal Software Radio (USRP) 7

2.2 Software Platform: GNU Radio 8

2.3 The ‘Hello World’ in the Experimental Setup 10

3 Preliminary Work: OFDM Architecture on USRP2 **14**

3.1 Introduction to OFDM 14

3.2 Existing OFDM Architecture And Implementation On USRP2 15

3.2.1 OFDM Tx Baseband Model 16

3.2.2 OFDM RX Baseband Model 17

3.2.3 Receiver Synchronization 19

3.2.4 Classic Channel Estimation 23

4 CS-OFDM Architecture And Implementation 25

4.1 Compressive Sensing 25

4.2 Proposed Architecture 26

4.2.1 How CS Fits In? 26

4.2.2 CS System Model 28

4.2.3 Bandwidth Utilization 30

4.3 Soft Radio Implementation 32

4.3.1 Approach Used for Implementing CS 32

4.3.2 Proposed System Design 33

4.3.3 YALL1 for ℓ_1 Optimization 36

4.3.4 ℓ_2 Optimization 37

4.3.5 CS OFDM Receiver Design 39

4.3.6 Software Stack 40

5 Simulations and Discussions 43

5.1 Simulations 43

5.2 System Characterization 45

5.3 Experimental Setup 47

6 Conclusions and Future Work 51

6.1 Challenges 51

6.2 Future Work on CS-OFDM 52

6.2.1 OFDM SNR Estimators 52

6.2.2 FPGA Implementations 53

6.3 Summary and Conclusions 54

Bibliography **56**

List of Figures

1.1 Wireless Channel Model 4

2.1 USRP2 Motherboard 8

2.2 USRP MotherBoard Architecture 9

2.3 USRP Tx Flow 9

2.4 USRP Rx Flow 9

2.5 GNU Radio Architecture 10

2.6 System Model of FM Receiver using GRC in GNU Radio 11

2.7 Python code snippet in GNU Radio 12

3.1 OFDM System 15

3.2 Effects From Deep Fading 15

3.3 Effects From Delay Spread 16

3.4 OFDM TX Flow Diagram 16

3.5 OFDM TX Flowgraph ofdm_tx.py 17

3.6 OFDM RX 18

3.7 OFDM RX Flowgraph ofdm_rx.py 19

3.8 S & C Model 20

3.9 Receiver Synchronizer 20

3.10 Timing Metric from Received Samples 21

3.11 Detection of Preambles by Matched Filter 22

3.12 Arrangement of Pilots 23

3.13 Channel Estimation 24

4.1 Compressive Sensing 25

4.2 MSE vs SNR for CS Algorithm 29

4.3 Compressive Sensing Flowchart 31

4.4 Bandwidth Analysis 32

4.5 Ideal FFT Bins and Shifted FFT Bins due to Frequency Offset 33

4.6 Top View for CS Time Equalizer 34

4.7 Comparison of FFTW3 and GSL Libraries 35

4.8 Block Diagram for CS Time Equalizer 36

4.9 Complex SVD Solution in GSL 39

4.10 OFDM Rx with CS Estimation Flowgraph ofdm_receiver_cs.py 40

4.11 Software Stack of the CS Receiver in GNU Radio 42

5.1 USRP2 Setup 44

5.2 Simulations for Symmetrical CS Setup 44

5.3 Simulations for Asymmetrical CS Setup 45

5.4 Simulations for CS Setup with Custom ℓ_1 Optimization 46

5.5 Images Obtained with the Setup at Different Tx Gains 46

5.6 USRP2 Receiver Packet Dump 48

5.7 Sparse Channel Estimates With Symbol Numbers 48

5.8 Sparse Channel Estimates for Preamble Symbols with Transfer Functions from USRP2 49

5.9 Symbol Output from cs_cpp_sigmix Block 49

5.10 CS Estimated BER vs SNR plot 50

List of Algorithms

4.1 Theoretical CS OFDM 30

4.2 Implemented CS OFDM 39

Chapter 1

Introduction

Wireless communication has been ubiquitous and continues to be a rapidly growing form of communication. Although optical networks and wired networks can reach higher rates, the ease of adding any portable device in a network has increased the popularity of wireless communication. This has definitely paved way to develop technologies to obtain higher data rates and secure information channels for better information quality. New standards and technologies have helped wireless networks to be a replacement in home, work and in industries. There are various applications that make use of the wireless channel like cell phone voice communication, data transfer including high-speed video transfer, sensor networks etc. Moreover wireless technology is used for a wide variety of applications ranging from a gate opener or a cordless telephone to a sophisticated GPS based tracking system or satellite television. Moreover with the increasing use of the internet, Wi-Fi is used to connect to the internet using 802.11a/b/g/n bridging the gap between wireless networks and the information backbone. In fact, diversity techniques in both time and space have helped us to improve the wireless link at a low cost. However, there are many technical challenges involved in using wireless channels. Apart from the fundamental capacity limits of wireless channels and designing specific circuitry and systems for a wireless platform, we need to characterize the channel. Wireless channels are time varying and undergo path loss for different environments.

Wireless channels can exhibit flat channel or frequency-selective properties of multipath fading. Similarly the signal undergoes doppler spread along with deep fading if it is in motion. These are channel impairments which must be taken into account at the receiver while retrieving a signal. The fading can be large-scale fading or small-scale fading. Large-scale fading occurs due to large buildings or topological changes. Small-scale changes happen due to the constructive and destructive interference of the signals from the transmitter and the receiver [1]. We use certain physical or statistic models of the channel variation over time and over frequency which are used in [2,3]. There are two major parameters that interests us while estimating a wireless channel. They are doppler spread which gives the coherence time and the delay spread which gives the coherence bandwidth. Depending on how a channel changes with respect to the coherence time, it is categorized as a slow fading and a fast fading channel. The coherence bandwidth is the reciprocal of the multipath delay spread. If the bandwidth of the input signal is less than the coherence bandwidth of the channel, it becomes a flat fading channel. This means the channel only has a single tap. If the bandwidth is much larger, it becomes a frequency selective channel and needs multiple taps to represent it. Similarly the channel can also undergo slow fading or log normal fading which is caused by the blockages on the ground. This is independent of path loss. Hence the wireless channel is most commonly modeled as a linear time varying system and it is known as the channel impulse response (CIR). This tells us about the path attenuation as well the delay that each of the multipaths undergo in a channel. In every receiver, these parameters are estimated to equalize the channel.

This gives us an insight into the importance of obtaining the characteristics of a wireless channel in real time time. In a fixed network, the channel is studied in the beginning with a training sequence and this does not change in the rest of the transmission. Since the channel changes in wireless network, a good channel estimation algorithm decides the quality and the quantity of information transmitted in a network.

1.1 Channel Estimation in Wireless Channels

In any wireless system, a transmitted signal undergoes reflection, diffraction or scattering, and reaches the receiver as an attenuated and delayed version of the original signal. In order to estimate these nonlinear effects, channel estimation is done at the receiver of the wireless system. Orthogonal Frequency Division Multiplexing (OFDM) is widely used in most modern day wireless systems making it attractive in applications like 802.11a (WLAN), IEEE 802.16 (WiMAX) and LTE. Channel estimation becomes a complex two dimensional problem in OFDM. However, increasing complexity of the existing systems and the demand for higher transmission rates have pushed the requirements of estimating the channel to a higher accuracy. This thesis aims at bridging an existing signal processing technique, compressive sensing to estimate the channel impulse response in a practical OFDM system.

Compressive Sensing [4] has gained increasing transparency in many diverse signal processing applications. Since we are always looking to get more out of the less in a communication system, CS is a promising technique. We aim at using CS in estimating the channel in an OFDM wireless system where the channel state information is noisy and time-varying. Various approaches have been made in obtaining the channel response owing to the increased requirement for more accuracy within a shorter time. Channel estimation is done either using a combination of pilots or a training sequence [5] or by blind estimation [6]. Here the arrangement of the pilots are known to both the transmitter and the receiver. The design of a block type estimator has low complexity however it does not track the channel as good as a cob-type estimator. This also works which aim at design error reducing pilots like in [33]. However, using pilots can bring a tradeoff between spectral efficiency and estimation accuracy. So an increasing use of data-aided channel estimation is found in [10, 11] with joint data and channel recovery techniques. However, [7] has shown how CS can improve the spectral efficiency in the training based method [29] shows how compressive sensing can be used in wideband systems. However, there has been a lot of research on using compressive in practical systems. The increased use of CS in wireless systems is seen in [8, 9] , which motivates us to use CS models in a real time system. Moreover [12, 13] works have motivated us to use CS in a real time system and prove the architecture. So any wireless channel can be modeled as Rayleigh or a Rician channel. The Rician model is a more closer model since it takes the Line-of-Sight into account. Fig. 1.1 shows how a rayleigh channel looks into across various frequency bins that is varying with time. Our main aim is to estimate such a channel in a practical wireless system.

Apart from using compressive sensing in our model, we also look into the importance of shifting estimation from the frequency domain to the time domain. Most of the wireless standards use pilots for tracking the OFDM system where the estimation is done in the frequency domain. Since there are variations in the time and the frequency domain, the ideal channel estimator filter would be a 2-D Wiener filter. The papers [30, 34] looks at how time domain channel estimation can be done using the traditional zero forcing estimators, unbiased Least Squares(LS) or Minimum

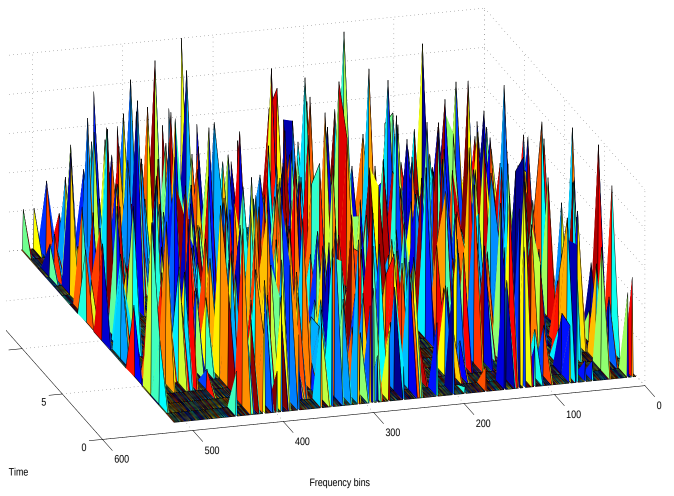


Figure 1.1: Wireless Channel Model

Mean Squares(MMSE) estimators. It also cites the improvements in performance when estimation is done in time domain. However, this is at the cost of complexity.

1.2 Contributions of This Thesis

We have investigated the existing Orthogonal Frequency Division Multiplexing (OFDM) implementation in the current hardware setup and implemented a compressed sensing based channel estimation model. There are several hardware platforms like WARP, HPSDR, FLEX3000, LYRTECH and wireless chipsets which helps us in bridging the gap between the simulations and the hardware prototype. Rather than using traditional chipsets, the Software Defined Radio (SDR) has become increasingly popular due to its cost effectiveness, capability and versatility. The Industrial, Scientific and Medical (ISM) band at 2.4GHz can be used for developing many protocols as well as physical layer architectures in a real-time environment. We chose Universal Software Radio Peripheral (USRP2) platform for our experiments which is available with an open source software package named GNU Radio. Using GNU Radio is far more economical than building an expensive dedicated hardware.

- In this thesis, we have tried to bridge the gap between simulations and a hardware prototype by proving a concept in software radio which is more resource intensive. CS systems introduce a tradeoff between complexity and accuracy. Moreover, we also had to use the resources that were available on the chosen platform. We have performed our studies on the RF daughter board XCVR2450 with the USRP2.

- This work has adopted CS based algorithms in a realistic OFDM framework and significantly improved the OFDM channel estimation technique. The use of CS in OFDM is cited in [15, 17], however this work has implemented CS directly on a hardware platform.

- We have also made compressive sensing and other related modules more accessible as a plug in module for software radio implementations for other implementations outside OFDM for various applications where GNU Radio can be used. ℓ_1 and ℓ_2 models have been built which can be reused.

1.3 Organization of This Thesis

In this thesis, the problem of effective channel estimation is practical OFDM systems are analyzed and a solution based on CS is formulated. We designed a stable and physically recognizable CS structure and obtained the CS estimates to prove the sparsity of the channel.

In Chapter 2, we discuss about the experimental platform used in the thesis. The growing emergence of software radio systems and the ease of implementing practical systems have motivated us to use such a platform. We look into the specifications that the hardware platform, USRP2 has to offer and the the radio software development kit, GNU Radio.

In Chapter 3, orthogonal frequency based multiplexing is discussed and literature review is done on the transmitter and the receiver design. We also look into the receiver synchronizer model used in the current hardware setup in Section 3.2.3. Various models for channel estimation are discussed in Section 3.2.4 and the specifications of this model is discussed.

In Chapter 4, a brief overview of CS is provided along with how CS can be used in current setup. Section 4.2.1 explains how the sparsity of a channel can be exploited in the current scenario and estimation can be done with fewer measurements. We define a system model in Section 4.2.2 and look into the detailed design and implementation in the following sections. This chapter talks about the entire implementation done in the USRP2 platform.

In Chapter 5, we discuss the preliminary studies and the simulations done for an OFDM system with compressive sensing channel estimation in MATLAB. Section 5.2 describes the system characterization done for the experimental setup to check the feasibility of the platform and finally we explain the results gathered from the implemented CS setup.

In Chapter 6, we discuss the challenges that this thesis offered and how they were curbed. We finally make valuable pointers in Section 6.3 to conclude the thesis.

Chapter 2

USRP2 and GNU Radio

We chose a software radio to build the CS based channel estimation in OFDM. While choosing a platform to built a practical system, we considered two paths. The first path was to build a digital design system where compressive sensing can be used as an analog to information processing block. However, we would only be limited to proving the algorithm and not extending its application to a communication system. Otherwise, we would have to look for traditional chipsets with supported daughtercards which would be expensive and application specific. We were looking for something more flexible with a general purpose processor and basic hardware where reusability and reconfigurability were the required traits. So the software radio fitted our requirement well. The work in [44] clearly states how the resulting software-defined radio (or software radio) extends the evolution of programmable hardware, increasing flexibility via increased programmability. This would help us prove our algorithm in the most closest practical system. An early version of the software defined radio system is seen in [45] which gives an overview of the prototype.

2.1 Hardware Platform: Universal Software Radio (USRP)

In order to prove CS estimation in a real time system, we use a software defined radio. The hardware platform to implement software radio is the USRP2 that is provided by Ettus Research [19]. The software defined radio used is a GNU Radio [20]. There are two versions of USRP: USRP1 and USRP2. These line of products helps the academic, scientific, and industrial community to perform experiments on RF from DC to 6GHz.

The USRP2 helps to communicate between the host computer and the analog intermediate frequency signal. It has a general purpose processor along with a FPGA. The FPGA is used for upconversion and downconversion. The FPGA includes digital down converters (DDC) implemented with cascaded integrator-comb (CIC) filters (for receivers) and interpolators for transmitters. Digi-

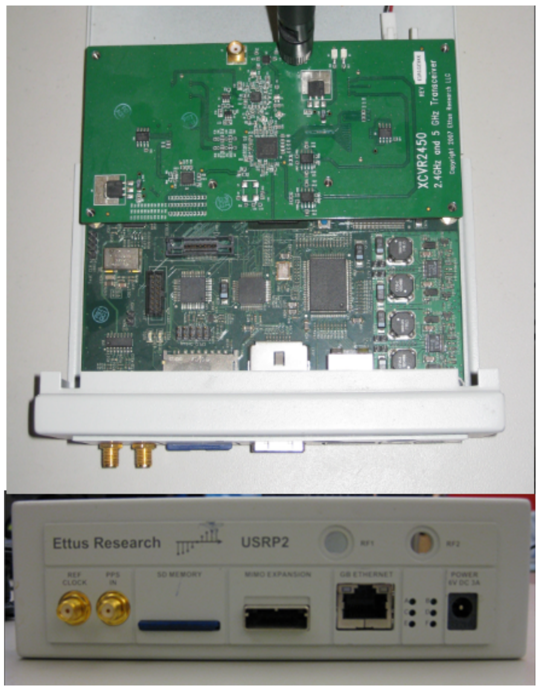


Figure 2.1: USRP2 Motherboard

tal up converters (DUCs) on the transmit side are actually contained in the AD9862 CODEC chips. The USRP2 has a 100 MSs, 14 bit, ADC for RX and 400 MSs, 16 bit, DAC for TX. It has slots to place interchangeable RF cards. The main daughter boards used in the current setup are Dual band Transceiver XCVR 2450 with ranges 2.3 - 2.9 GHz & 5GHz & RFX2400 with 2.3 - 2.9 GHz. The setup is shown in Fig. 2.1 and uses Gigabit Ethernet to communicate to the host. All the basic tasks like interpolation, decimation, upconversion, and downconverion are done in the FPGA of the USRP2 board.

