POLARIZATION ANGLE DIVERSITY TECHNIQUE FOR MACHINE TO MACHINE WIRELESS COMMUNICATION

A Project Report

submitted by

MUNAGAPATI VENKATA SATISH KUMAR

for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2015

THESIS CERTIFICATE

This is to certify that the thesis is titled **POLARIZATION DIVERSITY TECHNIQUE FOR MACHINE TO MACHINE WIRELESS COMMUNICATION**,submitted by **MUNAGAPATI VENKATA SATISH KUMAR**,to the Indian Institute of Technology Madras,for the award of the degree of **MASTER OF TECHNOLOGY** is a bonafide record of the research work done by him under our supervision.The contents of this thesis,in full or in parts,have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof.Radhakrishna Ganti
Research Guide
Assistant Professor
Dept.of Electrical Engineering
IIT-Madras,600036.

Prof.Devendra Jalihal
Research Guide
Professor
Dept.of Electrical Engineering
IIT-Madras,600036.

Prof.Venkatesh Ramaiyan
Research Guide
Assistant Professor
Dept.of Electrical Engineering
IIT-Madras,600036.

Prof.Ravindra David Koilpillai
Research Guide
Professor
Dept.of Electrical Engineering
IIT-Madras,600036.

ACKNOWLEDGEMENTS

I would like to acknowledge with deepest appreciation,gratitude and love of my family-my parents Munagapati Venkateswarlu and RenukaVenkateswarlu.They all kept me going,gave kind cooperation,moral support,amazing chances they've given me over the years and this project would not have been possible without them.

"I have taken efforts in the project,However it would not have been possible without the kind support and help of my research advisors Dr.Radhakrishna Ganti,Dr.Devendra Jalihal,Dr.Venkatesh Ramaiyan,Dr.Ravindra David Koilpillai.I would like to thank sincerely all of them."

I would like to specially acknowledge Dr.Radhakrishna Ganti as I was being directly and constantly guided by him. I would like to extend my thanks for his guidance and constant supervision as well as for providing necessary information regarding the project and also for his full support in completing the project.He has shown me a great concern and continual encouragement.

I Would like to thank Dr.Takai for his novel idea of introducing polarization angle diversity for converting static channels to fast fading channels.I wish to express my deepest gratitude to his effort in making us to understand the technicalities of the whole project.

A sincere thanks to my project mates Kandregula Chalapati Rao,Arunkumar S C,Walke Nilesh Uttamrao,Theeksha A P in the development and completion of the project.

I would like to thank all the professors who taught me during my days at IITM.I would like to thank all my lab mates for their support and making my stay more memorable.

ABSTRACT

In this project polarization angle diversity technique for Machine to Machine(M2M) is developed.The architecture is useful for M2M communication using line of sight and non line of sight waves(signals) that are reflected by electric and magnetic scatters.The succesfull radio transmits the signal,which is expanded by orthogonal codes on different wireless paths using different polarization directions.This approach of communication results in good resistance against outer interferences and unexpected interruption,which enables it to achieve highly-reliable communication.

This novel idea of angle diversity due to polarization was proposed by Dr.Takei from department of green mobility Hitachi,japan.

This project have conducted for several times and collected apporipate data what we have transmitted.The software used matlab,c,c++ and hardware used transmitter, receiver kits from Dr.Takei,USRP kits from Dr.Devendra Jalihal,some necessary equipment from Dr.Radhakrishna Ganti.The experiments are done and verified the results.

The experiments are conducted in:

1)Mechanics lab in IIT Madras,the environment consist of many mechanics.
2)IIT Madras CSD 309 is used to keep the kits,Computer and necessary equipment.

In this thesis, i have explained prosperously the mathamatical notations,methods and diagrams.The codes(matlab,c,c++,USRP interfacing) for USRP transmitter and receiver are also explained clearly.The hardware part shown in respective diagrams.

The detailed results have been provided with proper explanation.

The Project Team

The hitachi team that took part in the experiment had the following members:

1.Dr.Ken Takei,Chief designer and over all in-charge.

2.Mr.Aono,Engineer

3.Mr.Rethish

The IIT Madras team consist of the following members:

1.Prof.Radhakrishna Ganti

2.Prof.David Koilpillai

3.Prof.Devendra jalihal

4.Prof.Venkatesh Ramaiyan

5.Munagapati Venkata Satish Kumar

6.Theeksha A P

7.Walke Nilesh Uttamrao

8.Arunkumar S C

9.Kandregula Chalapathi Rao

Contents

List of Figures

Chapter 1

INTRODUCTION

Reliable wireless machine to machine(M2M) communication in Electromagnetic distrubed industrial environments is a scientific project plan.

The main objective of the project is to create improved technical solution for increased reliability of wireless applications in industrial environments to create this goal the following four industrial and scientific problems are addressed:

1)Characterization of the total electromagnetic inteference in selected frequancy bands for wireless applications in industrial environments focus on measurements and model development both interference signals and wave propagation properties are investigated.

2)Assessment of the sensitivity of present wireless technologies to the industrial interference environments characterized.

3)Improvement of selected present communication technologies in order to increase the robustness again the industrial interference environments characterised.

4)To achieve angle divesity.

Till today most of the industres use wired communication with large electromagnetic scatteres between machine to machine for reliability.The wireless communication is could be realiable because highly non-line-of site and lot of intereferane scattering factors during propagation of channel in industry.

In general fading is very slow and fading coefficient will not effect that much.
This will be observed in pedestrians(e.g,. pagers,cellphones,e.t.c.,) machine to machine communication in industries or slow moving vehicle communication applications where a very small fading rate(Doppler spread) is usually encountered.

Fading environment in industry is static and slow.one's signal is deep fade then that signal would continue to be indeep fade for a long time.In wired communication one of the disadvantage is,if wire may be damaged.It is diffcult to resolve that kind of issues.

In this approch,the known signal is get converted the very slow fading channel to fast fading channel to fast fadig channel.Therefore signal avoids contineous deep fades resulting low Bit Error Rates(BER).

The wireless M2M comminication was studed theorotically by Hitachi then after a contract was established between the Hitachi Japan and Indian Institute of Technology Madras team for evaluation of reliability and feasibility of the technique in wireless M2M communication.

1.1 Statement of the project

The project is to implement forced fast fading environment at the receiver for improving the effect of channel coding on bit error rate(BER) performance in a very slow multipath fading environment.This is because when the signal is in deep fade.Which is below the threshold level resulting not a waterfall BER curve.

This project is generating forced fast fading by polarization angle diversity.which is happend by two perpendicular antennas because of this channel has multiple fades.

1.2 Problems introducing multipath fading without polarization diversity

Multipath fading severely degrades BER performance i.e.,the signal strength decreases when it reaches to the receiver via multiple paths.This effect can be avoided by diversity reception and channel coding.In channel coding techniques each bit's envelope is effected by independent fading otherwise,the decoded BER performance may be degraded by burst errors before decoding therefor bit inter-leaving is often applied in order to randomize bursty errors.

1.3 Organization of the thesis

Hitachi has proposed a new radio with circular polarization as an effective solution for M2M communication for process industry such as oil refinery,thermal power plants,cements plants,slow moving environment where Doppler frequency is very low.The thesis describes the following works carried out in the project.

In Chapter 2: Describing the project in proposed mathematical form.Mathematical notation of signals (transmitter and receiver signals),chip sequences.

In Chapter 3: Diversity techniques,Bit spreading techniques.

In Chapter 4: Transmitter and receiver structures,Electromagnatic wave propagation in free space.

In Chapter 5: Algorithm explanation.

In Chapter 6: USRP Implementation.

In Chapter 7: Analysis of the data collected is explained.

In Chapter 8: It describes the future work that can be done.