2.2 Software Platform: GNU Radio

The software defined radio used is GNU Radio, which is a free and open-source software development toolkit and is highly compatible with the USRP2 peripheral. The GNU Radio has signal processing blocks area available to implement software radios. So typically all the baseband processing is done in the GNU Radio. We currently use GNU Radio 3.3.0 for our setup. The flow graph for reception and transmission is shown in Fig. 2.4 and Fig. 2.3. The reception path consists

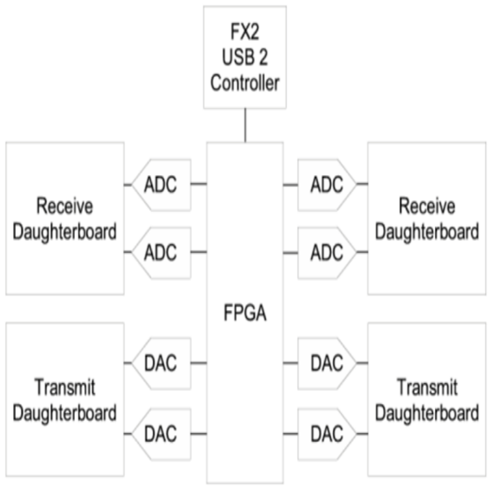


Figure 2.2: USRP MotherBoard Architecture

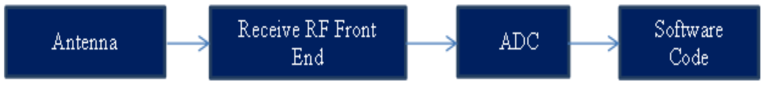


Figure 2.3: USRP Tx Flow

of the antenna, an RF Down converter and then the ADC which digitizes the signal. From the ADC to goes to the USRP2 motherboard which has an FPGA, this is responsible for data rate conversion and timing. The gigabit ethernet (GbE) interface is from the USRP2 motherboard and connects the USRP2 to the host. The transmission flow is in the reverse manner, except that it uses the DAC instead of the ADC.

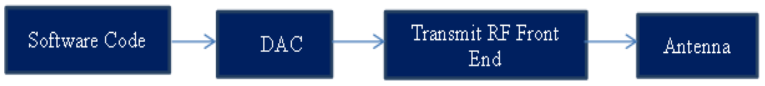


Figure 2.4: USRP Rx Flow

GNU Radio is an open source and has a Python based architecture for building SDR projects. Performance-critical modules are written in C++ and Python is used to glue the modules and also noncritical blocks. SWIG is used for patching all of them in the software defined radio. In Python, we can describe the signal flow from the source to the sink as shown in Fig. 2.5. All the signal processing functions like filtering, modulation, demodulation, encoding are done in the host.GNU Radio gives us a platform to build these functions are C++ routines which can be called by a

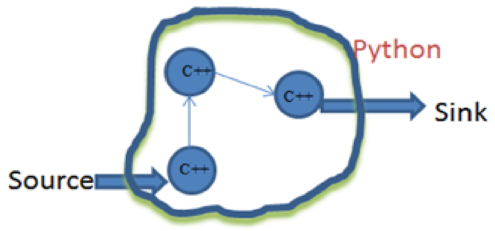


Figure 2.5: GNU Radio Architecture

python script. The intermediate wrapper for the C++ modules is done in SWIG which creates a MAGIC_BLOCK to link the python script to the C++ modules. This is possible due to the smart pointer from the boost libraries that encapsulates the C++ modules. There is also a drag and drop model which helps in simulating the system. This is a open-source Visual programming language for signal processing using the GNU Radio libraries called GNU Radio Companion (GRC) [21]. GRC can also generate the source codes for theses blocks and this can help us in tweaking the setup. The structure of the GNU Radio has a hierarchical approach. However we might only see the signal processing modules as black boxes and need to customize for our requirements. The GRC blocks are described in XML format and can be build with much ease. Fig. 2.6 shows us the system model of a narrow band FM receiver done using USRP2. We can configure a source file and add blocks of interest for our system. We need to setup the interpolation or decimation rate, gain and also the center frequency depending on the type of system we have while connecting to a USRP2 sink or source.

2.3 The 'Hello World' in the Experimental Setup

Before we proceed with USRP2 platform, we need to make sure that the necessary hardware is available and the required softwares are installed. The following specifications are used in the experimental setup:

- Host Platform: Ubuntu 10.04 LTS (Lucid Lynx)
- Hardware Peripheral: USRP2 with daughter board, XCVR2450

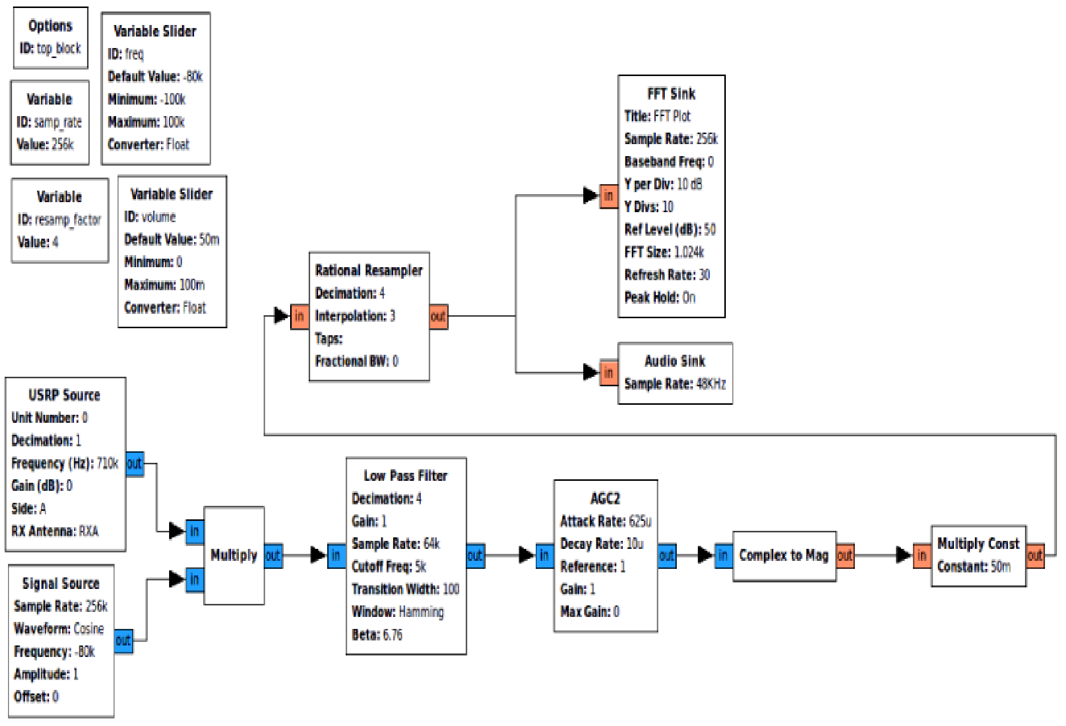
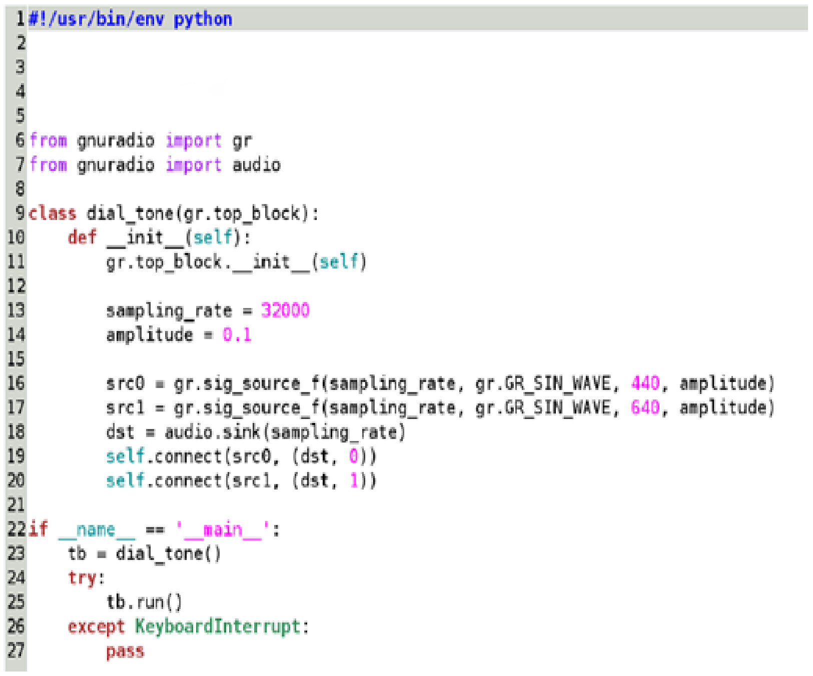


Figure 2.6: System Model of FM Receiver using GRC in GNU Radio

- SDR Version: GNU Radio 3.3.0

The USRP2 is connected to the host platform with a gigabit ethernet cable. To check if the USRP2 is detected by the computer after plugging it in, use the following command by starting a new terminal: *sudo find_usrps*. This will provide us the MAC address of the sensor node connected to the host. To start the GNU radio companion, a GUI for building projects, use the following command in the terminal: *gnuradio-companion*.

Executing your project requires the use of Python, which can be invoked via the GUI or the terminal. It is recommended to execute any Python file from the terminal with the sudo command because general errors can result from lack of admin permission. For example, to execute a file, the following command will enable you to run the python file 'myfile.py': *sudo python myfile.py*. Many of the example files provided by GNU Radio also have options that can be specified with the execution command. To see if there are any options, use the following command: *sudo python myfile.py -help*.



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




from gnuradio import gr
from gnuradio import audio

class dial_tone(gr.top_block):
    def __init__(self):
        gr.top_block.__init__(self)

        sampling_rate = 32000
        amplitude = 0.1

        src0 = gr.sig_source_f(sampling_rate, gr.GR_SIN_WAVE, 440, amplitude)
        src1 = gr.sig_source_f(sampling_rate, gr.GR_SIN_WAVE, 640, amplitude)
        dst = audio.sink(sampling_rate)
        self.connect(src0, (dst, 0))
        self.connect(src1, (dst, 1))

if __name__ == '__main__':
    tb = dial_tone()
    try:
        tb.run()
    except KeyboardInterrupt:
        pass
```

Figure 2.7: Python code snippet in GNU Radio

Once both the USRPs are connected to the host, we can check the transmission and reception paths. This will ensure the sanity of the daughter boards, the antennas and the SDR version we are using.We use the usrp2_fft.py from gr_utils folder and tune for a known signal. We send it at a known center frequency from the transmitter USRP2 and tune the receiver USRP2 with this script. Once we obtain the required signal at the correct center frequency, we can use the platform. We can also track any offset related issues in the two nodes due to ppm differences and take in into consideration for the rest of the experiment. There are some useful tools available in the setup too. usrp2_fft.py can also be used as a spectrum analyzer and usrp2_rx_cfile.py can be used as a recorder.

The dial tone example is the commonly used program for checking the USRP2 platform and the GNU Radio. Here the GNU radio is used to produce a tone that is similar to the dial tone one would hear from a landline telephone. In this example, the python code and the GUI interface of GNU radio will both be used. We will look into the python code given in Fig. 2.7 to understand how a script is constructed with class *dial_tone*. Function *gr.sig_source_f* is used to create a float type sine wave with the same amplitude and sampling rate but with different frequencies, one at 440Hz and the other at 640Hz. The first signal source output is connected to the input at the first port (Port 0) of the sound sink and the second is connected to Port 1. We also define a sink or destination, which writes the input into a sound card.

Chapter 3

Preliminary Work: OFDM Architecture on USRP2

In this chapter, we look into how OFDM is used in modern wireless system and study the existing setup on the USRP2. Section 3.2 discusses the transmitter and receiver flowgraphs in GNU Radio along with the signal processing blocks used in the platform. We also look into receiver synchronization and channel estimation.

3.1 Introduction to OFDM

Orthogonal Frequency Division Multiplexing (OFDM) is a combination of modulation and multiplication. It provides large data rates and highly robust to radio channel impairments. In the OFDM scheme, a large number of orthogonal, narrow band sub channels or subcarriers, transmitted in parallel, divide the available bandwidth. There is minimal separation between the carriers; hence there is very compact spectral utilization. Since the carriers are orthogonal, there is no crosstalk between the sub channels. The OFDM signal generated by taking the IFFT over N sub-carriers can be written as,

$$x(t) = \sum_{k=1}^{N} c_k e^{j2\pi f_k t}, 0 \leq t \leq T, \tag{3.1}$$

where c_k is the data symbol, $f_k = k/T$ is the subcarrier, and T is the length of the OFDM interval. The OFDM symbol duration T_s is added to the guard interval T_g which gives $T = T_s + T_g$, which is the total OFDM duration. This makes adjacent subcarriers separated by $1/T$. The OFDM system is implemented with the IFFT/FFT pair along with a DAC/ADC pair. At the transmitter, the signal is defined in the frequency domain. The frequency is discrete and such that each carrier corresponds to each element of the discrete Fourier spectrum. The amplitude and phases of the carriers correspond to the data to be transmitted. A serial data stream is parallelized. The IFFT helps to serialize this parallel data, and hence obtain a time domain signal. Then at the receiver, FFT is done to obtain the subcarriers and the data on each on these carriers.

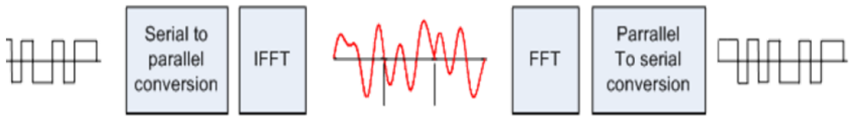


Figure 3.1: OFDM System

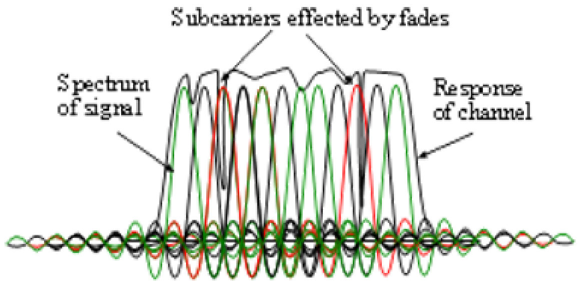


Figure 3.2: Effects From Deep Fading

In any wireless channel due to multipath effects, a signal undergoes fading and delay [22]. The fading in Fig. 3.2 changes the amplitude of the signal whereas the delay phase brings about a change in the phase and hence a frequency offset. The frequency offset leads to loss of orthogonality. The advantage of using OFDM is that in spite of having deep fading only some subcarriers are affected. We can overcome the information lost using good channel coding techniques. Similarly the delay spread brings about a timing offset and loss of information which can be overcome by using a Cyclic Prefix, as shown in Fig. 3.3. This helps in solving intersymbol interference(ISI) issues and provides the receiver with some guard time. A section of the composite OFDM symbol is copied so that even if there is a delay we do not lose information although it is redundant. Due to its advantages, OFDM is prevaent in WiFi 802.11a/g/n, MIMO, ADSL, UWB and many more wireless systems.

3.2 Existing OFDM Architecture And Implementation On USRP2

In this section we discuss about the most common implementation of an OFDM with a IFFT/FFT pair. We investigate the current implementation in GNU Radio and look into the details of the specifications. We used the benchmark codes for all our initial studies.

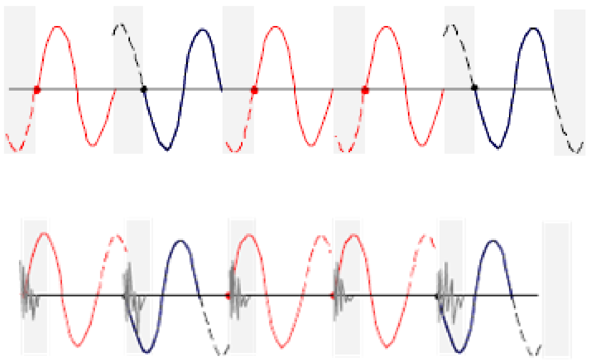


Figure 3.3: Effects From Delay Spread

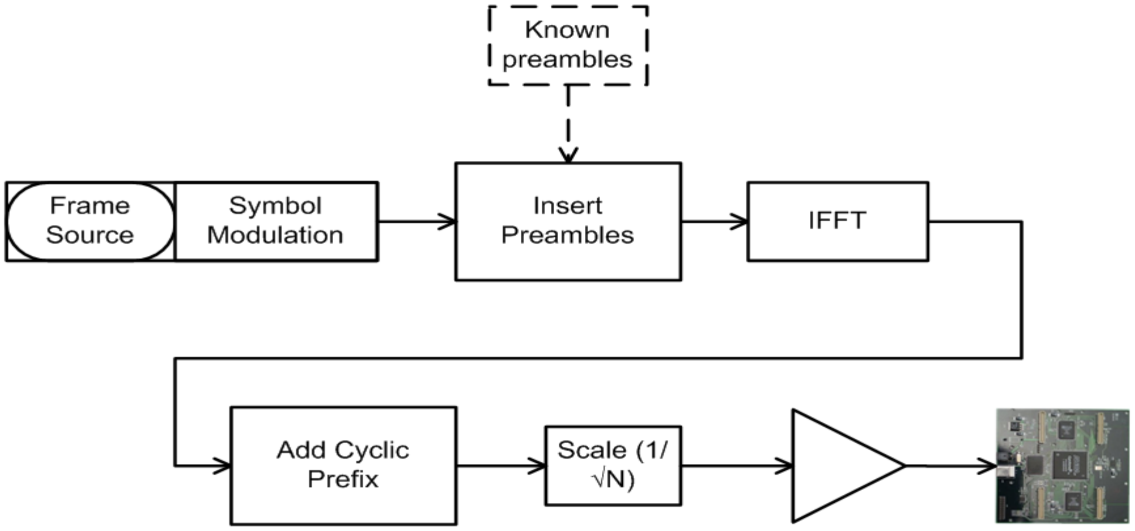


Figure 3.4: OFDM TX Flow Diagram

3.2.1 OFDM Tx Baseband Model

The binary information which is the form of serial data is grouped, modulated and coded. The serial data is parallelized depending on the number of available subcarriers or occupied tones. A known PN sequence is inserted before the data followed by serializing it with IFFT block. Then the N-point IDFT is done to change the data sequence from frequency domain to time domain. Once the composite signal is framed, the cyclic prefix is added. The scaling is done so that the data is normalized while streaming to the USRP2. This goes to the upconvertor and then goes to the DAC for transmission. Fig. 3.4 has a flow diagram on which ofdm_tx.py is built. The USRP2 implements the blocks with C++ modules are defined in Fig. 3.5. This is implemented in benchmark_ofdm_tx.py

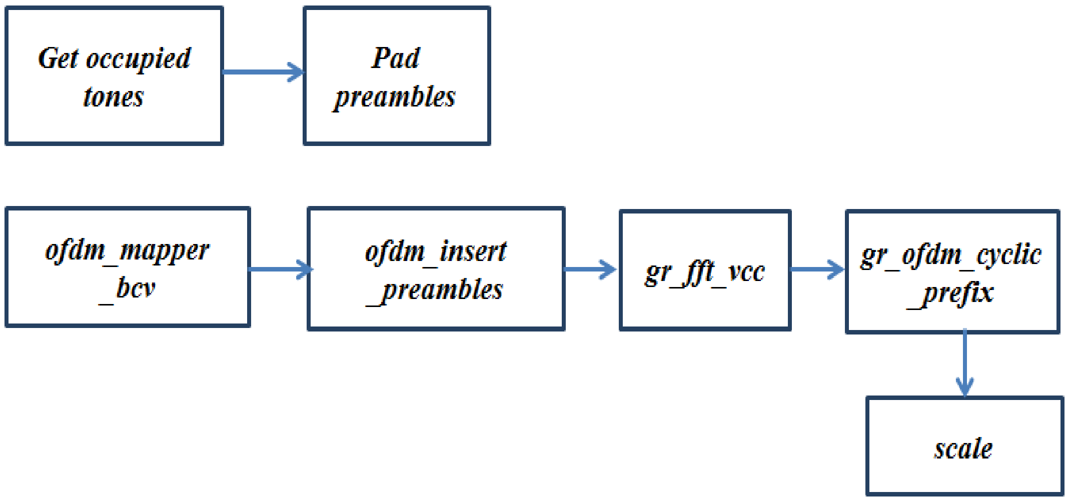


Figure 3.5: OFDM TX Flowgraph ofdm_tx.py

which calls the ofdm.py where class ofdm_mod defines the flow of the OFDM transmission. The architecture is hierarchical rather than flat. If $\bar{X}$ denotes the input data to the IFFT block such that

$$\bar{X} = [X_k]^T, k = 0, 1, ..N - 1, \tag{3.2}$$

then the output of the IFFT block can be written as

$$x(n) = IDFT_N(\bar{X}) = \sum_{k=0}^{N-1} X_k e^(i2\pi nk/N). \tag{3.3}$$

The *ofdm_mapper_bcv* packs the data into the subcarriers, followed by insertion of preambles. *ofdm_insert_preambles* checks for the preamble length and appends the modulated payload to the preambles. Once the preambles are inserted in *ofdm_insert_preambles*, we use the FFTW C library to calculate the N-point backward DFT. The function *gr_fft_vcc* handles this. Then the cyclic prefix length and symbol is send to the *gr_ofdm_cyclic_prefix* to construct the final signal. We need to scale the signal before transmitting it.

3.2.2 OFDM RX Baseband Model

At the receiver, after USRP2 ADC, it is windowed using a Hamming Window. The symbol needs to be synchronized since it undergoes fading and delay spread in the channel. So it goes to

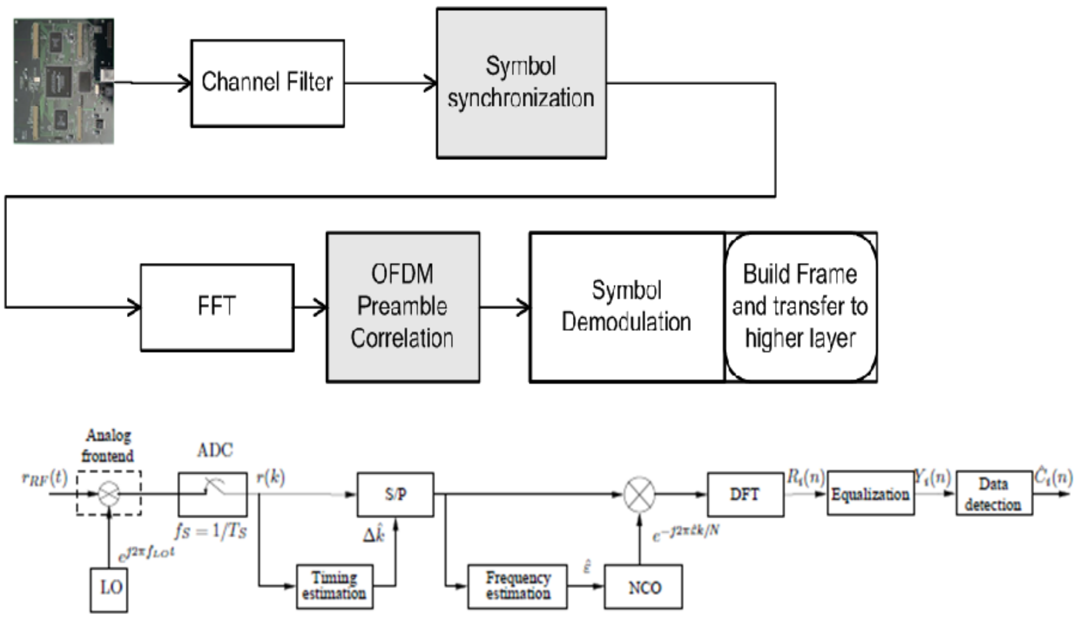


Figure 3.6: OFDM RX

the synchronization block which provides the timing and frequency information while sampling the received signal. Once the cyclic prefix is removed, the N-point DFT is used to convert from time domain to frequency domain. Then it is corrected for fine frequency and estimation. If $\bar{Y}$ denotes the output data of the FFT block

$$\bar{Y} = [Y_k]^T, k = 0, 1, ..N - 1, \tag{3.4}$$

we can derive it's relationship with the input data symbols $\bar{X}$ as,

$$\bar{Y} = DFT_N(IDFT_N(\bar{X}) \otimes h_n + w_n), \bar{Y} = diag(\bar{X})H + W, \tag{3.5}$$

where h_n and w_n are the sampled channel impulse response and the AWGN (Additive White Gaussian Noise) and H and W denote the N-point DFT of h_n and w_n. Here ofdm_rx.py calls classes *ofdm_receiver* and *ofdm_demod*. The module ofdm_receiver takes care of the time and frequency synchronization with *ofdm_sync*. This gives the timing and frequency correction information to the *ofdm_sampler*, which samples the received signal. The *ofdm_sampler* looks for the preamble and removes the cyclic prefix to give the OFDM symbol to *ofdm_frame_acq*. The block *ofdm_frame_acq*

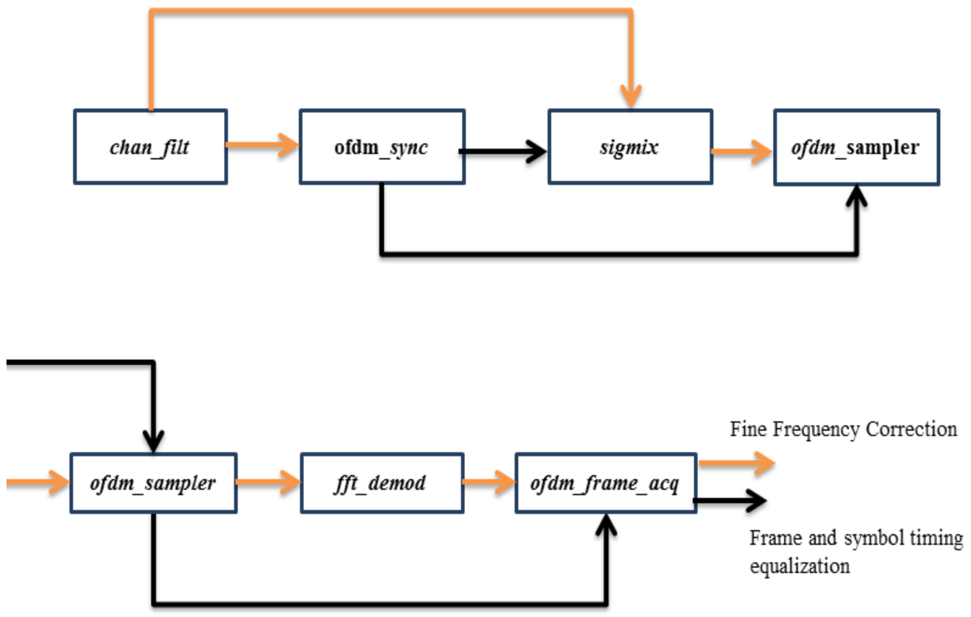


Figure 3.7: OFDM RX Flowgraph ofdm_rx.py

takes care of the fine frequency tuning by preamble correlation method and does the channel estimation. The ofdm occupied tones along with the preamble information is given to *ofdm_demod*. *ofdm_demod* does the demodulation followed by *ofdm_frame_sink* which captures the packets in a target queue. The sub modules of ofdm_receiver are given in Fig. 3.7.

3.2.3 Receiver Synchronization

In any wireless system, the multipath effects add to timing and frequency offsets. So it is important to find the start of every symbol correctly. It becomes necessary to get the right time delay and the phase offset that the symbol undergoes. In this section, we discuss how the preambles or known symbols are used to identify the start of the symbol and various synchronization models available in GNU Radio.

The Schmidl & Cox (S&C) model [23] is used to create the preambles or the pilot symbols in the current USRP2 implementation. Here the preambles are known symbols used for used for synchronization and initial channel estimation in the current setup. Complex models have pilots for

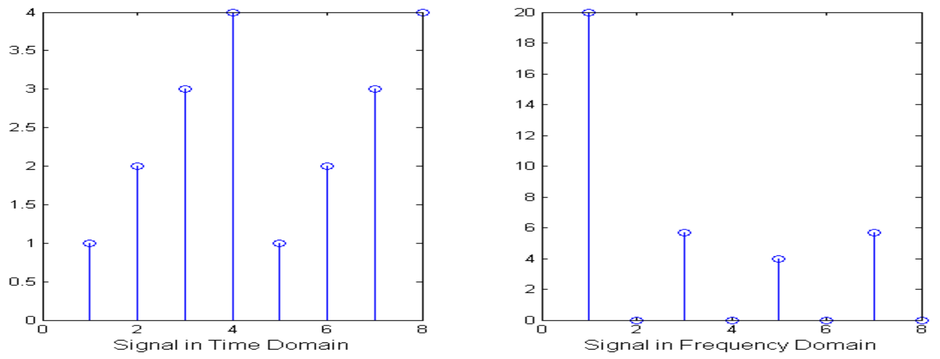


Figure 3.8: S & C Model

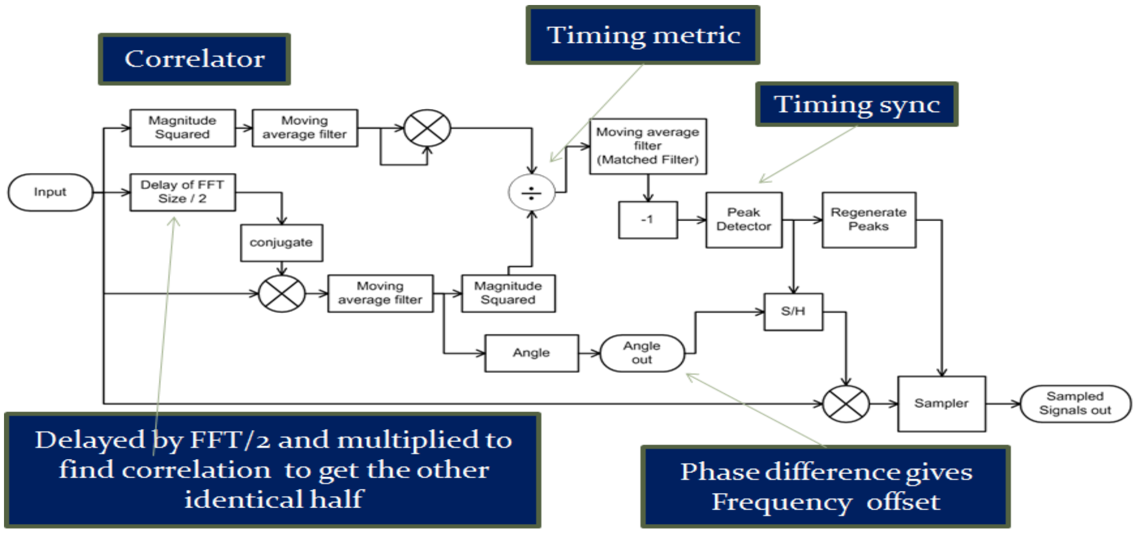


Figure 3.9: Receiver Synchronizer

tracking channel and fine tuning the estimates [24]. However the receiver can be chosen between PN synchronization [23], ML Estimation [26, 27] and PN Sequences with Acknowledgements [25]. Both PN synchronization and ML estimation were tried in the existing setup. Here we discuss the Symbol Timing and Carrier Frequency Offset Estimation algorithm. In the S & C model, the receiver tries to find start of frame/symbol by looking for a symbol whose first half is identical to the second half in time domain. Both the halves will remain the same while travelling through the channel except that there will be a phase difference between them due to the carrier frequency offset. Here they are made identical in time by sending a PN sequence on the even subcarriers and zero on the odd subcarriers. In Fig. 3.8, a symbol with two identical halves [1, 2, 3, 4] is set is time domain. While taking the FFT, we can observe that the odd carriers are nulled. Since FFT/IFFT are

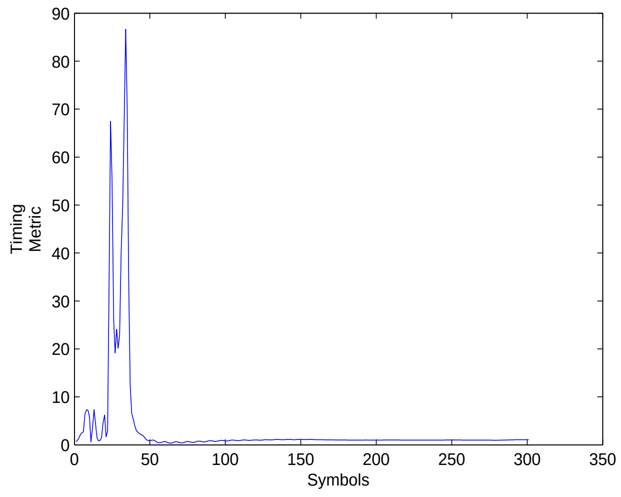


Figure 3.10: Timing Metric from Received Samples

commutative, the inverse frequency to time is what happens with the preambles. Correlation btw identical halves used to calculate timing metric. This helps determine the start of frame. Assuming that we have L samples in the first one-half of the training symbol (avoiding the cyclic prefix), we can calculate the moving average sum of cross correlation $P(d)$ as

$$P(d) = \sum_{m=0}^{L-1} r_{(d+m)} \cdot r_{(d+m+L)}, \tag{3.6}$$

where r is the sampled complex sample and d is the time index of the first sample in a window of $2L$ samples. The moving average sum of received energy of the second half of the training symbol is given as,

$$R(d) = \sum_{m=0}^{L-1} |r_{(d+m+L)}|^2. \tag{3.7}$$

From the above we can calculate timing metric as,

$$M(d) = |P(d)|^2/(R(d))^2. \tag{3.8}$$

The timing metric is like a plateau and the start of the symbol can be taken anywhere on this plateau. The timing metric *ofdm_sync_pntheta_f.dat* is spiky and this could be because of the presence of the cyclic prefix. Fig. 3.10 shows the timing metric from the received samples in the USRP2 calculated

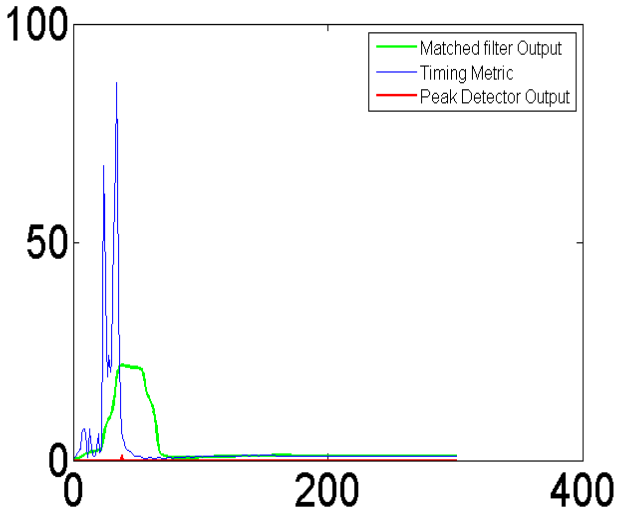


Figure 3.11: Detection of Preambles by Matched Filter