The next part consist of the appendix describing the transmitter,receiver
MATLAB,C,C++,USRP Interfacing codes.

Chapter 2

Mathematical Analysis of the project

Introducing fast fading at the receiver can be done by many ways.They are space,time,frequancy diversities techniques.The new polarization angle diversity is also an one of the technique to introduce fast fading.

At the transmitter same data is transmitted on both vertical and horizontal antennas with different carriers which is given below.The actual data is a(t) transmitted on vertical and horizontal antennas are $S_v(t)$ and $S_h(t)$ respectively given as

$$S_v(t) = a(t)\ cosw_1t\ cosw_ct$$

$$S_h(t) = a(t)\ sinw_1t\ cosw_ct$$

where $w_1 = 156.25KHz$ and $w_c = 429.84375MHz$
These two antennas are perpendicular to each other as shown in the figure 2.1.

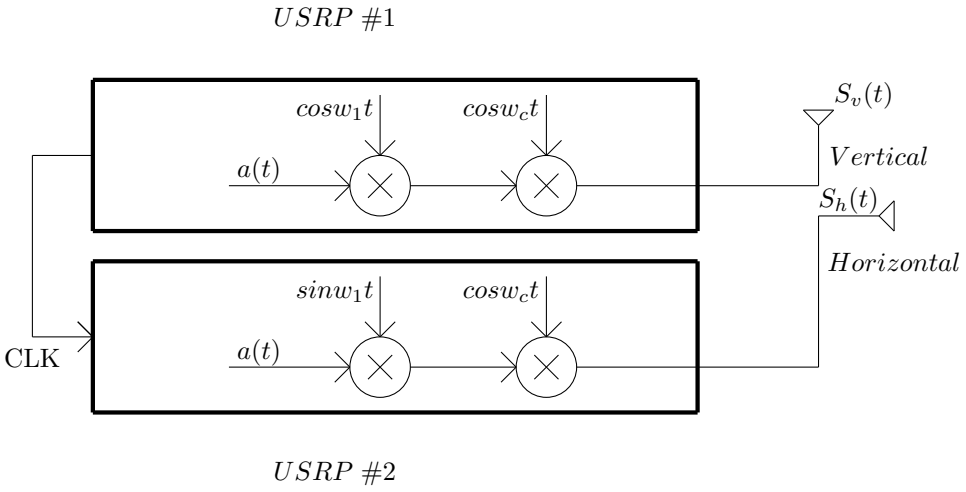


Figure 2.1: Block diagram of the USRP's transmitter

$S_v(t)$ and $S_h(t)$ are the signals at the output of the vertical and horizontal antennas.

The propagation signal is explained with the help of 3-dimensional axis.The transmitted vertical and horizontal signals are summed vectorially such that its projection at time t on the x-axis is $S_v(t)$ and on

the y-axis $S_h(t)$.Thus the propagating signal is $a(t)cosw_c(t)$.This is shown in figure 2.2.

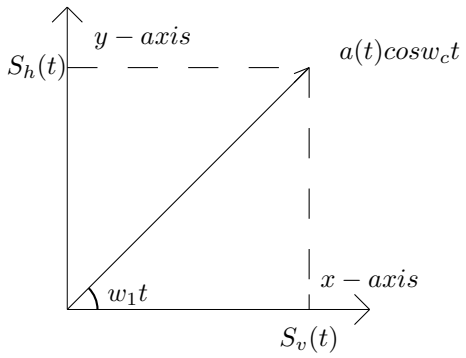


Figure 2.2: Projection of transmitted signal on to x-axis and y-axis

The information $a(t)$ is formed by chip sequence which is shown below.

$$C = \begin{pmatrix} -1 & -1 & 1 & 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 & 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 \end{pmatrix}$$

where c_1 ,c_2 ,c_3 and c_4 refer to the four rows of above matrix.$a(t)$ is formed by arranging c_1 ,c_2 ,c_3 andc_4 for one half cycle of w_1 and repeating it in the other half and eventually multiplying the entire chip sequence with the sign of the bit,which is to be transmitted over that cycle of w_1.This is performed for a every bit.

The chip sequence is formed by taking pseudo-noise(PN-7) sequences.The 7 chips are put in first row,second is formed by circular right shifting two chips from above row.Third row is formed by circular right shifting two chips from above row.The last row is formed by circular right shifting one chip.For all this rows putting -1 in last 8th coloum.one cycle of w_1 signal spans 20μ sec and thus $a(t)$ will have a band with of 3.25MHz.

The propagation signal experiences narrow band fading due to scattering in the presence of obstructions and the received signal $r(t)$ can be written as

$$r(t) = h(t)a(t)cos(w_ct + \theta(t))$$

where h(t) represents the narrow-band time-varing channel coefficient and $\theta(t)$ is the phase in the received signal due to the propagation time delay.This is received by a combination of vertical-horizantal

antennas and a splitter as shown in figure 2.3.

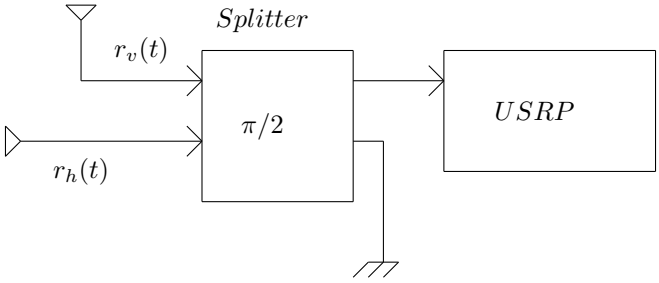


Figure 2.3: Receiver with two perpendicular antennas and a splitter

The narrow band fading and signal tap coefficient is assumed in the project.The distance of propagation being considered(less than 100mts) are so small that all the signigicant multipaths have approximatley the same delay,that of the line-of-sight(LOS) path.

The received signal $r(t)$ is projected on to vertical and horizontal antennas at the receiver generating $r_v(t)$ and $r_h(t)$ respectively as shown in figure 2.4.

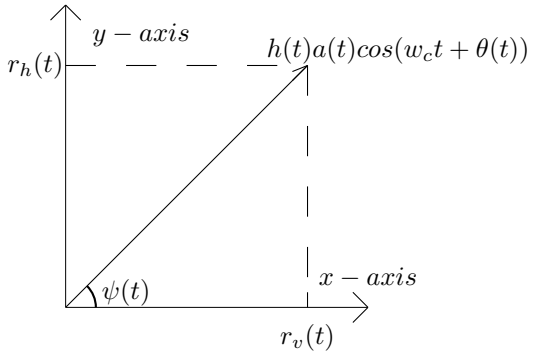


Figure 2.4: Projection of transmitted signal on to x-and y-axis

$r_v(t)$ and $r_h(t)$ are given as:

$$r_v(t) = h(t)a(t)cos\psi(t)cos(w_ct+\theta(t))$$
$$r_h(t) = h(t)a(t)sin\psi(t)cos(w_ct+\theta(t))$$
$$\text{and } \psi(t) = w_1t+\phi(t)$$

where $\phi(t)$ is the change in angle of the received signal vector due to multipath propagation of the signal and reflections by electromagnetic scatteres due to splitter $r_h(t)$ undergoes a 90deg phase shift and the phase shifted signal is added to $r_v(t)$.The resulting signal $q(t)$ that is

$$q(t) = \underbrace{h(t)a(t)cos\psi(t)cos(w_c t + \theta)}_{r_v(t)} - \underbrace{h(t)a(t)sin\psi(t)sin(w_c t + \theta)}_{r_h(t) with\ 90\ deg\ phaseshift\ in\ the\ carrier\ w_c}$$

The equivalent baseband signal $\tilde{q}(t)$ is given by

$$\tilde{q}(t)=h(t)a(t)exp(j\psi(t))= h(t)a(t)exp(jw_1 t + \phi(t))$$

After having $\tilde{q}(t)$, detected $a(t)$ in the presence of fading h(t).The channel fade coefficient h(t) can be estimated using a training sequence of bits in each frame as preamble and the chip sequence,$a(t)$ can be eventually extracted.