in ofdm_sync.py using Equation 3.8. This information goes to the matched filter which detects the presence of a known symbol and received signal. We regenerate the peaks with which we sample the received signal. Fig. 3.11 shows the output of the matched filter *ofdm_sync_pn-mf_f.dat* from the USRP2. The phase difference ϕ between the identical halves gives the frequency offset estimation as in,

$$\phi = \pi T \delta f, \tag{3.9}$$

$$\bar{\phi} = angle(P(d)). \tag{3.10}$$

Hence, the carrier frequency offset δf is calculated as,

$$\delta f = \bar{\phi}/\pi T. \tag{3.11}$$

Then the two halves are corrected by multiplying the two samples with exp-2j/T. This angle is used by the *ofdm_sampler* to sample the signal that comes directly from the channel. Once the phase difference between the subcarriers of the known symbols or the preambles is known, it is applied to all the symbols till the next preamble comes. This gives us the coarse frequency and timing synchronization. The fine tuning and estimation is addressed in the next section.

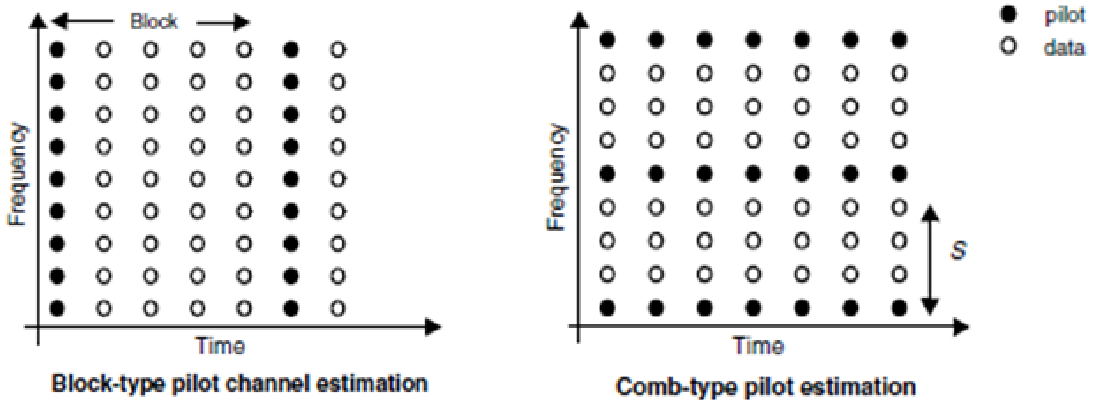

Block
Frequency
Time
Block-type pilot channel estimation
Frequency
Time
S
Comb-type pilot estimation
● pilot
O data

Figure 3.12: Arrangement of Pilots

3.2.4 Classic Channel Estimation

Once the OFDM symbols are synchronized, they need to be corrected for channel impairments. Channel estimation is a complex two dimensional interpolation problem. Since it is time varying, a 2D Wiener filter should be the ideal way to estimate a channel. However since building such an estimator is highly complex, most implementations have a 1D equalizer.

The most common type of channel estimation is pilot-aided estimation. Known symbols called as preambles are used for getting the initial channel estimates and pilots are inserted in between to track the channel. These are mainly categorized as block type pilots and comb type pilots [36]. Fig. 3.12 shows how pilots are arranged in the time and frequency domain in both these mthods. As cited in [35], this work discusses various estimation methods, LS, MMSE and also the importance of estimating channels in OFDM. Again in all these methods, it is limited to the frequency domain and only half the interpolation issue is solved. We get an insight of how frequency domain pilots can be used to obtained the channel impulse response (CIR) by cyclic correlation method in [38]. Another way to improve the channel estimation is improve the accuracy of the interpolation itself. The paper [39] gives an overview of good interpolation techniques like second order or spline interpolation can improve the channel estimate accuracy and the lays down the importance of time domain interpolation also.

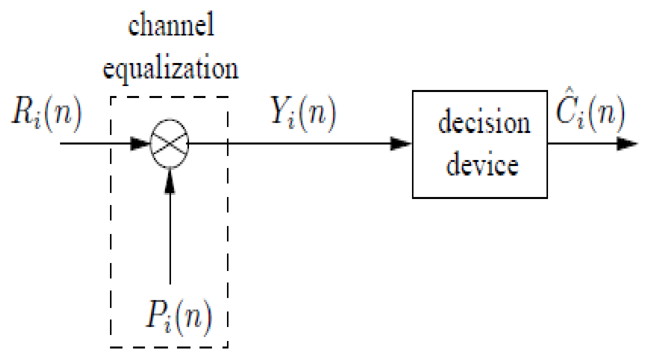


Figure 3.13: Channel Estimation

In the current OFDM implementation in USRP2, the channel estimation is done after taking the FFT. The *ofdm_sampler* feeds the sampled signal to the *fft_vcc* block for taking the FFT. So the channel estimates are taken in the frequency domain. The channel estimation is done in the *ofdm_frame_acq* module. Here they use one of the known symbols to estimate the channel response and apply a single tap equalization to all the carriers. The symbol to be used for the channel estimates is known by doing a cross correlation over a shifted FFT length or bins (default = 10). This helps in estimating the taps of the odd subcarriers. The taps on the even subcarriers are then obtained by simple linear interpolation. The received signal after FFT modulation can be given as,

$$R_i(n) = H_i(n)C_i(n) + W_i(n), 0 \leq n \leq N-1, \tag{3.12}$$

$$P_i(n) = 1/(H_i(n)), \tag{3.13}$$

where $R_i(n)$ is the received sample, $H_i(n)$ is the channel gain in the frequency domain, $C_i(n)$is the original transmitted data and $W_i(n)$ is the AWGN at the nth sample

$$Y_i(n) = (R_i(n))/(H_i(n)) = C_i(n) + W_i(n)/H_i(n), 0 \leq n \leq N-1, \tag{3.14}$$

$$H(n) = (R_p(n))/(C_p(n)) = H_p(n) + W_p(n)/C_p(n), 0 \leq n \leq N-1, \tag{3.15}$$

where $R_p(n)$ is the received known symbol, $H(n)$ is the channel estimate in the frequency domain, $C_p(n)$ is the preamble or known symbol transmitted at the nth sample.

Chapter 4

CS-OFDM Architecture And Implementation

In this chapter, we present the scheme that will be used to change the channel estimation in the existing setup. An overview of compressive sensing (CS) is given in Section 4.1 and we look into how CS can be applied to our model in Section 4.2.1 This is followed by the receiver design and the transmitter design.

4.1 Compressive Sensing

Compressive sensing uses less information to retrieve the required signal. Unlike Shannon's Nyquist sampling, compressive sensing uses subNyquist sampling to reconstruct the entire signal [4]. However the reconstructed signal must be sparse in nature which enables fewer linear measurements to reconstruct it. Since the channel estimates h_n is sparse in nature, we can reconstruct it by constructing a carefully designed measurement matrix ϕ, and using optimization techniques like ℓ_1 minimization a.k.a Basis pursuit algorithms [40, 42], matching pursuit algorithms [41] et al. to obtain the channel estimates h_n.

$$y = \phi \mathbf{h_n}. \tag{4.1}$$

This helps us overcome over-parameterization that occur with traditional techniques like least squares and which results in the poor performance of the estimator.

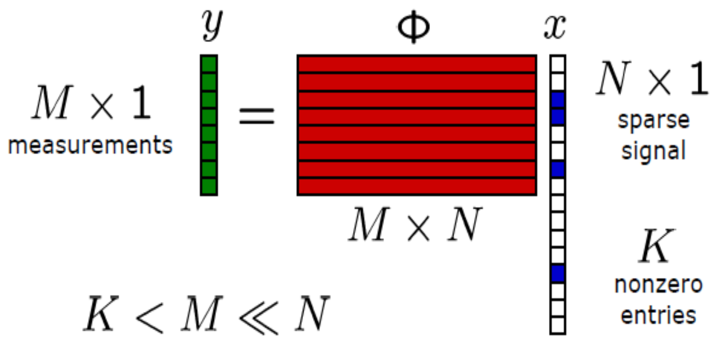


Figure 4.1: Compressive Sensing

The measurement matrix ϕ must be designed in such a way that it adheres to a few properties so that we can obtain h from this under determined set of equations. Any information in h must not be damaged by a dimensionality reduction of ϕ. So ϕ must obey the Restricted Isometry Property (RIP) [20], or the uniform uncertainty principle after scaling in Equation 4.2 which is given as,

$$(1-\delta_s)||h||_2^2 \leq ||\phi h||_2^2 \leq (1+\delta_s)||h||_2^2, \tag{4.2}$$

where δ_s is the RIP parameter and $||.||_2^2$ is the ℓ_2 norm of the vector. ϕ obeys RIP for

$$K \leq M \leq logN, \tag{4.3}$$

where

$$\phi_m = \text{random Gaussian},$$

$$\phi_m = \text{random Binary } or$$

$$\phi_m = \text{randomly selected Fourier samples}.$$

We could solve for h, using the ℓ_2-norm, however the solution is never sparse. Since we know a priori that our signal is sparse, we could use ℓ_0-norm. However, this is computationally exhaustive, and hence not a feasible solution. Hence, we can solve for h_n by

$$min||h||_{\ell_1} \tag{4.4}$$

$$\text{s.t. } y = \phi h_n.$$

4.2 Proposed Architecture

4.2.1 How CS Fits In?

Calculating radio impairments in a wireless system is what makes the baseband receiver different from other pass band receivers. For an OFDM system, the channel impulse response can be modeled as

$$h(t) = \sum_{l=1}^{L} a_l \delta(t-\tau_l), \tag{4.5}$$

where a_l is the complex multipath component, L is the total number of multipaths, δ is the Dirac delta function and τ_l is the multipath delay. A cyclic extension of length T_g is chosen greater than τ_l to avoid intersymbol interference [14]. We can see that h_n is sparse in time domain, since the number of multipath components l is small. This is because wireless channels are highly susceptible to noise and time. Hence we can incorporate compressive sensing in estimating the channel impulse response in time domain. The paper [15] gives a method which helps in using compressive sensing for channel estimation in the frequency domain. This system model for this proposal is based on [16], which uses random convolution and the asymmetric structure of a DAC/ADC pair to estimation. This is a novel approach compared to existing CS based channel estimation structure. The work in [17] points to an interesting model to use frequency based channel estimation, however the asymmetric nature of the DAC/ADC pair cannot be used in this model.

We will discuss the system model discussed in [16], and then demonstrate how it can be fitted in the existing system. The sampled received signal $z(n)$ in a wireless node can be written as,

$$z = x \otimes h_n + w, \tag{4.6}$$

where $\otimes$ denotes convolution, h_n is the sampled channel response, and w is the sampled AWGN noise. At the ADC, z will be sampled at rate M to obtain y_m. However, we have to note that this is convolution and not multiplication. So we need to rewrite Equation 4.6 as,

$$z = Ch_n + w, \tag{4.7}$$

$$y = P_\Omega z = P_\Omega(Ch_n + w), \tag{4.8}$$

$$y = (P_\Omega C)h_n + w_\Omega, \tag{4.9}$$

where C is the full circulant matrix determined by x, $P_\Omega C$ denotes the down sampled points and w_Ω is the down sampled AWGN noise. Here we need to design $P_\Omega C$, such that we can still reconstruct h.

Equation 4.10 shows the multiplicative property of a time based convoluted signal, which helps us in designing the pilots in Equation 4.11,

$$x \otimes h = F^{-1}(F(x) \cdot F(h)), \tag{4.10}$$

$$x \otimes h = F^{-1}(F(F^{-1}(X)) \cdot F(h)), \tag{4.11}$$

where X is the pilots in frequency domain. On simplifying the Equation 4.11 we get,

$$x \otimes h = [F^{-1} diag(X) F] h. \tag{4.12}$$

X is designed such that discrete value X(k), k=1,2,..N, have independent random phases and uniform amplitude. This is random convolution [43], where a random waveform is convoluted with h, followed by random time domain subsampling. So here the channel estimates are done in the time domain. In order to obtain higher resolution, [16] also implements a down sampling at the receiver. Hence the RX will be down sampled by about 8 to 16 times. The asymmetric DAC/DAC pair helps in a attaining the high resolution and also takes advantage of the high speeds a DAC can take. Hence the equation for the final system model becomes

$$y = \Omega[F^{-1}(F(F^{-1}(X)) \cdot F(h))] + \Omega w. \tag{4.13}$$

This makes the sensing matrix take the form of Equation 4.14 as,

$$A = M^{-1/2} P_{\Omega} C. \tag{4.14}$$

We compare the MSE vs. SNR for the existing model for the maximum down sampling, 8 possible in the existing USRP2 setup (Max TX = 512 DFT/Min RX = 64 DFT). The MSE converges for high SNR values hence we can rely on this model to estimate the channel impulse response.

The algorithm used in [16] for estimating the channel gains in time domain using compressive sensing is given in Table 4.2. ℓ_1-norm on the h is done followed by minimum mean square.

4.2.2 CS System Model

We modeled the OFDM model as a basis pursuit model in the YALL solver [18] as,

$$\min_{h_n \epsilon C_n} |W h_n|_{w,1} \tag{4.17}$$

$$\text{s.t. } C h_n = y_m,$$

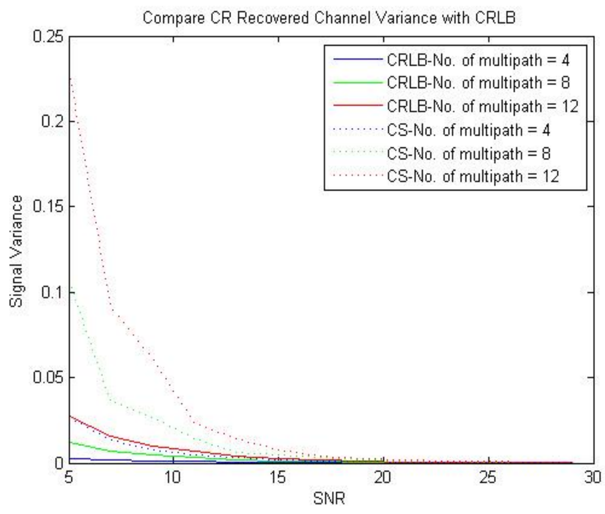


Figure 4.2: MSE vs SNR for CS Algorithm

to solve the undetermined equation to get h_n where $||.||_{w,1}$ is the weighted ℓ_1 norm. The algorithm in Fig. 4.3 uses iterative support detection for solving the ℓ_1 equation in noisy environments. Here basis pursuit denoising is done followed by the updation of weights which is used in the next iteration. The tolerance margin can be reduced at various steps of the iteration to improve the solution. Finally the least squares solution is used as a final denoising step. The complexity and the computation time in the least square step are reduced due to the sparsity of the h. We used YALL1 solver [18] since it has an accelerated convergence compared to other solvers. BP helps in attaining high resolution and speed [18] by dividing a signal into smallest number of ℓ_1 coefficients. Moreover it was easier to implement a custom portion of the YALL1 on the embedded platform after it was available as a open source module.

There are two possible models for compressive sensing based estimation namely symmetric and asymmetric model. In the symmetric model, the transmitter and the receiver are maintained at the identical IFFT/FFT. At the receiver, the preambles are subsampled and compressive sensing estimation is done. However since we need to extract the data from the entire sample set, this method is just a proof of concept. A better approach would be to model in an asymmetrical method for channel estimation by using a higher bin FFT for just the preamble at the transmitter. Since the channel impulse response is sparse in nature, we can use CS to obtain a high resolution response

Algorithm 4.1 Theoretical CS OFDM

```
Input: $\phi$, $y$;
Initalize:
$\tilde{\phi}$ as the first $\tilde{N}$ columns of $\phi$.
$I^0 \leftarrow \emptyset$ and $w_i^0 = 1, \forall i \in \{1, 2, \ldots, \tilde{N}\}$
while the stopping condition is not met, do
    Subproblem:
        $\tilde{h} \leftarrow \arg\min \sum_{i \notin I^j} |\tilde{h}_i|, \text{ s.t. } \tilde{\phi}\tilde{h} = y.$    (4.15)
    Support detection: $I^{j+1} \leftarrow \{i : |\tilde{h}_i^j| \geq 2^{-j}\|\tilde{h}^j\|_\infty\}$, where $\|\tilde{h}^j\|_\infty = \max_i\{|\tilde{h}_i^j|\}$.
    Weights update:
    $w_i^{j+1} \leftarrow 0, \forall i \in I^{j+1}$; otherwise $w_i^{j+1} \leftarrow 1$.
    $j \leftarrow j + 1$
end while
Final least-squares:
let $T = \{i : |\tilde{h}_i| > \text{threshold}\}$, then:
        $\tilde{h}_T \leftarrow \arg\min_{\tilde{h}} \|\tilde{\phi}_T\tilde{h} - y\|_2^2$, and $\tilde{h}_{T^c} \leftarrow 0.$    (4.16)
Return $\tilde{h}$
```

from smaller number of samples at the receiver. Based on the work in [16], a high speed DAC at the transmitter and a regular speed ADC at the receiver will help us attain a higher dimensionality for h_n. So instead of using a higher bandwidth or a longer preamble [37], we can slice the same bandwidth at a higher rate to obtain finer channel estimates. This also gets translated to shorter probing times.

4.2.3 Bandwidth Utilization

Inorder to estimate the bandwidth that the OFDM signal spans over, we analyze the FFT of the received samples in the experimental setup. Fig. 4.4 shows the fourier transform of the received OFDM signal. We see that from the figure the bandwidth is about 400KHz. The bandwidth should be maintained during the estimation and the data reception. The same bandwidth needs to be utilized for both the preamble and the data. If we consider FFT size N = 512, occupied tones, occ = 128 with an interpolation rate i of 64, an ADC clock rate ADC_{clk} and a sample size of 1024 bytes, we can obtain a maximum transmission rate T of 1.562500 MS/s

$$T = ADC_{clk}/i, \tag{4.18}$$

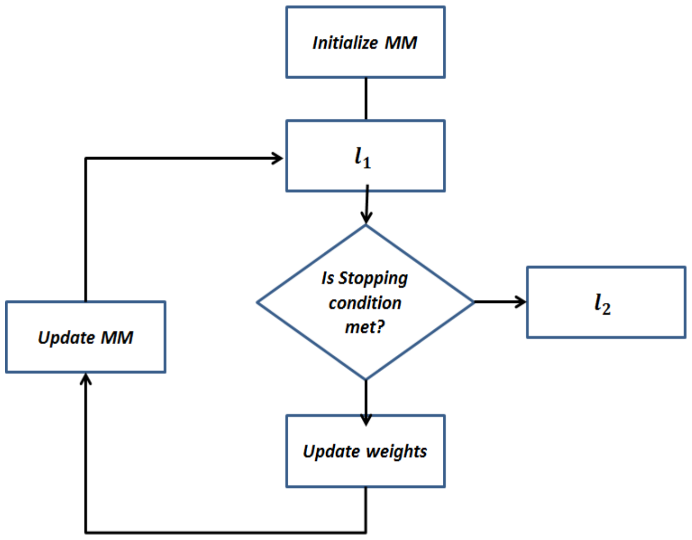


Figure 4.3: Compressive Sensing Flowchart

where 100MS/s is the maximum ADC rate. However we are limited by the occupied tones, so the bandwidth bw gets limited to

$$bw = T * occ/N, \tag{4.19}$$

which makes our bandwidth 390.625KHz. So the subcarrier spacing in our case turns out to be 762.9395Hz which is given by

$$\delta f = bw/N. \tag{4.20}$$

The symbol time T_s can be calculated considering the cyclic prefix length of CP = 192 samples. Therefore the total number of samples transmitted turn out to be 512 + 192 = 704 samples. We get a symbol time of 180.224 μs while applying the equation,

$$T_s = (1 + occ/N) * (1/\delta f). \tag{4.21}$$

The symbol time period T_s is small compared to the average computation time for CS to converge. In other words, we see that CS convergence takes ms which might be a huge hit against the symbol period.

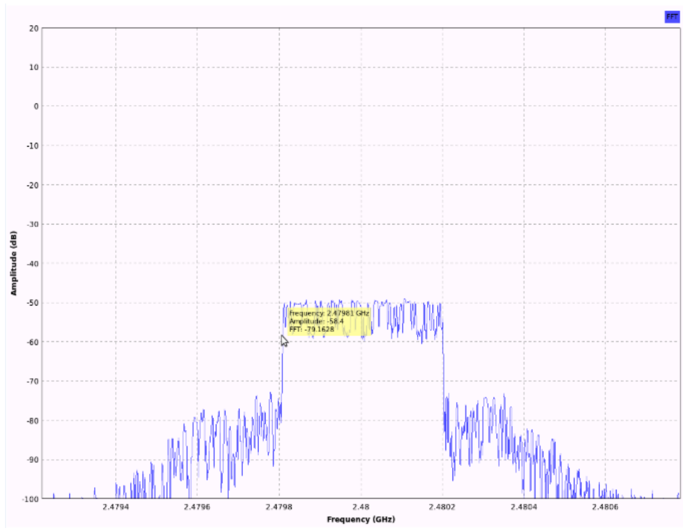


Figure 4.4: Bandwidth Analysis

4.3 Soft Radio Implementation

In this section, we look into different ways to implement the CS model in the embedded platform. The proposed system design is discussed along with the details in each block.

4.3.1 Approach Used for Implementing CS

A Soft Radio can implement most of the signal processing blocks defined in a theoretical model. However, practical systems need a lot of leverage than the theoretical ones to take care of processing time and rounding errors. So implementation of an existing algorithm to a practical systems has a lot of limitations. GNU Radio uses a mixture of Python and C++ and many believe that using both these platforms hinders the performance. The best approach is to lay down the signal flow in Python and the performance critical modules in C++. Although performance of the module might be a bottleneck, the reconfigurability trait in the software radio makes it attractive to use.

Another approach was whether to use a hierarchical model using *gr_hier2.block* with the GNU Radio. This was not pursued due to the iterative nature of the algorithm and the current constraints in the GNU Radio. Moreover developing each block on its own would give better observability and easier testability with the *qa* tests.

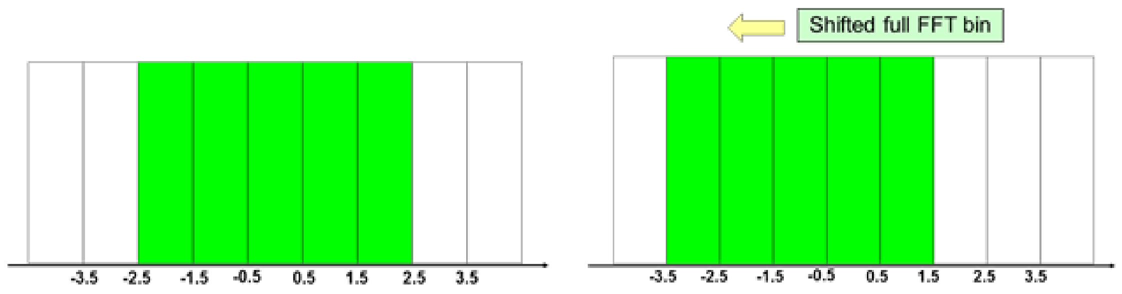


Figure 4.5: Ideal FFT Bins and Shifted FFT Bins due to Frequency Offset

The existing setup in the OFDM for GNU Radio uses frequency domain based channel estimation where the received known symbols was compared with the known symbols (preambles). Frequency compensation is done along with channel estimation in the *ofdm_frame_acq* block. The OFDM frequency bins can shift due to the multipath effect introduced by the channel and auto-correlation is done to find the number of bins shifted. The shifted index is used to calculate the weights on the odd carriers and the weights on the even carriers are then interpolated. This is a coarse method of channel estimation [28].

Currently both the frequency compensation and the channel estimation are done together in the *ofdm_frame_acq* block in the GNU Radio. For implementing CS, the channel estimation needs to be shifted and made into a separate block called the CS Estimator. The samples from the sampler block called *ofdm_sampler* are used since the estimation needs to be in the time domain. Once the channel estimates are obtained by the CS Algorithm, they need to be corrected for the frequency compensation. One of the major architecture related changes was shifting the channel estimation from the frequency domain to time domain. While shifting the channel estimation to time domain, the estimation error decreases owing to the decrease in the number of parameters to be estimated [30], although it is with the cost of complexity.

4.3.2 Proposed System Design

The top level view of the CS integration with the existing system is given in Fig. 4.6. In the symmetric model for Fig. 4.6, we need to downsample the time domain signal before the CS

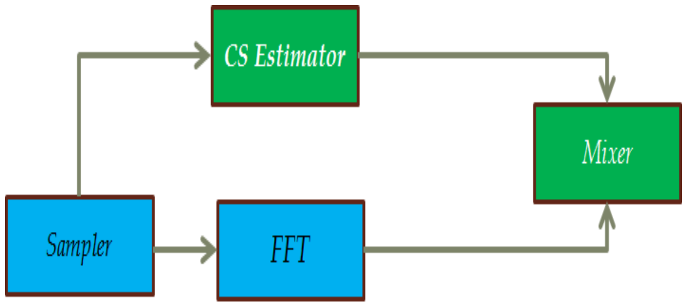


Figure 4.6: Top View for CS Time Equalizer

estimation in the existing setup. This held good for higher FFT models however we are throwing away known data in this case. In the asymmetric model, we use CS on the preamble which has been transmitted at a higher FFT while preserving the bandwidth. The CS model is explained extensively later. Apart from data, we also use a control signal, to indicate the end of a complete CS estimation cycle.

We built an initial measurement matrix (MM) based on the pilots and the received signal and update it to a new MM based on the estimated h_n. The measurement matrix $\mathbf{A}$ is defined by two function handles in Matlab which are defined as,

$$\mathbf{A}.times = def\mathbf{A}, \tag{4.22}$$

$$\mathbf{A}.trans = def\mathbf{At}, \tag{4.23}$$

$$= \text{IFFT } (\text{FFT } (y) * conj(pilots)).$$

We used FFT3W over GSL libraries for all the FFTs since this was computationally more efficient [32] as see in Fig. 4.7. Once the measurement matrix was available $\mathbf{A}$, it was used in the YALL block to solve for the ℓ_1 model. A stopping condition(SC) was implemented based on the algorithm and the limit on the symbol numbers so that convergence happen. Many supporting functions used in Matlab like abs, nnz, find had to be built in the MISC block. We could also use patches like Armadillo for the translating Matlab functions into C++ plug ins. Then the final denoising step was done using least-square block L2. Since this has an infinite number of solutions, we can use the pseudo inverse which is given in Equation 4.31. The ℓ_2 block implements the Moore-

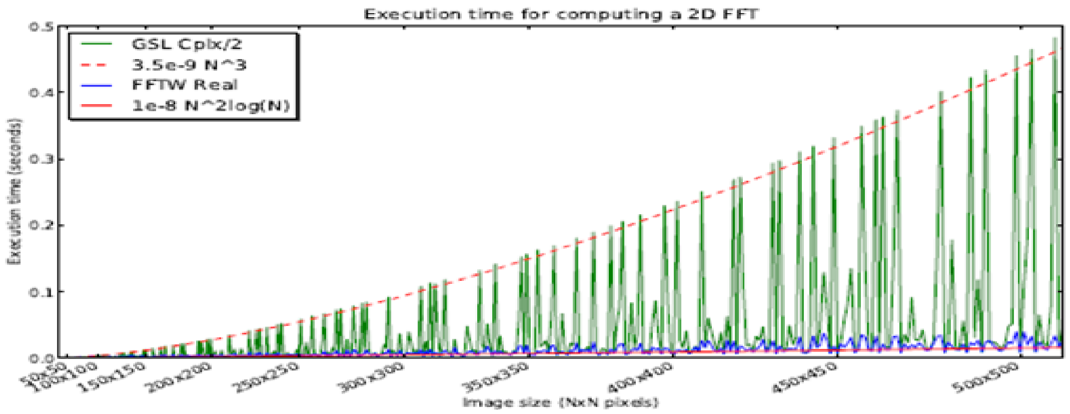


Figure 4.7: Comparison of FFTW3 and GSL Libraries

Penrose inverse using SVD. For any matrix, there exists $\mathbf{U}$ and $\mathbf{V}$ orthogonal matrices, and also a diagonal matrix $\mathbf{S}$ which contains the singular values. Since the input vector is sparse in nature, the final step is not computationally taxing. The GSL libraries are used for SVD. However, this could only be done in real number domain. This is rewritten to use the available functions since $\mathbf{A}_{new}$ is complex.

One of the major constraints in building a real-time system from simulations is the parallelism in the system. GNU Radio has a TPB (thread-per-block) scheduler by default. So when a system gets complex with increase in the number of blocks, it slows down the CPU. Since the CS algorithm takes more time to process due to its inherent complexity, we have to take a hit on the throughput. This pushes us to add one more parameter to converge the optimization. The CS block will converge on the number of symbols, the number of iterations and also a timestamp. Whenever the estimation takes longer than expected, we will converge with the existing results and to that we do not stall the whole system and packets can be captured. Similarly we had a lower bound for the time stamp too since we did not want samples that were less accurate. So this was a tradeoff between speed and accuracy. We also compared the data before and after the mix block in Fig. 4.6 to check what percent of the data is affected by incomplete in convergence, which comes to only 1.5% which is small portion of the total data set.

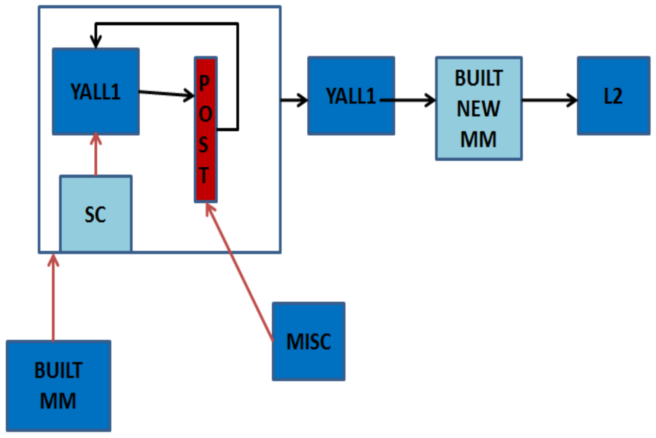


Figure 4.8: Block Diagram for CS Time Equalizer

4.3.3 YALL1 for ℓ_1 Optimization

In simulations YALL1 was used to solve the ℓ_1 scenario. The major task was to write an equivalent of the

$$h = yall1(\mathbf{A}, y, opts), \tag{4.24}$$

in the real time system. Here y is the m-vector downsampled received time domain sample and h is the n-vector channel estimate in time domain where m is less than n. $\mathbf{A}$ is the measurement matrix which can be constructed as a matrix directly or as a combination of two handles. We chose the same approach as the simulations since it was easy to implements. The two function handles f and g are

$$\mathbf{A}.times = f(\mathbf{A} * y), \tag{4.25}$$

$$\mathbf{A}.trans = g(\mathbf{A}'h). \tag{4.26}$$

which is defined in our algorithm. The $opts$ helps us to set various parameters like $opts.tol$ stopping tolerance, $opts.maxit$ maximum iterations, $opts.mu$ the penalty parameter etc.

The BP model was chosen out of the six models that YALL1 can solve. Basis Pursuit is a principle for decomposing a signal into an "optimal" superposition of dictionary elements, where optimal means having smallest number of ℓ_1 coefficients among all such decompositions [47]. Basis

Pursuit approach utilizes the sparse nature of the object. It attains super resolution and speed compared to the traditional approaches. Method of Frames (MOF) and Orthogonal Matching Pursuit (OMP) are other methods that can be adopted. However in method of frames, the ℓ_2 norm is done instead of ℓ_1 norm. MOF leads to quadratic optimization with linear equality constraints whereas BP is nonquadratic problem with a convex solution. Linear programming is a kind of optimal basis problem. However YALL1 is based on a classical approach of Alternate Direction Method Multipliers (ADMM) [48] which uses augmented Lagragian functions. The ADMM problem seeks to

$$\min f(x) + g(z) \tag{4.27}$$
$$\text{s.t. } \mathbf{A}x + \mathbf{B}z = c,$$

where f and g are convex functions. Two separate variables x and z was identified from the Equation 4.27 for YALL1 and solved in this method.

The stopping condition or the condition to converge for this algorithm was based on the penalty parameter μ and residue rd. The penalty paramater μ is calculated as,

$$\mu = mean(abs(y)). \tag{4.28}$$

A residue rd was calculated from z and using the residue calculate a relative gap and a dual residual. A comparison is made based on how quickly the relative gap grows based on the dual residual and μ is updated based on this step. However this was done after a certain number of iterations in YALL1.

4.3.4 ℓ_2 Optimization

The pseudo inverse is then used for find the least squares for Equation 4.29. A least squares problem can be approached in many ways on an embedded Linux platform. There are various packages available like MINPACK, ceres-solver from Google project [31] for solving least squares directly. We solved the least squares problem using the most common approach, the Moore-Penrose pseudo inverse due to its robustness and ease to use. We need to solve the equation,

$$b = \mathbf{A}y, \tag{4.29}$$

which can be re-written as,

$$y = \mathbf{A^t} b, \tag{4.30}$$

where $\mathbf{A^t}$ is the Moore-Penrose "pseudo inverse" of the matrix $\mathbf{A}$. The pseudo inverse is best computed using the Singular Value Decomposition (SVD). For every matrix $\mathbf{A}$, there exists orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$, such that $\mathbf{A}$ can be decomposed. In our flowchart, we obtain $\mathbf{A}_{new}$ from $\mathbf{A}$. The new measurement matrix $\mathbf{A}_{new}$ obtained after post processing, is decomposed into $\mathbf{U}$, $\mathbf{S}$ and $\mathbf{V}$ using,

$$\mathbf{A}_{new} = \mathbf{U}\mathbf{S}\mathbf{V}^t, \tag{4.31}$$

$$\mathbf{A}_{new}^{-1} = \mathbf{V}\mathbf{S}_i\mathbf{U}^t, \tag{4.32}$$

and $\mathbf{A}_{new}^{-1}$ was obtained.

SVD is easily available in numerical linear algebra packages like LAPACK, GSL. A Matlab-C++ cross compatible tool called Armadillo can also be used instead. GSL (GNU Scientific library) was chosen due to the quick availability and easiness in debugging. The ability to use GSL in high computing applications has been often overlooked in the GNU Radio platform. This is inspired from the *gr-wavelet* block that uses gsl for its wavelet applications. The final denoising step which gives us the channel estimate is

$$h_{out} = y * pinv(\mathbf{A_{new}}), \tag{4.33}$$

where $\mathbf{A_{new}}$, the updated measurement matrix is used to solve the the least squares problem. GSL provides a function *gsl_linalg_SV_decomp* for real number SVD decomposition.

However the current algorithm has to be extended owing to the complex nature of $\mathbf{A_{new}}$. An NxN complex matrix is extended into a 2Nx 2N matrix by separating the real and imaginary portions. $\mathbf{U_r}$ and $\mathbf{U_i}$ are the real and the imaginary parts of Matrix $\mathbf{U}$ whereas $\mathbf{V_r}$ and $\mathbf{V_i}$ are the real and the imaginary portions of matrix $\mathbf{V}$. The only tradeoff with this approach is the speed. Fig. 4.9 shows the how SVD decomposition of a complex matrix is done using the available functions in GSL library. The decomposition is done on the larger matrix and the results are concatenated to get

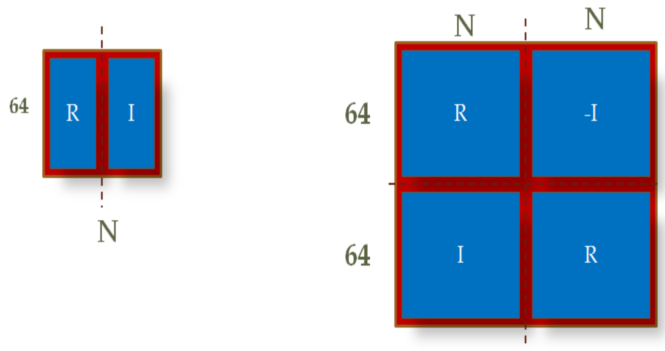


Figure 4.9: Complex SVD Solution in GSL

the complex numbers. GSL function *gsl_blas_dgemm* is a part of the CBLAS library which is used in matrix multiplication. The function *lu_invert* is used in the inversion of sigma S to get Sinv.

4.3.5 CS OFDM Receiver Design

Finally the CS receiver was constructed using the blocks built. Fig. 4.10 shows the flowgraph *ofdm_receiver_cs* with the existing USRP2 OFDM implementation. The channel estimation(CE) is taken out of the *ofdm_frame_acq* block and is implemented by the *cs_cpp_csest* block. Then finally the estimates and the symbols are combined in the *cs_cpp_sigmix* block which is *ofdm_mixer* in the given Fig. 4.10.

Algorithm 4.2 Implemented CS OFDM