The channel is assumed to be narrow-band fading single tap channel and time-invariant over a transmitted frame,which has 40 bits,in the expermients.

Chapter 3

Diversity Techniques Vs Polarization Angle Diversity

3.1 Diversity techniques

Orthogonal Transmit Diversity(OTD),Time Switched Transmit Diversity(TSTD),
Selection Transmit Diversity(STD).

In OTD different orthogonal codes are used to transmit same data to get diversity.Here not using orthogonal codes in an efficient manner.In TSTD over one complete time period,some amount of time is used to transmit data from one antenna and remaning time from another antenna.Here also inefficient use of time.In STD one-bit antenna selection is used to select antenna if there are two antennas.In case if space diversity for more number of antennas requires more bits some how it is fine diversity technique for finite number of antennas.

This paper clearly explains innovated diversity technique.For this technique implementation here considering Binary Phase Shift Keying(BPSK) information bits(+1 and -1).In this technique information bits spreading is used to avoid busty errors.It actually spread one bit into multiple chips as shown in the figure 3.1.

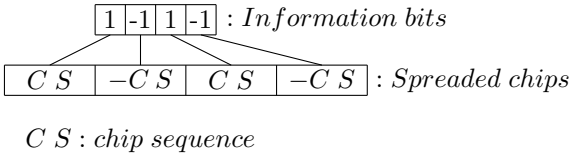
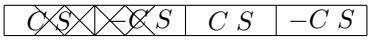


Figure 3.1: Information bits spreaded by spreading chip sequence

If the channel is slow fading or fast fading there might be a case that the transmitted frame affect at certain duration abruptly.With this technique if the information bits is spreaded by chip sequence,effected part at receiver can also be retrieved with the help of remaining un-affected part.some times there would be a little chances of total spread sequence affected,this cause results loss of one information bit.This loss of one information bit will contribute less probability of error in calculating Bit Error Rate(BER).If chip sequence length is high then it is difficult to lose even one information bit also. This is the good advantage of bit spreading.

The figure 3.2 explains effected part of first information bits last-most spreaded chip sequence and second information bit first-most spreaded chip sequence.Both first and second information bits can be



$C\ S$: *chip sequence*

Figure 3.2: Effected part of the spreading chip sequence in the channel

retrived with the help of remaining un-effected part of first most spreaded chip sequence of first bit and last most spreaded chip sequence of second bit respectively.

Each information bit is multiplied by chip sequence which is spreaded by 360 degrees as shown in figure 3.3.The main issuse of spreading it takes some more time during the transmission at transmitter and decoding at the receciver.

3.2 Spreading sequance

It should cover total 360deg.If one took 180deg spreaded sequence then that 180deg only should repeat one more to make 360deg to complete.One more thing to notice in the spreading code is if one spreading code is divided into some equal parts they all should be orthogonal to each other.

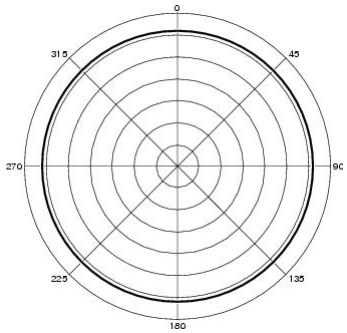


Figure 3.3: 360 degrees spreaded chip sequence

3.3 Bit Spreading

Each bit is spreaded into more chips to avoid loss of bits incase of busty errors during transmission more than a certain chips lost then the data could not be correctly decoded at the receiver.If channel conditions are not good then channel distrubs transmitted information from and causes high BER.To avoid this we have to reduce data rate.

Each information bits is taken to be spreaded by or certain no.of chips such that the contineous information data stream is spreaded into consecutive chips sequences.The chip sequence is extraced during decoding at receiver.

During without bits spreading by chip sequence if the channel is in slow fading then it causes high

BER.Because if having bit spreading by chipsequence the higher BER can be avoided.

Using bits spreading the receiver reassamble the data or if necessary request the part of the data that is unable to recover by improving the orthogonality between the chip sequence codes and preamble interpolation.There will be a significant improvement on error performance.

Chapter 4

Polarization Angle Diversity Real Time Structure and Electronmagnatic Wave propagation

4.1 Polarization Angle Diversity Real Time Structure

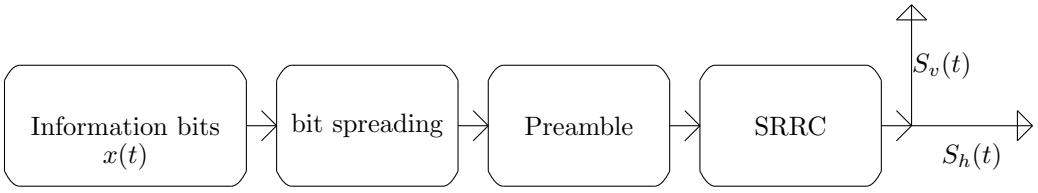


Figure 4.1: Transmitter Block Diagram

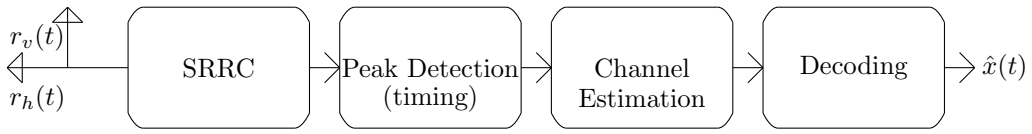


Figure 4.2: Receiver Block Diagram

The transmitter and receiver block structures as shown in above figures 4.1 and 4.2.

4.2 Transmitter

The block contain in the transmitter are information bits generation block,chip sequence block,addition of preamble block,Square Root Raised Cosine (SRRC) pulse block,two antennas horizontal and vertical.

4.2.1 Information bits block:

It consist of source bits given by the user.This source bits can be any length.In the experiment our team took 40 bits.

4.2.2 Spreading block:

This block consist of 64 chips such that when first 8 chips dot product with next 8 chips should produce zero.This has to satisfy first four 8 chip sequences.These first four 8 chip sequences only has to repeat for the next four 8 chip sequences.Therefore totally 64 chips will be formed.This 64 chip sequence should multiply with each information bit.The resultant sequence should pass through preamble block.

4.2.3 Preamble block:

This block consist of a PN sequence such that it has to satisfy a propety when PN sequence multiply with the same PN sequence from first position it should give a maximum value.If it multiply with second or any other position it should give very less value compaired to maximum value.This PN sequence is added to input data of this block then it is send to SRRC block.

4.2.4 SRRC block:

This block has Square Root Raised Cosine pulse of length 32 samples.This block upsamples data by a factor 4 of what it gets and convolved with SRRC pulse.This data is transmitted into free space via two perpendicular antennas.

4.3 Receiver

The blocks contain in the receiver are SRRC block,Peak detection block,channel estimation block,decoder block and extra vertical and horizontal antennas.
The both antennas receives vector summed transmitted data.

4.3.1 SRRC block:

The received data is passed through this block to cancle the pulsing effect.It acts as a matched filter.

4.3.2 Peak detection(timing)

The preamble used in the transmitter is interpolated here.This interpolated preamble and received data are used in cauchy schwarz inequality to calculate peak.This block gives starting point of frame.

4.3.3 Channel Estimation:

From starting point of frame preamble is extracted.This preamble is divided with actual preamble used in transmitter then we will get the channel response.