```
Input: $y$;
Downsample in cs_cpp_ds:
$\ell_1$ in cs_cpp_csest:
Initalize:
$\tilde{\phi}$ as the first $\tilde{N}$ columns of $\phi$.
$I^0 \leftarrow \emptyset$ and $w_i^0 = 1, \forall i \in \{1, 2, \ldots, \tilde{N}\}$
while the stopping condition is not met, do
        $\tilde{h} \leftarrow YALL1(y, \phi)$        (4.34)
end while
$\ell_2$ in cs_cpp_pinv
        $\tilde{h} \leftarrow pinv(y, \phi)$        (4.35)
Obtain the Transfer Function:
Use the estimates in cs_cpp_sigmix
Return $\tilde{y}$
```

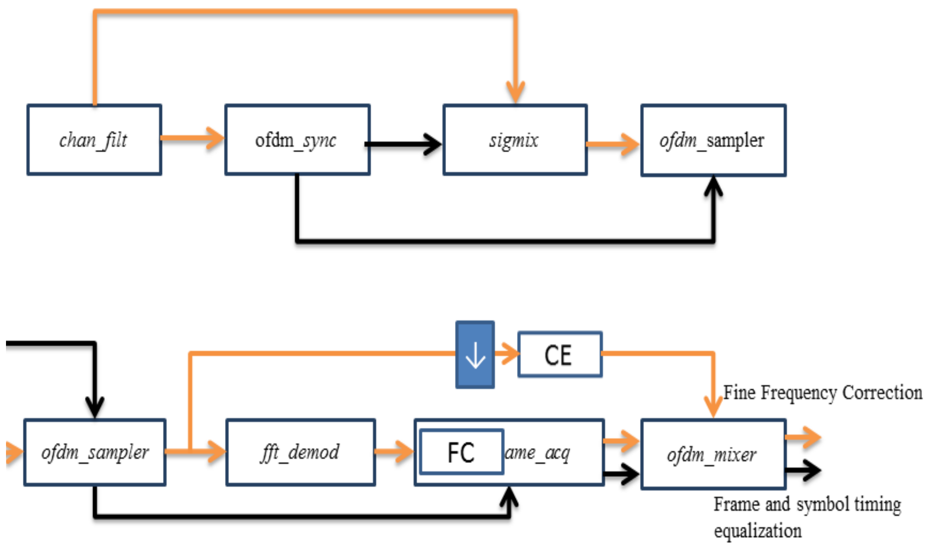


Figure 4.10: OFDM Rx with CS Estimation Flowgraph ofdm_receiver_cs.py

4.3.6 Software Stack

The compressed sensing module was built on gnuradio-3.3.0 version and followed all the guidelines mentioned in the version. A block was built based on how-to-built-a-new-block called *cs_cpp* and each of the code snippets was developed.The algorithm was divided into smaller standalone modules in C++ which were populated in the /lib directory. Each C++ modules had a work function which served as a block which took the inputs and gave the outputs. Each module also had an associated Simplified Wrapper and Interface Generator (SWIG) file which connected the python to the C++ modules. The top level SWIG files *cs_cpp.i* called all the intermediate SWIG files. A SWIG binary 'name' was generated based on the cs_cpp.i file and this was used by the python to generate cs_cpp_swig.py. These are all available in the /swig module. Inorder to check the validity of the C++ modules, QA tests can be written and these can be incorporated as a part of the Makefile, so that we check the C++ module every time we compile the *cs_cpp* module. Python files are populated in the /python folder and these files call the associated C++ modules for their use. Each of these modules can be written in a xml format which can be used for the gnuradio-companion tool. The XML files are populated in the /grc folder.

gri_fftw_MatrixA_ccc

This class was derived from the base class *gri_fft_ccc* which defines forward and backward Fourier Transforms.We use the FFTW3 libraries to built the 2 components for define the measurement matrix for the ℓ_1 solver. The measurement matrix $\mathbf{A}$ was built from the two function handles,

$$\mathbf{A}.times\text{= IFFT (FFT } (y) * (pilots)),$$

$$\mathbf{A}.trans\text{= IFFT (FFT } (y) * conj(pilots)).$$

cs_cpp_yall_fftw

This routine is used for L1 optimization. It solves the primary basis and then adds the weights from previous iteration to solve the current iteration thereby converging the results when the relative gap is significantly reduced.

cs_cpp_misc

All the misc functions available in Matlab like nnz, find, abs, norm is implemented on cs_cpp_misc.cc.

cs_cpp_pinv

This is a gsl based routine which uses SVD to solve the pseudo inverse of the finally constructed measurement matrix.

cs_cpp_ds

This routine was used for downsampling the received samples from *ofdm_sampler*.

cs_cpp_csest

This is the core block of the CS based channel estimator. In this case the input is the downsampled receiver input from *cs_cpp_ds* and the output is channel estimate. It uses the functions from *cs_cpp_yall_fftw* and *cs_cpp_misc* to get the sparse channel estimate, h. Hence this block acts like an interpolator.

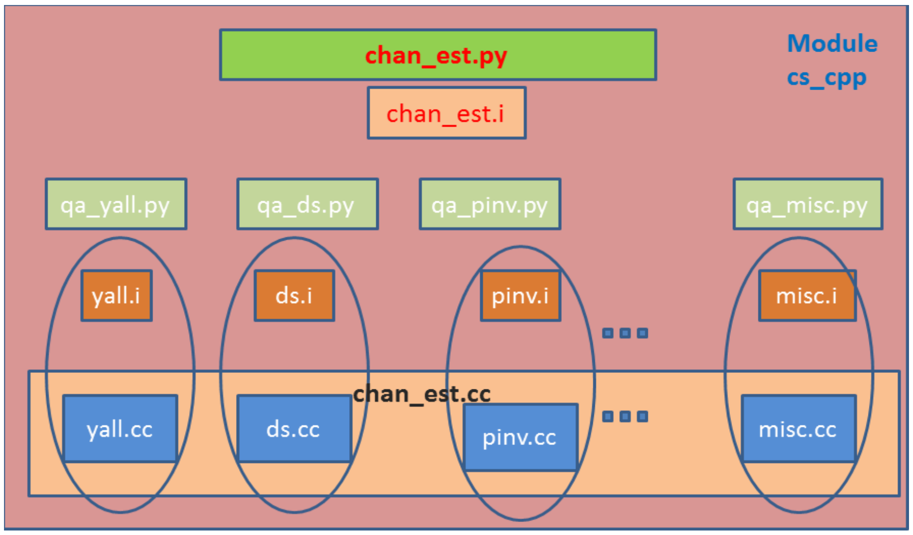


Figure 4.11: Software Stack of the CS Receiver in GNU Radio

cs_cpp_sigmix

The *cs_cpp_sigmix* was built to join the frequency compensated received samples from the *ofdm_frame_acq* block and the estimates from the *cs_cpp_csest* block. The output of the *cs_cpp_sigmix* is the final output from ofdm_receiver_cs.py

All the C++ blocks have a work method, which has output streams and input streams. The data from the input streams is taken and does the signal processing on this data. This is generally a child class of gr_sync_block or gr_block. The gr_sync_block is a derivative of the gr_block. Depending on the type of the input-output relationship, the blocks can be Sync, Decimator or Interpolator in nature. This defines a ninput_items, noutput_items and nitems. Each of the C++ block has a .i swig file which uses the shared pointer from the C++ class using the boost libraries. This is used by the Python script to reference the signal processing package. Quality assurance tests were built to test each block as a standalone routine. Test vectors were generated from MATLAB and these were used to check the integrity and the correctness of the routines. This helped in solving a lot of algorithm related issues as well as memory leaks which would get trickled to upper layers.

Chapter 5

Simulations and Discussions

In this chapter, we present the numerical simulations of the improvement in using CS in channel estimation over the traditional frequency domain channel estimation used in USRP2 in Section5.1 and also explore the practical results from the USRP2 Setup. Before moving to the experimental setup we also characterize the system in Section 5.2 to understand the upper bounds of the system.

5.1 Simulations

In this section we simulate the CS sparse recovery algorithm for channel estimation in the OFDM system and calculate the error rate performance in MATLAB. Random pilots are assigned as required in the algorithm and this is compared against the channel estimation used in the current setup. A few assumptions were taken into consideration for the simulations. The power in pilots or the preamble must be same as power in data. The number of taps in channel impulse response h was assumed to the lesser than cyclic prefix length. We are also aware that moving to time domain helps in decreasing the estimation error, although it increases the complexity. A comparison is made against the theoretical BER performance of a BPSK modulated system for the channel used.

Fig. 5.2 shows the symmetrical CS demonstration where $N = 256$ subcarriers with 128 occupied tones have been considered in the transmitter and the receiver. Here, the down sampling is done at the receiver to make use of CS, and shows a significant improvement over the traditional 256 FFT. However, this does not achieve good BER performances for the low bin FFT case. In the experimental setup, we used $N = 512$ subcarriers with 256 occupied tones.

In Fig. 5.3, we have the asymmetrical model where the transmitter send the preambles at a higher FFT(512/1024) and the receiver behaves like a normal 64 FFT. Here the receiver only looks at the downsampled samples. The Asymmetrical OFDM system has considered $N = 64$ subcarriers

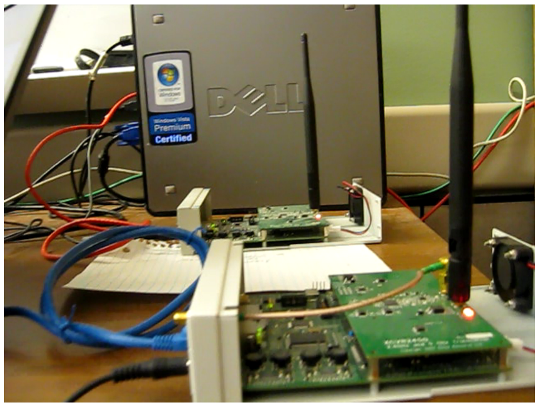


Figure 5.1: USRP2 Setup

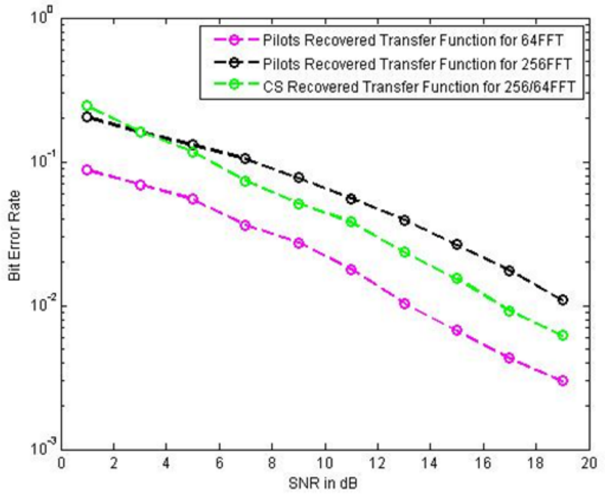


Figure 5.2: Simulations for Symmetrical CS Setup

of which 32 are the occupied tones. The preamble uses $N = 512$ subcarriers at a higher bin FFT in the same bandwidth as the data. This is attained by maintaining the occupied number of tones as 256 and following Fig. 4.4 the same bandwidth can be obtained. Hence this establishes the asymmetry during the training sequence when CS is done. There is a significant improvement with higher FFT on the transmitter since this helps to build better resolution at the receiver. However since we are downsampling a higher FFT preamble, this can introduce aliasing. This can be avoided using a different subcarrier allocation, yet conserving the bandwidth in the practical system.

Since the YALL1 package had to be customized for a real time system, simulations were also run with a custom YALL1 based ℓ_1 optimizer in Fig. 5.4. The OFDM parameters used were

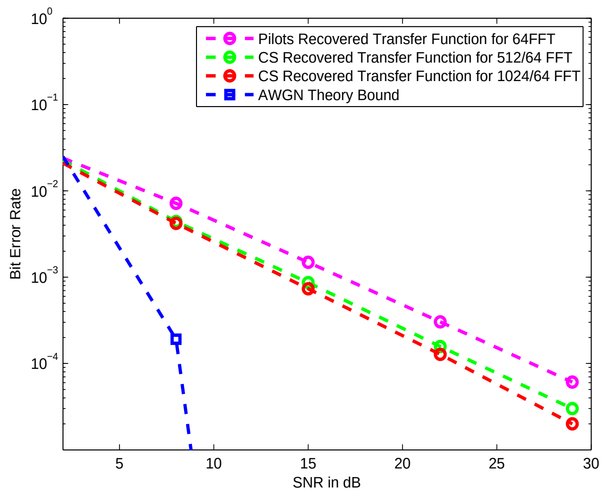


Figure 5.3: Simulations for Asymmetrical CS Setup

the same as the symmetrical model and this too shows a significant improvement over the usual channel estimation model. The tolerance for the YALL1 convergence were varied to see if there was a significant dip in performance. Even if we used a tolerance for 1e-4, the convergence was good and this could be used for the software radio implementation.

5.2 System Characterization

We had to conduct a feasability study of the experimental setup with our scenario before carrying out the experiment. This would reinstate the reliability of the system and if performance studies can be done in this system. Although the USRP2 is not intended for performance studies, it can definitely be used as a platform to prototype a new algorithm in a wireless system.

We transmitted an image and checked if changes in the TX Gain of the transmitter USRP2 would improve the SNR at the receiver and observed there was an improvement in the image quality. We used $N = 64$ subcarriers, 16 subcarriers are occupied and with a CP length of 24 subcarriers. The best way to analyze an image is check if its PSNR increases with the SNR. The PSNR was also

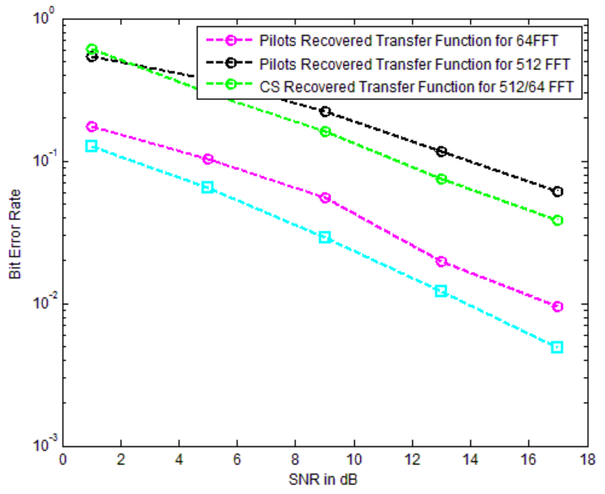


Figure 5.4: Simulations for CS Setup with Custom ℓ_1 Optimization

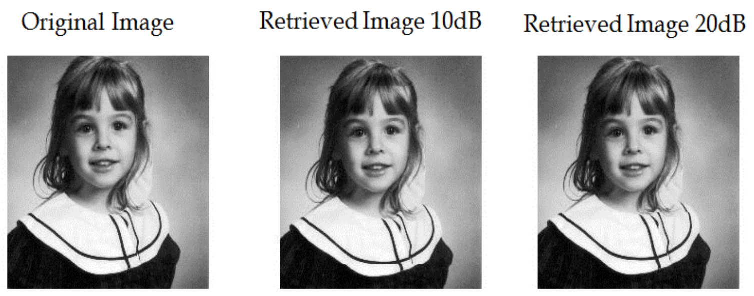


Figure 5.5: Images Obtained with the Setup at Different Tx Gains

calculated from the image using,

$$MSE = 1/m * n \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |I(i,j) - K(i,j)|^2, \tag{5.1}$$

$$PSNR = 10 * log(MAX_I{}^2/MSE). \tag{5.2}$$

As in Fig. 5.5 it is evident that a higher gain helps in getting better images. Hence PSNR improves with the iamge quality and was proved to be better in the latter case.

We studied the receiver synchronization and the preamble pattern used which have been mentioned in Section .The preambles were changed to the random pilot structure that we needed for our model and the experiments were done. Once they were through, we characterized our setup using a wired channel to simulate an AWGN scenario. The wired channel was setup with a SMA to

SMA cable. Fig. 5.1 shows the setup used for the experiments. We also tested the platform with a standard AWGN channel to prove the setup. We transmitted 1000 packets of size 1024 bytes from one USRP2 node and received it from another host. There was also a question of using Packet Error Rate versus the Bit Error Rate. However BER was found to be more reliable than PER.

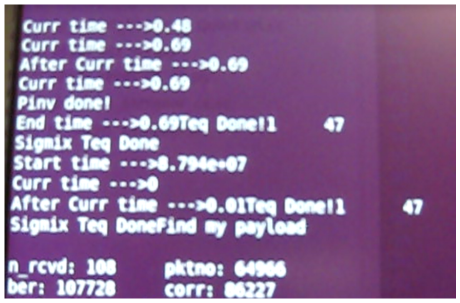
The **SNR estimation** was done in two ways. One was inserting a probe in the ofdm_frame_acq block and calculate the SNR from the received samples after the FFT. This was not very accurate and did not give us an average value for the SNR. So we reused the probe from the *mpsk_snr_probe* class. This makes use of a moving average filter which helps us obtain an average signal amplitude and average noise amplitude after the USRP2 Sink. The GRC flowgraph was used to built the python script for the testability. The average filter value y is obtained from the input x for the ith sample as,

$$y_i = 1/M \sum_{j=0}^{M-1} x_{i+j}. \tag{5.3}$$

5.3 Experimental Setup

The first action was to check if the existing blocks can be integrated along with the GNU Radio tree. Once this was done, the block was tested with test vectors from MATLAB and checked for functionality. The *cs_cpp_csest* gives out the time estimate for a downsampled signal along with the *teq_done* signal which increments when the CS is done completed entirely. Fig. 5.6 shows an entry while the CS receiver is used and the *teq_done* signal is asserted and incremented. This also shows a Start time which indicates that the CS block has been entered and the clock has started ticking for counting the amount of time taken for CS to converge. The Curr Time indicates the current time, which when exceeded suspends the current CS routine. Once the iterations are converged in the time required, we go ahead with the pseudo-inverse of $\mathbf{A_{new}}$, which is the last step of the optimization.

The block *cs_cpp_csest* was integrated in ofdm_receiver_cs.py as shown in Fig. 4.10 in Chapter 4. The estimated equalizer taps from the CS estimator block were collected from the GNU Radio



```
Curr time --->0.48
Curr time --->0.69
After Curr time --->0.69
Curr time --->0.69
Pinv done!
End time --->0.69Teq Done!1      47
Sigmix Teq Done
Start time --->8.794e+07
Curr time --->0
After Curr time --->0.01Teq Done!1       47
Sigmix Teq DoneFind my payload

n_rcvd: 108      pktno: 64966
ber: 107728      corr: 86227
```

Figure 5.6: USRP2 Receiver Packet Dump

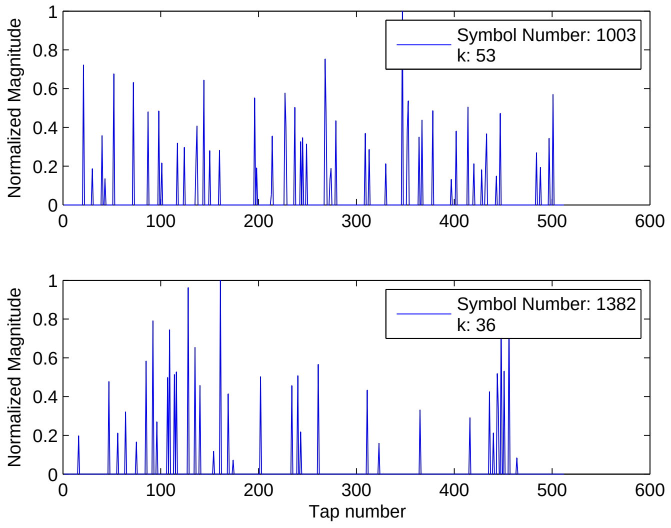


Figure 5.7: Sparse Channel Estimates With Symbol Numbers

and plotted in Fig. 5.7. We can see that the number of active taps versus the total number of multipaths in both the cases is less than 10%. This proves that the channel impulse response can be sparsified and only some of the taps provide us with useful information.

The estimated equalizer taps from the CS Estimator block along with the transfer function for the occupied frequency bins from the mixer block were collected from the GNU Radio and plotted in Fig. 5.8. The original channel estimate for the occupied frequency bins were also collected in Fig. 5.8. We can see that the number of active taps versus the total number of multipaths. The original channel estimate in the setup was calculated by dividing the known symbol with the frequency compensated received symbol in each occupied frequency bin. The absolute amplitude of this estimate is much larger than the compressive sensing estimate however we can see that the

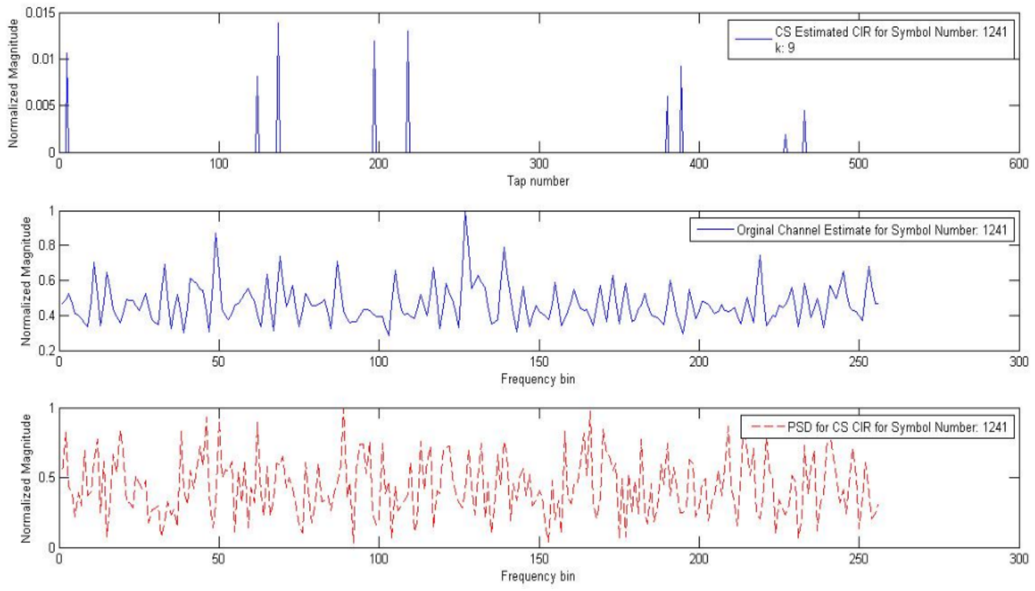


Figure 5.8: Sparse Channel Estimates for Preamble Symbols with Transfer Functions from USRP2

normalized amplitude of the CS estimated channel response follows the original channel estimate response, which means it is reconstructed.

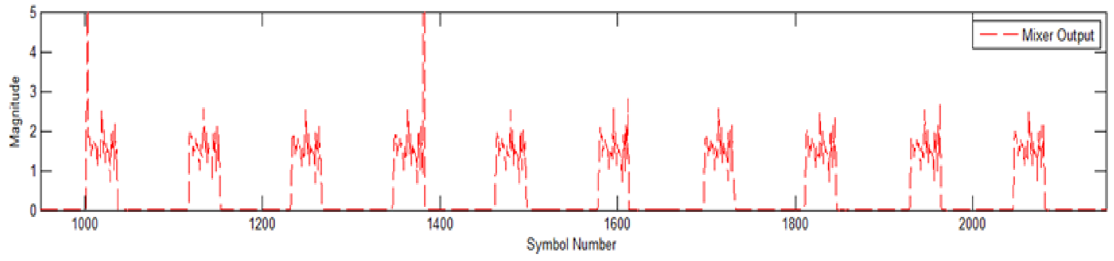


Figure 5.9: Symbol Output from cs_cpp_sigmix Block

Once the time estimates are obtained, we get the transfer function (TF) by taking the Fourier transform of h_n. This is applied to the frequency compensated symbols from *ofdm_frame_acq* block in the *cs_cpp_sigmix* block. Fig. 5.9 shows the symbols from this block while we transmit 1000 packets of 1024 bytes each. We used 256 occupied tones in this setup, so that would give us 32 symbols for each packet.

Fig. 5.10 shows the BER rates of the CS setup for $N = 512$, with 128 occupied tones and a cyclix prefix length $CP = 192$. Our primary aim of proving CS in a real time system was proved

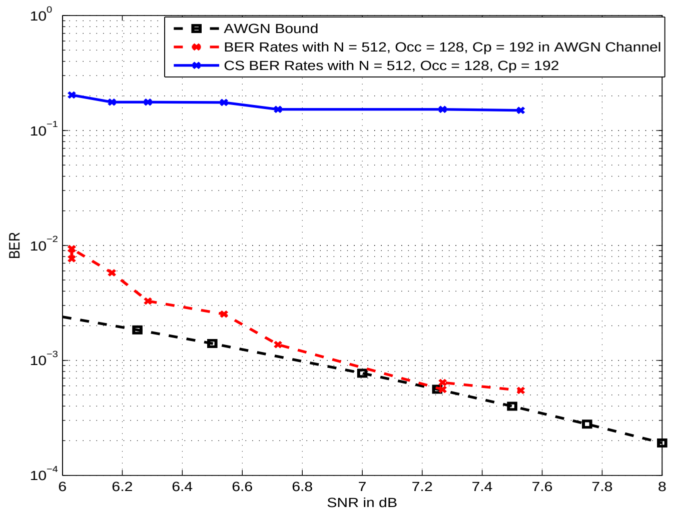


Figure 5.10: CS Estimated BER vs SNR plot

and although the theoretical bound seems further away, this can be significantly improved with higher sampling rates. The USRP2 platform is used for prototype testing and acting as a base for performance measurement platform is limited. So due to throttling of the packets in the CPU, the packet drops add to a limit in the performance. We also repeated the same setup with a wired SMA to SMA cable to prove that the setup is feasible to be used to prove the effectiveness in an AWGN channel.

Chapter 6

Conclusions and Future Work

6.1 Challenges

Every project has many obstacles and roadblocks. This research also saw some challenges most of which were worked through. Simulations are easier to built whereas translating it to a real environment needs a lot of redesign and re-engineering.

One of the major challenges was building the C++ modules in the GNU Radio framework with Python-SWIG-C++ interfaces and memory leaks. We had never built a stand alone signal processing block in our lab and we did not have the required knowledge base. So it took a lot of time to ramp up on understanding the underlying issues while constructing a signal processing package.

There were also many real time issues while running the USRP2 setup which we encountered while running the new design. Moreover there was an upgrade in the GNU Radio platform from the time we started our project and some of the new modules had to be rebuilt for the old version.

Currently the channel estimation is done in the frequency domain; however the channel estimates on all the used subcarriers should be moved to the time domain. Time domain estimation is more complex however with good simulation results we were able to debug all most of the points in the flow graph. Moreover the frequency compensation was not extended to the time equalizer(*cs_cpp_csest*) block, which could affect our throughput. Similarly we also saw phase synchronization issues emerging in the packets which curbed the BER rates. Throughput issues was also a constraint for the SDR since we are depending on the host to converge the algorithm. SDR was good for proving a concept, however there are better platforms for analyzing and comparing performances.

6.2 Future Work on CS-OFDM

Apart from building a compressive sensing based channel estimation system, there are various submodules which can be considerably improved. This section looks at an extended work from this research thesis and also the next step after the SDR implementation. One of the submodules which can be improved is the OFDM SNR estimation technique which is described in Section 6.2.1. Section 6.2.2 describes how the transition can be made into a hardware platform.

6.2.1 OFDM SNR Estimators

One of the offshoots from this research is the requirement to built a better and sturdier unbiased SNR measurement system in real time platforms. The SNR was measured using the bpsk probe, however the need for a good quality OFDM estimator is required. Currently the probe is a moving average filter and gets an estimate of the SNR based on the time domain received signal after downsampling. Although there are estimators based on the second and the fourth moments available in the GNU Radio platform, this did not show significant improvements. We would extend this work to built a SNR Estimator from the data in the *ofdm_frame_acq* which separates the occupied data tones from the total number of allocated sub carriers. SNR estimation can also be built from the FFT samples also by taking into account the samples in the frequency domain in the occupied tones along with the noise in the unoccupied tones. The estimated SNR of the OFDM platform is estimated between 6dB to 15dB. We were able to characterize our experiments in the 6dB to 8dB range. However this was not a fool proof method, so we need to significantly improve the SNR estimation method, which could calculate SNR per packet, carrier and keep track of the minor changes with time.

We stuck to the bpsk probe in our experiments, however [49] gives an account of how zero point autocorrelation method can be used for estimating SNR. We can also try to estimate SNR during the training sequence, however this will limit the use of the data symbols that we are using. As cited in [50, 51], various SNR techniques were employed in modern day systems which can influence us to redesign a novel SNR estimation methodology in our current setup. The work [52] shows the use of as simple LS estimator in frequency domain for SNR estimation in a short simulation time. As cited in [53], we see the use of blind SNR estimation. In adaptive OFDM(AOFDM), the SNR per sub carrier is calculated. However in our experiments we try to calculate the average SNR per packet. This opens the scope for more accurate and easily implementable SNR estimators in real time systems.

6.2.2 FPGA Implementations

One of the major constraints for the soft radio approach is the speed at which the data is processed. Packets are received at a rate of μsec whereas the host computer takes msec to process the packets. Instead if we analyzed the received samples in an FPGA and calculated the channel estimates in the FPGA, it would avoid host computation congestion and also packet drops. Moreover the transition from a soft radio to a hardware accelerator based estimation will increase accuracy. We can also use the onboard FPGA to receive the samples and alter them for our requirements. However this might need an integration of the timing synchronization block. The current algorithm works on time synchronized received samples to make the CS estimate.

Another approach is to implement compressed sensing in FPGA using the same flow of the algorithm. The matrix decomposition and multiplications can be done with the existing RAMs and ROMs on the FPGA platform. Optimizing matrix operations on a hardware platform help in accelerating the computation time. Memory switching technique [54] is used to route memory traffic between a functional unit and the bus master and hence overlap the I/O computations. The functional unit takes care of the basic matrix operations. This was done using a PowerPC processor and the algorithm was based on sub-matrix blocking. The increasing use of intelligent memory controller can help solve complex signal processing and mathematical problems. An insight into a platform like WARP which has existing OFDM reference designs could help us achieve this. Again better results are achieved using square matrices however this is at the cost of resources. The papers [56] and [57] are recent additions in the research community to explore VLSI architectures for compressive sensing. However if this could be used for channel estimation, it could change the way channels are analyzed and in a more practical sense. Our next aim will be to incorporate such an estimator in a CPLD or a FPGA and prove the algorithm so that it can be used in ASICs or sold as an IP.

6.3 Summary and Conclusions

This thesis looks into how wireless channel estimation can be improved. Chapter 1 looks into the impact the improvement of better estimation techniques bring in. The need for higher data rate supplemented by higher quality of information pushes us to develop system that has very little tolerance for channel estimation errors. The main contribution of this thesis are highlighted in the Chapters 3, 4 and 5. Although it took a considerable amount of time to study the platform and developing the theoretical simulations, building a real time working system was the major milestone of this research. Simulations assume ideal behavior and in practical systems, we observe that there is trade off between accuracy and complexity.

In Chapter 2, we surveyed the hardware platform that would be used in the experiments. The USRP2 is a cost effective platform that can be used to prove a new algorithm. Although performance measurements tend to be a disadvantage for this platform, this greatly help us to investigate the practical problems in a communication system. We also look into GNU Radio and the way it is structured in this chapter.

In Chapter 3 of this thesis, we surveyed the existing architecture of OFDM in a software radio and did a feasability check to see if the CS model can be applied here. This also takes us a step closer to implement more CS friendly algorithms on the wireless platforms. We also proved that as far signal sparsity is maintained and linear and non-adaptive measurements are taken, an efficient signal recovery algorithm can be used for extracting useful information.