4.3.4 Decoding block:

After channel estimation the effect of channel on data will be removed.After this data is decoded with the help of chip sequence used in the tranmitter.

4.4 Wave Propagation

The wave is propagate in the medium with the help of electric(E) and magnetic(M) components which are perpendicular to each other and also perpendicular to direction of propagation.Same information is transmitted through both E and H components.If from one perpendicular antenna both E and H components have same magnitude(M),frequency and angle between them is 90 degrees then it is called as circular polarization.It is because of constant circular envelope at every time as shown in figure 4.3.

$$E_x = E_0 \sin(wt)$$

$$H_x = E_0 \cos(wt)$$

$$M = \sqrt{(E_x)^2 + (H_x)^2} = E_0$$

If one perpendicular antenna with different magnitudes(M1,M2) of both E and H in most of the time i.e., both magnitudes are same only when $2\pi f_2 t = 45(or)225$ degrees,each having frequency(carrier f_1,beating f_2) and angle is between them is 90 degrees then it is called as rotating polarization.Which is shown in figures 4.4,4.5,4.6.

$$E_x = E_0 \sin(2\pi f_1 t) \sin(2\pi f_2 t)$$

$$H_x = E_0 \sin(2\pi f_1 t) \cos(2\pi f_2 t)$$

$$M1 =\mid E_x \mid=\mid E_0 \sin(2\pi f_1 t) \sin(2\pi f_2 t) \mid= E_0$$

$$M2 =\mid H_x \mid=\mid E_0 \sin(2\pi f_1 t) \cos(2\pi f_2 t) \mid= E_0$$

Amplitude of rotating polarized wave is

$$A = \sqrt{(E_x)^2 + (H_x)^2} = E_0 \sin(2\pi f_1 t)$$

Because of $\sin(2\pi f_1 t)$, amplitude will vary from $-E_0$ to $+E_0$ with in the radius(E_0) of the circle.

In this popagation one information bit is spreaded through out 360 degrees by spreading sequence.

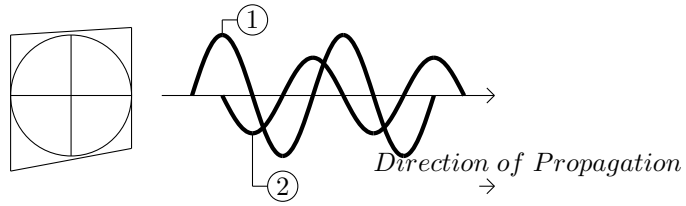


Figure 4.3: Circular Wave Propagation 1: Electrical Wave Propagation,2: Magnetic Wave Propagation

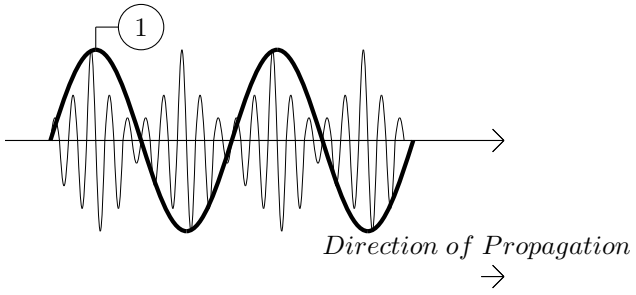


Figure 4.4: Rotating Propagation Wave 1: Electrical Wave Propagation

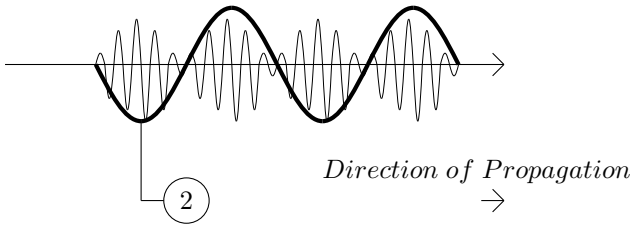


Figure 4.5: Rotating Propagation Wave 2: Magnetic Wave Propagation

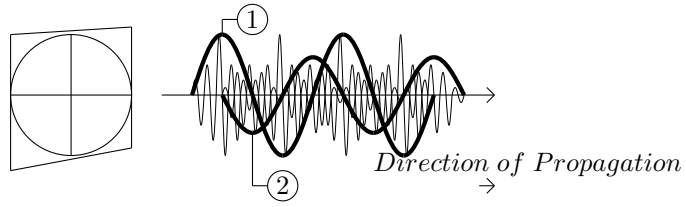


Figure 4.6: Rotating Propagation Wave 1: Electrical Wave Propagation,2: Magnetic Wave Propagation

Chapter 5

Algorithm

5.1 Transmitter part

Information bits are generated randomly.These bits are multiplied by 64 length chip sequence.Now taking perfect preamble such that when it is auto-correlated should give only one peak at first position.Added the preamble with data multipled chip sequence structure.Up-sample the entire sequence with fixed up sampled factor.Multiply the this sequence with constant frequency exponent i.e.,transmitting frequency.Now it would become a frame.If this process occur multiple times then it would call as successive frames.

5.1.1 Frame Structure

The information to be sent is of length 40 bits. Each bit is spread by a 64 length chip sequence($c_1, c_2, ...c_{64}$). A 256 length PN sequence is used as the preamble to aid in frame synchronization and channel estimation. The frame to be transmitted looks similar to the one shown in figure 5.1

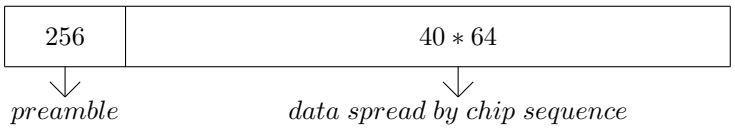


Figure 5.1: Frame Structure

This frame is appropriately upsampled and pulse shaped before being sent in the air. The up sampling factor used here is 4.

5.1.2 Channel

The channel coefficients $h_1, h_2, ..., h_{64}$ is assumed to be periodic with period 64 and each entry is iid Gaussian.Each information bit is spread with a 64 bit chip sequence.Essentially, each chip is affected by a unique channel coefficient as shown in figure 5.2.This creats to the worst case scenario where there is no correlation between the channel coefficients.Instead of interpolating channel 'h' to the appropriate oversampling factor(4),a 256 length h is generated in which each entry is iid Gaussian.A phase,uniformly chosen between 0 and 2π and constant over the complete frame adds a constant phase shift to the entire frame.

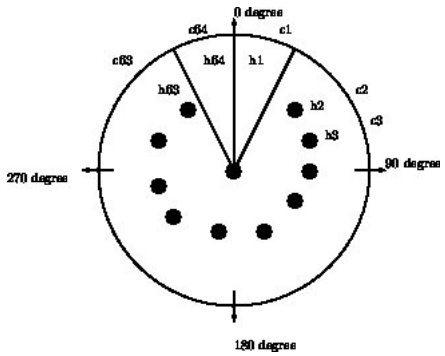


Figure 5.2: A bit affected by channel

Channel is a rayleigh,with real and complex parts.The channel is varying for each chip.Each spreading chip would multiply with rayleigh channel and doing this periodically for every information bit.From this one can say information bit is facing fixed and constant fading channel i.e.,Gaussion.It is in a way that each first chip of every information bit is facing same fading unil it received by the receiver.Like this remaining chips also faces the same fade until they go to the receiver.

5.2 Synchronization part

At the receiver the received signal is passed through Square Root Raised Cosine filter(Matched filter).Before going to timing part.There are some techniques which already present for calculating timing.They are schmidl-cox,cross-correlation,cross-correlation in frequency domain.

Schmidl-cox : This method is used in OFDM mostly.In this experiment timing part schmidl-cox method is tried but not get appropriate results.Because of polarization at the transmitter.

Cross-Correlation : In this method preamble cross correlated with received sequence.It is giving exact peak detection,but not all the time only when noise and channel are absent.

Cross-Correlation in frequency domain : This process started with great enthusiasm, thinking that it gives answer.Here FFT and IFFT is used i.e., FFT to preamble and received data.Multiplying both in frequency domain and doing IFFT to the resultant sequence.This also results same as cross correlation in time domain.

Timing technique would start calculating peak from the first chip of received sequence,it involves preamble interpolation and cauchy schwarz inequality.

Preamble interpolation : Here clearly stats how peak has to detect in rayleigh or fast fading channel.

This preamble that is used in the transmitter would be used in the receiver by linearly interpolating it.

Linearly interpolation means,if the preamble having bits like
$b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, ...(and\ so\ on)$ then during interpolation by a factor of 4 there would be three samples in between $b1$ and $b2$ they will form like this (0.75*$b1$)+(0.25*$b2$),(0.5*$b1$)+(0.5*$b2$),(0.25*$b1$)+(0.75*$b2$). The total 5 values from $b1$ and $b2$ after interpolation are
$b1$,0.75$b1$+0.25$b2$,0.5$b1$+0.5$b2$,0.25$b1$+0.75$b2$,$b2$

Cauchy Schwarz Inequality : It is explained with the help of below formula.

$$| XY |^2 \leq | X |^2 | Y |^2$$

For timing part currently 64 length preamble is using.Analysis is done for 32,16,8,4 bits length preambles also in the experiment.Till 32bit length preamble it is working properly.8bit,4bit not giveing good results.Here one novel idea is channel is estimated with the help of one bit(1) is added infront of information bits and chip sequence at the receiver.

From the peak position on-wards as shown in figure 5.3 start down sampling by an up-sampling factor used in the transmitter.

After down sampling, extract from first chip to till the length of preamble in down sampled sequence.With

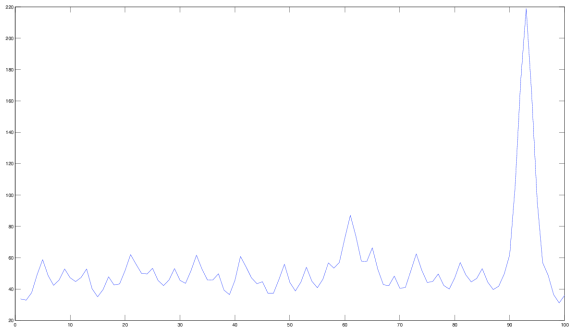


Figure 5.3: Peak with the above algorithm

the help of this one can calculate the channel response(channel estimation).

This channel response is used to remove effect of channel from the remaining down sampled sequence.Now decoding of data implies that re-multipling spreading chip sequence with remaning down sampled data by the keeping threshold as zero.Finally calculating Bit Error Rate(BER) with the help of input and decoded data.

The result as shown in figures 5.4,5.5.

The 64 length PN sequence is appropriately interpolated with an over sampling factor of 4 at the receiver. Thus, the preamble used at the receiver is of the length 256. The received signal is seen in blocks of 256 samples and the following operation is performed.

$$mag[i] = \sum_{i=1}^{64} \frac{((r(i)*p(i))+(r(i+64)*p(i+64))+(r(i+128)*p(i+128))+(r(i+192)*p(i+192)))^2}{(abs(r(i)^2)+abs(r(i+64)^2)+abs(r(i+128)^2)+abs(r(i+192)^2))*((p(i)^2)+(p(i+64)^2)+(p(i+128)^2)+(p(i+192)^2))} \quad (5.1)$$

Here $r(i)$ refers to the received samples and $p(i)$ refers to the interpolated preamble at the receiver. The values are correlated in this fashion because the channel after oversampling by the appropriate factor of 4 becomes periodic with length 256. So,each set of 256 samples is faded similarly. Combining the samples that have a similar fade and normalizing appropriately yields a peak to denote the frame start. The algorithm is pictorially shown in figure 5.6