In Chapter 4, we successfully implemented an OFDM channel estimation scheme using probing pilots with random phases, which preserves the information of channel response during the convolution and uniformly down-sampling processes. The sparse recovery algorithm for channel estimation was customized for a real time system to develop the simulations in Chapter 5. This gave us the confidence to go forward with the practical implementation and hence prove our results.

The purpose of this thesis is to built a practical compressive sensing based channel estimator in an OFDM system. We can also prove that CS can shower its advantages when higher resolution is required and also when hardware limitations affect the overall performance. This thesis also shows the importance of using SDRs to bridge the gap to develop a prototype from a theoretical model and gives us confidence to pursue various algorithms and understand their real-time advantages and limitations. This also gives access to CS and ℓ_1 implementations which can be used for various applications in USRP2 and other platforms. This also marks an entry to extending practical applications where ℓ_1 optimization can be used.

Bibliography

[1] D. Tse and P. Viswanath, *"Fundamentals of Wireless Communications,"* Cambridge University Press, May 2005.

[2] T. S. Rappaport, *"Wireless Communications: Principles and Practice,"* Pearson Education, 2009.

[3] A. Goldsmith, *"Wireless Communications,"* Cambridge University Press, 2005.

[4] D. L. Donoho, "Compressed Sensing," *IEEE Transactions on Information Theory*, Volume: 52 , Issue: 4, pp. 1289 - 1306, Apr. 2006.

[5] L. Tong, B. M. Sadler, and M. Dong, "Pilot Assisted Wireless Transmissions," *IEEE Signal Processing Magazine*, vol. 21, no. 6, pp. 12-25, Nov. 2004.

[6] L. Tong and S. Perreau, "Multichannel Blind Identification: From Subspace to Maximum Likelihood Methods," *Proceedings of the IEEE*, vol. 86, no. 10, pp. 1951-1968, Oct. 1998.

[7] G. Taubock and F. Hlawatsch, "A Compressed Sensing techniqye for OFDM Channel Estimation in Mobile Environments: Exploiting Channel Sparsity for Reducing Pilots," *IEEE International Conference on Acoustics, Speech and Signal Processing(ICASSP)*, pp. 2885 - 2888, Apr. 2008.

[8] C. R. Berger, S. Zhou, P. Willett, B. Demissie, and J. Heckenbach, "Compressed Sensing for OFDM/MIMO Radar," *Asilomar Conference on Signals, Systems and Computers*, pp. 213 - 217, Oct. 2008

[9] K. Yeo and S. Keat, "Time Domain Equalization for Underwater Acoustic OFDM Systems with Insufficient Cyclic Prefix," *OCEANS*, Sept. 2011.

[10] A. Dowler, A. Nix and J. McGeehan,"Data-derived Iterative Channel Estimation with Channel Tracking for a Mobile fourth generation wide area OFDM system," *IEEE Global Telecommunications Conference*, vol.2, pp. 804 - 808, Dec. 2003.

[11] C. Komninakis, C. Fragouli, A. Sayed, and R. Wesel, "Multi-input Multi-output Fading Channel Tracking and Equalization using Kalman estimation," *IEEE Trans. Signal Proc.*, vol. 50, no. 5, pp. 1065-1076, May 2002.

[12] N. D. Hemkumar and J.R. Cavallaro, "A Systolic VLSI Architecture for Complex SVD," *IEEE International Symposium on Circuits and Systems(ISCAS)*, vol. 3, pp. 1061 - 1064, 1992.

[13] A. Divekar and O. Ersoy, "Theory and Applications of Compressive Sensing," *ECE Technical Reports, Purdue University*, 2010.

[14] B. Le Floch, R. Halbert-Lassalle, and D. Castelain, "Digital Sound Broadcasting to Mobile Receivers," *IEEE Trans. Consumer Electron.*, vol. 35, pp. 493-503, Aug. 1989.

[15] N. Wang, G. Gui, Z. Zhang and P. Zhang, "Suboptimal Sparse Channel Estimation for Multicarrier Underwater Acoustic Communications," *International Journal of the Physical Sciences*, vol. 6, no. 25, pp. 5906-5911, Oct. 2011.

[16] J. Meng, W. Yin, Y. Li, N. T. Nguyen, and Z. Han, "Compressive Sensing Based High Resolution Channel Estimation for OFDM System," *IEEE International Conference on Communications (ICC)*, June 2011.

[17] Y. Zhang, "On Theory of Compressive Sensing via l1 Minimization: Simple Derivations and Extensions," *CAAM Technical Report*, Sep 2008.

[18] Y. Zhang, J. Yangz and W. Yin, "User's Guide for YALL1:Your ALgorithms for L1 Optimization," *CAAM Technical Report TR09-17*, ver. 1.0, June 2010.

[19] Ettus Research Product Brochure for USRP product family, www.ettus.com, Dec. 2012.

[20] GNU Radio Homepage, http://gnuradio.org/, Dec. 2012.

[21] S. Katz, "GNU Radio Companion Tutorial," SDR Project Home Page, Aug. 2011.

[22] G. Acosta, "OFDM Simulation Using Matlab," *Georgia Tech University Smart Antenna Research Laboratory Report*, Aug. 2000.

[23] T. M. Schmidl and D. C. Cox, "Robust Frequency and Timing Synchronization for OFDM," *IEEE Trans. Communications*, vol. 45, no. 12, 1997.

[24] M. S. Akram, "Pilot-based Channel Estimation in OFDM Systems," *Nokia Mobile Phones white paper*, 2007.

[25] F. Tufvesson, O. Edfors, and M. Faulkner, "Time and Frequency Synchronization for OFDM using PN-Sequence Preambles," *IEEE Proc. VTC*, pp. 2203-2207, 1999.

[26] J. Van de Beek, M. Sandell, and P. O. Borjesson, "ML Estimation of Time and Frequency Offset in OFDM Systems," *IEEE Transactions on Signal Processing*, vol. 45, no. 7, pp. 1800-1805, 1997.

[27] B. Chen, "Maximum likelihood estimation of OFDM carrier frequency offset," *IEEE Signal Processing Letters*, Issue: 4, pp. 123 - 126, Apr 2002.

[28] M. Ettus, T. W. Rondeau, and R. McGwier, "OFDM Implementation in GNU Radio," *Wireless@VT Symposium*, 2007.

[29] E. Lagunas and M. Najar, "Sparse Channel Estimation based on Compressed Sensing for Ultra Wideband Systems," *International Conference on Acoustics, Speech and Signal Processing(ICASSP)*, pp. 365 - 369, May 2011.

[30] Z. Cheng and D. Dahlhaus, "Time versus Frequency Domain Channel Estimation for OFDM Systems with Antenna Arrays," *International Conference on Signal Processing*, Beijing, China, 2002.

[31] Google Project - Ceres Solver, http://code.google.com/p/ceres-solver/, ver. 1.4.0, Nov. 2012.

[32] J. Fix, "Efficient Convolution using the Fast Fourier Transform", *Application in C++*, Finland, 2011.

[33] X. Cai and G. B. Giannakis, “Error Probability Minimizing Pilots for OFDM With M-PSK Modulation Over Rayleigh-Fading Channels,” *IEEE Transactions On Vehicular Technology*, vol. 53, no. 1, Jan 2004.

[34] Z. Taheri, M. Ardebilipour and M. A. Mohammadi,“Channel Estimation in Time and Frequency Domain in OFDM Systems,” *International Conference on Wireless Networks and Information Systems*, pp. 209-212, 2009.

[35] J. J. Van de Beek,O. Edfors, M. Sandell, S. K. Wilson and P. O. Borjesson, “On channel estimation in OFDM systems,” *IEEE Vehicular Technology Conference*, vol.2 , pp. 815 - 819, 1995.

[36] M. Hsieh and C. Wei, “Channel Estimation for OFDM Systems Based on Comb-type Pilot Arrengement in Frequency Selective Fading Channels,” *IEEE Trans. Consumer Electron.*, vol. 44, no. 1, Feb 1998.

[37] T. Liang, W. Rave and G. Fettweis, “On Preamble Length of OFDM-WLAN,” *IEEE Vehicular Technology Conference*, pp. 2291 - 2295, 2007.

[38] M. Li, J. Tan and W. Zhang, “A Channel Estimation Method Based on Frequency-Domain Pilots and Time-Domain Processing for OFDM Systems,” *IEEE Transactions on Consumer Electronics*, vol. 50 , no. 4, pp. 1049 - 1057, 2004.

[39] S. Coleri, M. Ergen, A. Puri, and A. Bahai, “Channel Estimation Techniques Based on Pilot Arrangement in OFDM Systems,” *IEEE Transactions on Broadcasting*, vol. 48, no. 3, Sep 2002.

[40] Y. Zhang, “On Theory of Compressive Sensing via l1 Minimization: Simple Derivations and Extensions,” *Rice CAAM Report*, Sept 2008.

[41] J. Tropp and A. Gilbert, “Signal recovery from random measurements via orthogonal matching pursuit,” *IEEE Trans. on Information Theory*, vol. 53, no. 12, pp. 4655-4666, Dec 2007.

[42] J. Tropp, “Just relax: convex programming methods for identifying sparse signals in noise,” *IEEE Transactions on Information Theory*, vol. 52 , no. 3, pp. 1030 - 1051, 2006.

[43] J. Romberg, “Compressive Sensing by Random Convolution,” *IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing*, pp. 137 - 140, Dec. 2007.

[44] J. Mitola III, “The Software Radio Architecture,” *IEEE Communications Magazine*, vol. 33 , no. 5, pp. 26 - 38, May 1995.

[45] J. Mitola III, “Software radios-survey, critical evaluation and future directions,” *IEEE Aerospace and Electronic Systems Magazine*, vol. 8, no. 4, pp. 25 - 36, 1992.

[46] Compressive GNU Radio Archive Network (CGRAN), https://www.cgran.org/wiki/, Nov. 2012.

[47] S. S. Chen, D. L. Donoho, and M. A. Saunders, “Atomic Decomposition by Basis Pursuit,” *SIAM Journal for Scientific Computing* , vol. 20, pp. 33-61, 1998.

[48] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers,” *Foundations and Trends in Machine Learning*, vol. 3, no. 1, Nov 2010.

[49] Y. Bhavani Shankar, J.Chandrasekhar Rao and B. Anil Babu, “An Improved SNR Estimation Approach for OFDM System,” *International Journal of Engineering Research and Applications (IJERA)*, Vol. 2, Issue 3, pp. 2561-2563, May-Jun 2012.

[50] M. Turkboylari and G. L. Stuber, “An efficient algorithm for estimating the signal-to-interference ratio in TDMA cellular systems,” *IEEE Transactions on Communications*, vol. 46, pp. 728-731, June 1998.

[51] D.-J. Shin, W. Sung, and I.-K. Kim, “Simple SNR estimation methods for QPSK modulated short bursts,” *Proceedings of IEEE Global Telecommunications Conference*, vol. 6, pp. 3644-3647, 2001.

[52] S. He and M. Torkelson, "Effective SNR Estimation in OFDM System Simulation," *IEEE Global Telecommunications Conference*, vol. 2, pp. 945 - 950, 1998.

[53] Y. Li, "Blind SNR Estimation in OFDM Systems," *International Conference on Microwave and Millimeter Wave Technology*, 2010.

[54] N. Dave, K. Fleming, M. King, M. Pellauer, M. Vijayaraghavan, "Hardware Acceleration of Matrix Multiplication on a Xilinx FPGA," *IEEE/ACM International Conference on Formal Methods and Models for Codesign*, pp. 97 - 100, Jun 2007.

[55] Y. Dou, S. Vassiliadis, G. K. Kuzmanov and G. N. Gaydadjiev, "64-bit Floating-Point FPGA Matrix Multiplication," *ACM Conference for SIGDA Field-Programmable Gate Arrays*, pp. 86-95, 2005.

[56] P. Maechler, C. Studer, D. Bellasi, A. Maleki, A. Burg, N. Felber, H. Kaeslin, and Richard G. Baraniuk, "VLSI Design of Approximate Message Passing for Signal Restoration and Compressive Sensing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, Vol. 2, No. 3, Oct 2012.

[57] J. Lu, H. Zhang, and H. Meng, "Novel Hardware Architecture of Sparse Recovery Based on FPGAs," *International Conference on Signal Processing Systems (ICSPS)*, vol. 1, pp. 302 - 306, 2010.