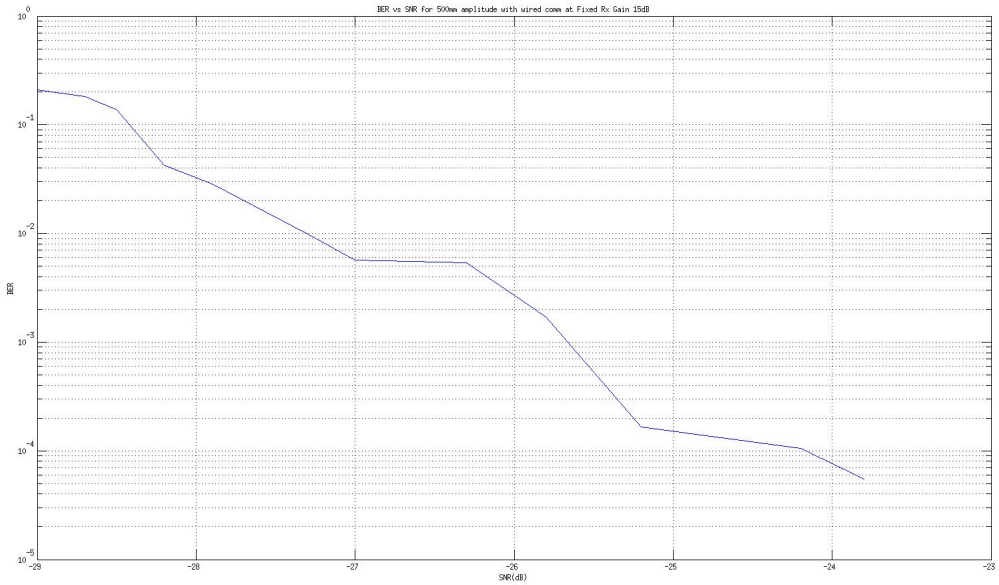


Figure 5.4: BER curve

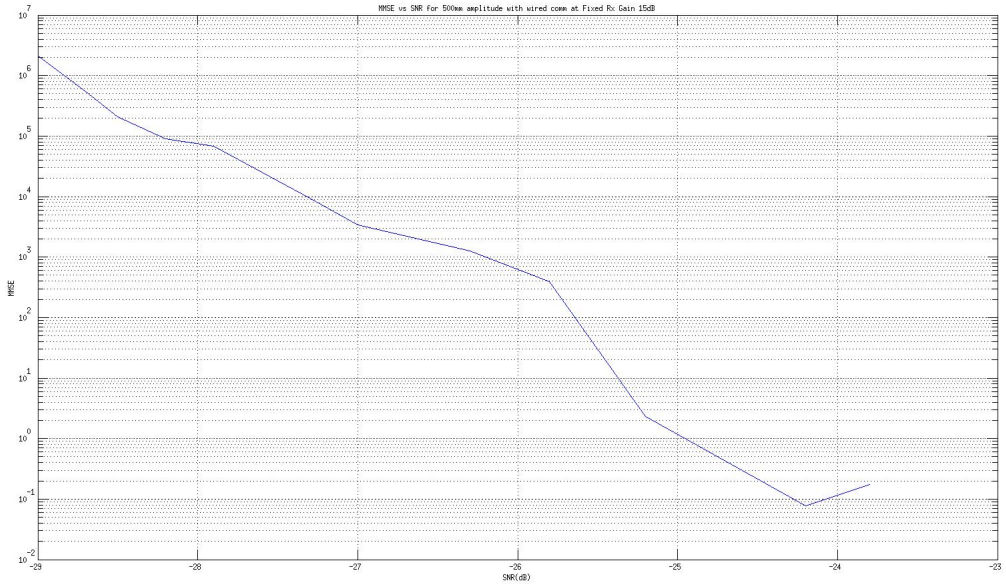


Figure 5.5: MMSE

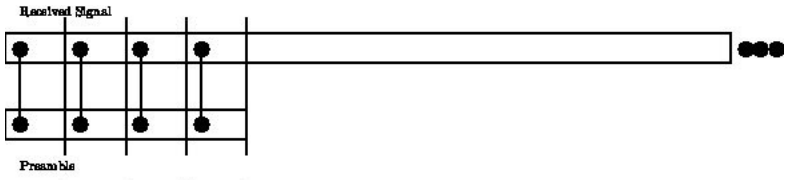


Figure 5.6: Frame Synchronization

Chapter 6

USRP Implementation

6.1 Transmitter

Transmission is done using two USRPs connected together by a MIMO cable and an external clock source. One of the USRP is given a center frequency of 430 MHz and the other one is given a center frequency of 429.6875 MHz. The same signal $a[n]$ is sent through both the USRPs. Essentially, the transmitted signal has a frequency of propagation given by $\frac{(430+429.6875)MHz}{2} = 429.84375MHz$ and frequency of rotation given by $\frac{(430-429.6875)MHz}{2} = 0.15625MHz$. Large send buffers tend to decrease transmit performance.The time taken by the device to populate a packet is proportional to the sample rate.Therefore, to improve receive latency, smaller buffer size is configured. Here the buffer size used is 1000 samples per buffer.The transmit gain can be adjusted from 0dB to 25dB.The block diagram of the transmitter is shown in figure 6.1

The bandwidth of the spectrum is $1.666MHz$ which is given by $\frac{10^7}{4\times(1+0.5)}$ where 10^7 is the sampling rate,

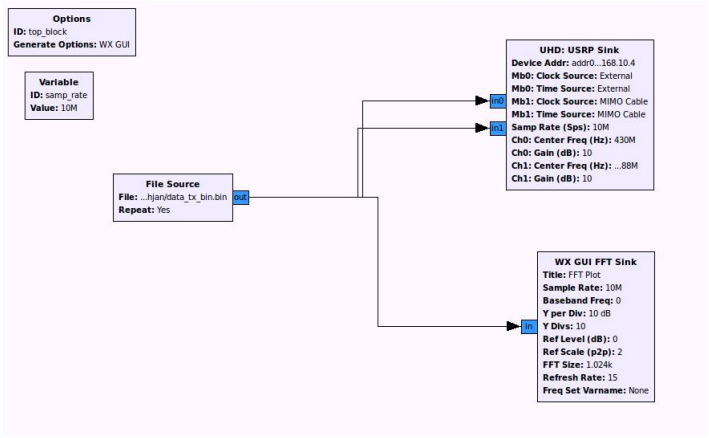


Figure 6.1: GRC Block Diagram of Transmitter

4 is the over sampling factor and 0.5 is the roll off factor of the pulse shaping filter. The transmitted spectrum is shown in figure 6.2

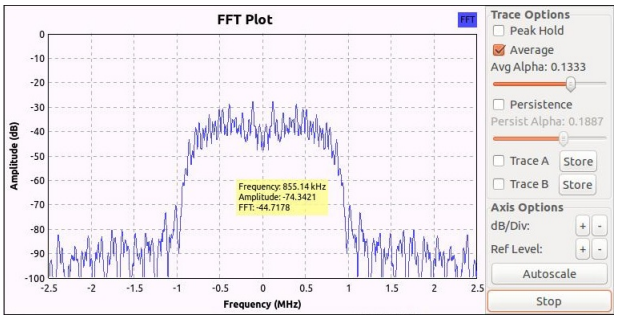


Figure 6.2: Transmitted Spectrum

6.2 Channel

The channel between the transmitter and receiver is captured at the receiver with the help of channel estimation at the receiver. The magnitude, real part and the imaginary parts of the channel are shown in figures 6.3,6.4,6.5 respectively. The channel h_1 to h_{64} has 8 periods. In other words, our assumption of h_1 to h_{64} being iid Gaussian can be thought of as the worst case scenario. The actual channel has a pattern with which it varies and is also periodic with period 8 instead of our assumption of 64.

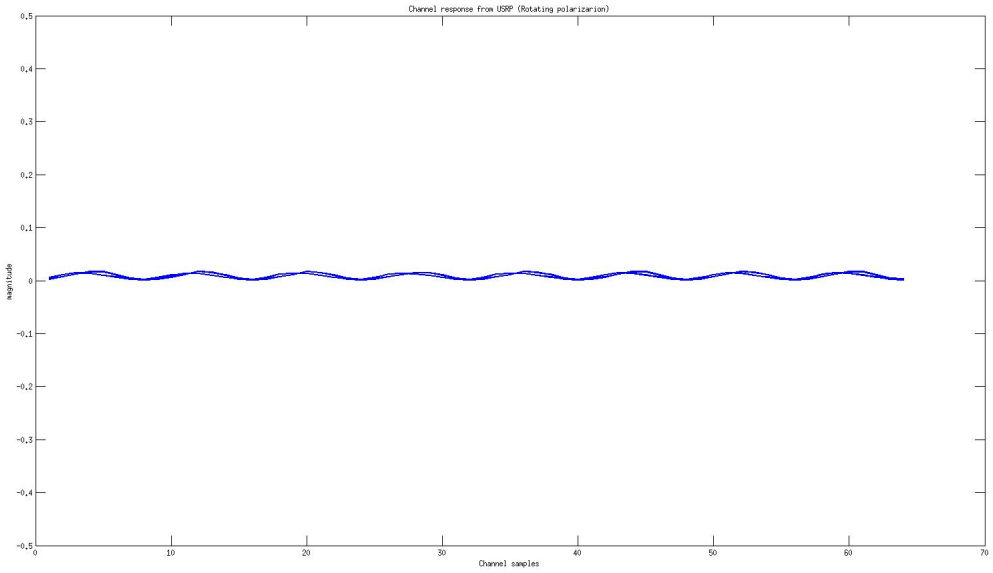


Figure 6.3: Channel coefficients - USRP

6.3 Receiver

The data transmitted over the air is received with the help of USRP. The center frequency of the receiver is set to $430MHz$.It is also connected to the same external clock so that the clocks of the transmitter and receiver are synchronized.The data received is written in a file for offline processing.Around 10^7 samples are stored to aid us in processing multiple frames and calculating the BER.The block diagram of the receiver is shown in figure 6.6.

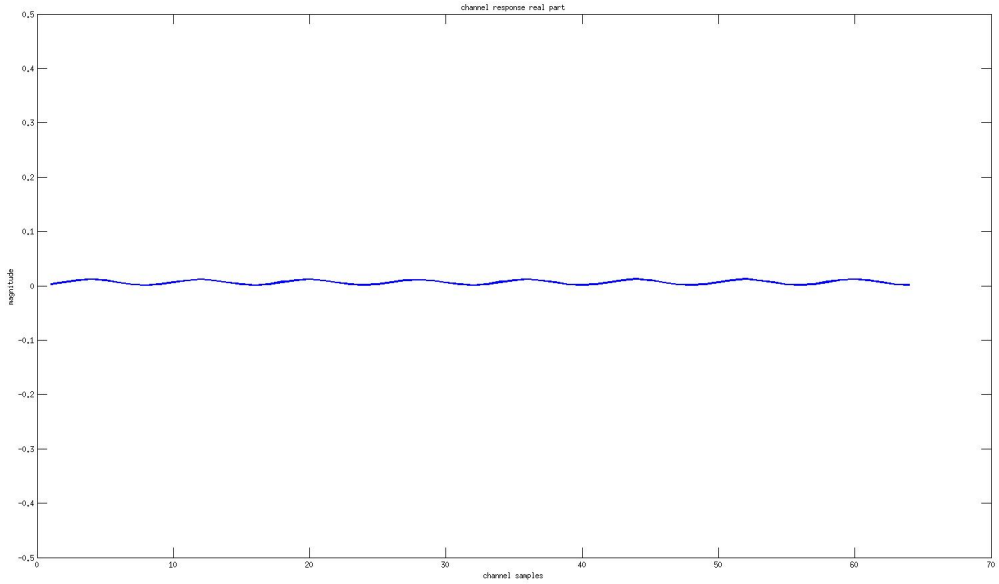


Figure 6.4: Channel coefficients - Real Part

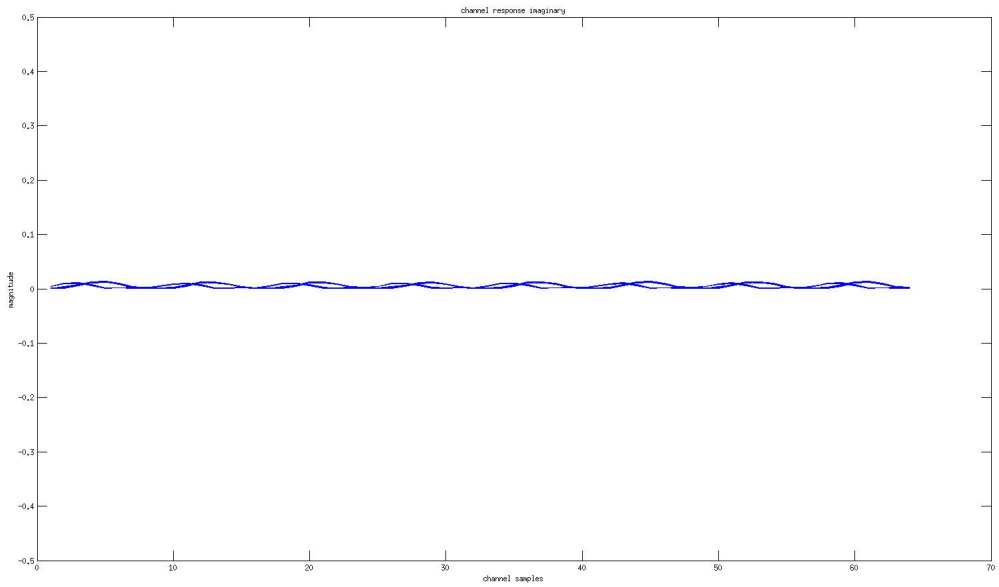


Figure 6.5: Channel coefficients - Imaginary Part

The received signal is shown in figure 6.7. The blue and green components refer to the real and imaginary parts received at the receiver. We have purposefully sent zeros after each frame in order to visualize the frame boundary.

The received spectrum is shown in figure 6.8.

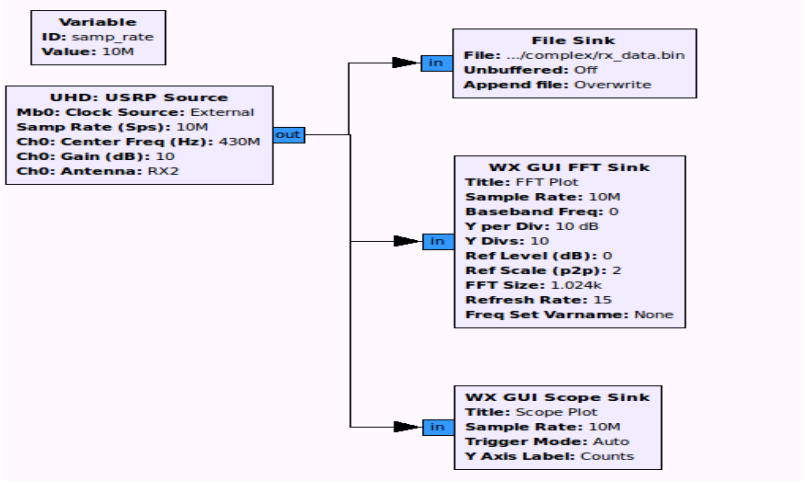


Figure 6.6: GRC Block Diagram of Receiver

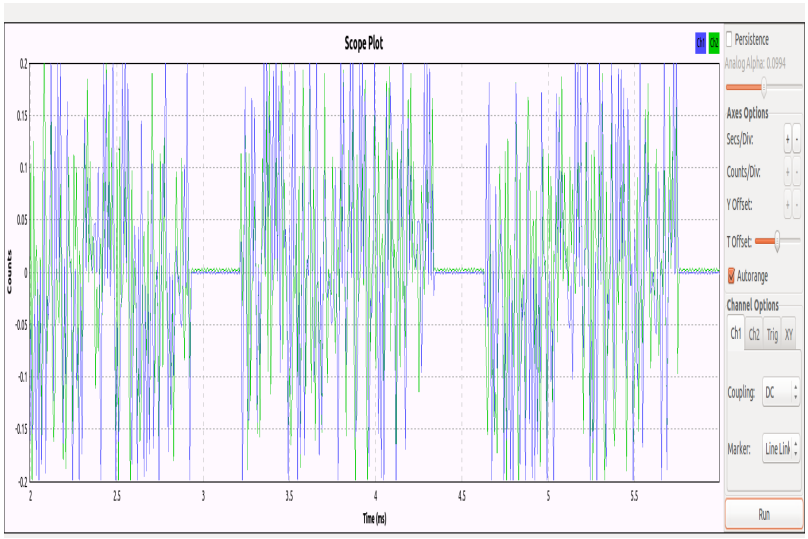


Figure 6.7: Received Signal

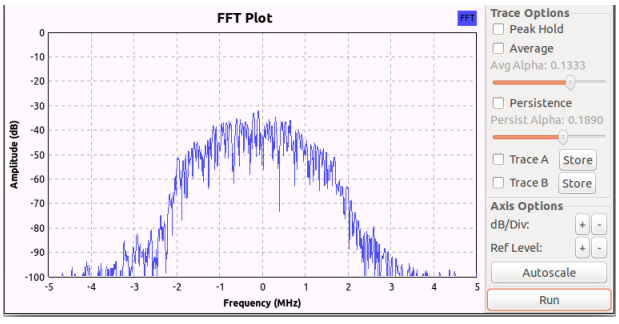


Figure 6.8: Received Spectrum

6.4 USRP Experimental Set up in ESB 218

The experiment was carried out with GNU Radio and real-time transmission with USRPs using wired and wireless mediums. The following are the pictures taken during the experiment figures 6.9,6.10.

Figure 6.9: Wired transmission and reception_1

Figure 6.10: Wireless transmission and reception_2

Chapter 7

Analysis of Collected Data

This chapter contain data collected in the machines lab at different times in the machines lab and analysis of these data.

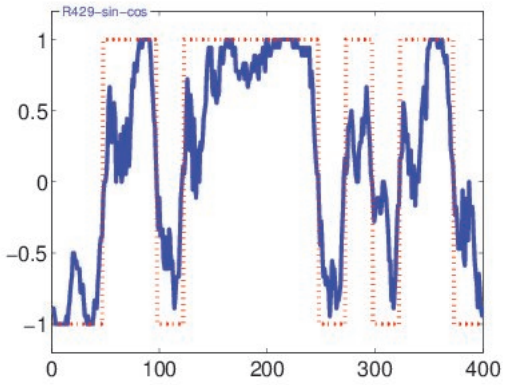


Figure 7.1: Right antenna frequancy 429MHz with -sin and -cos components

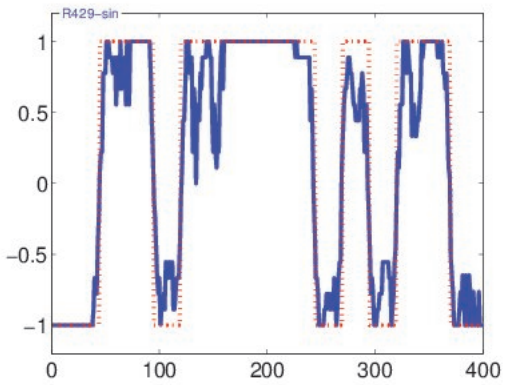


Figure 7.2: Right antenna frequancy 429MHz with -sin component

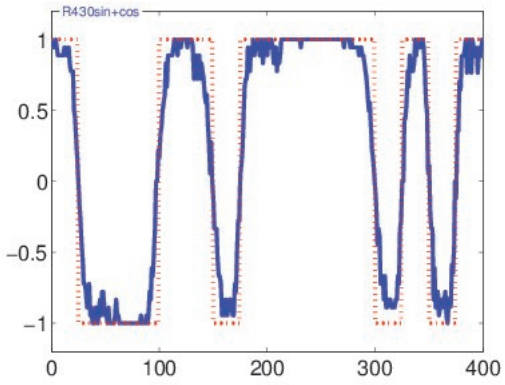


Figure 7.3: Right antenna frequancy 430MHz with sin and cos components

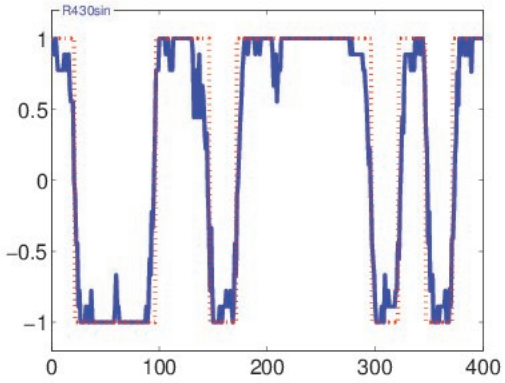


Figure 7.4: Right antenna frequancy 430MHz with sin component

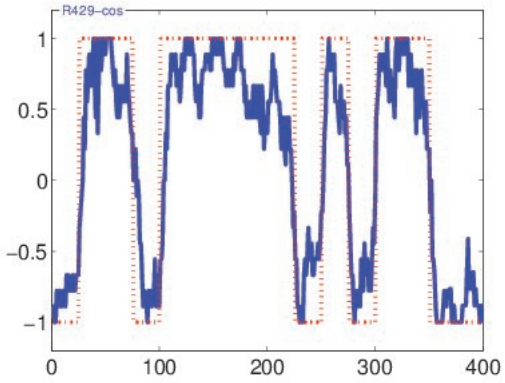


Figure 7.5: Right antenna frequancy 429MHz with -cos component

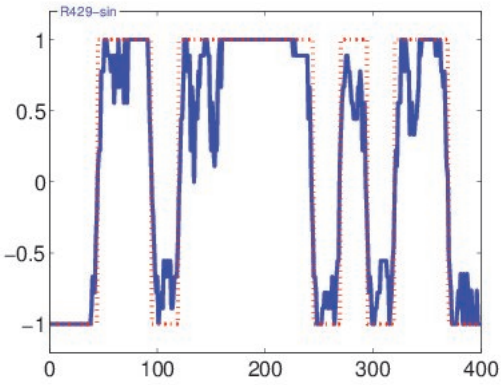


Figure 7.6: Right antenna frequancy 429MHz with -sin component

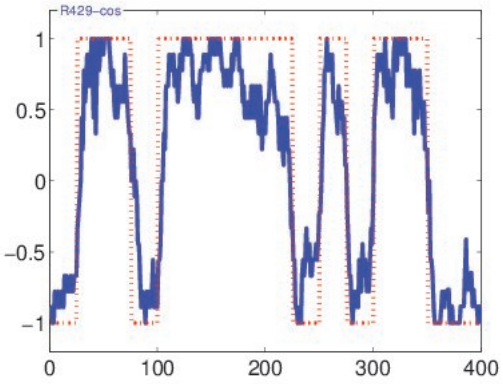


Figure 7.7: Right antenna frequancy 429MHz with -cos component

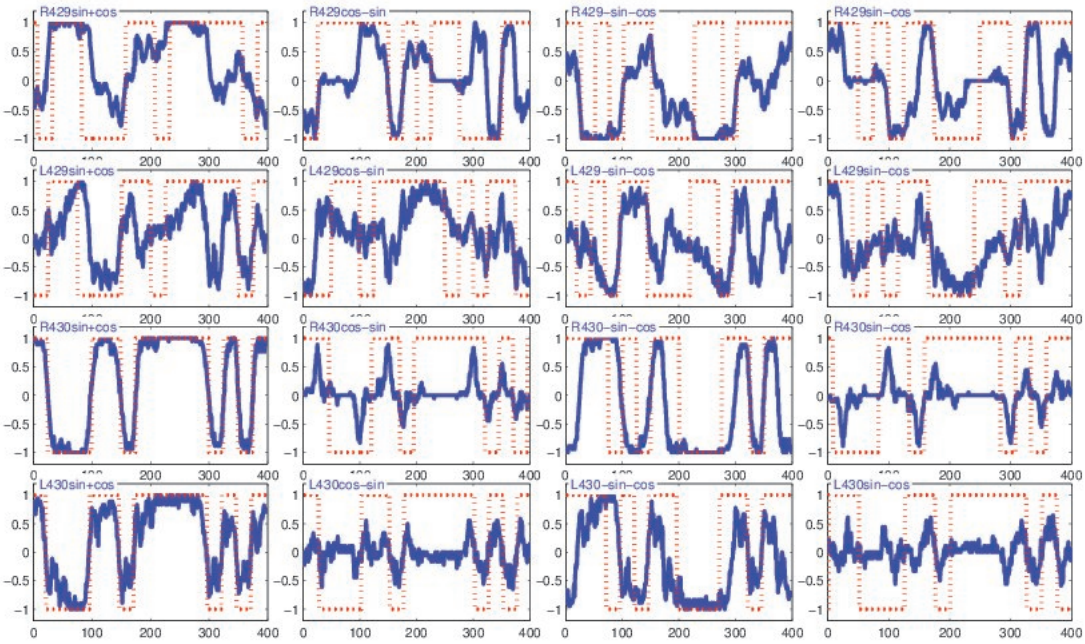


Figure 7.8: Right and Left antennas frequancy 429MHz,430MHz with sin,-sin ,cos,-cos components

Chapter 8

Future Work

In this chapter, we outline the research possibilities and the directions in which this work can be extended.

In this work we have considered flat fading model. This can be some times frequency selective fading too. In this scenario the receiver algorithm has to use channel equalization methods to deal with multi tap channel.

In this work we did not use bit interleaving technique for achieving time diversity.This bit interleaving can be utilized to improve BER performance. Since it is bit interleaving method, Forward Error Control codes are to be used.

Comparison of polarization diversity technique with other diversity techniques such as Maximal Ratio Combining(MRC) can be done to check its effectiveness over these in slow fading environments.

Appendices

APPENDIX A

Matlab Code for Transmitter,Channel,Receiver

This code contain Transmitter, Channel,Receiver parts. Function have called from this code they are also give below this code.At last there is one more function Read Complex Binary which is used to read .bin files in matlab. These .bin files are came from the C-code during data collection at receiver.

```
Clc;
clear all;
close all;

% number of frames : nfr

nfr=5;

% length of data : datalength

datalength=40;

% 40 bit input sequance genetarion

for i=1:nfr

x(i,:)=randi(2,1,datalength)*2-3;

end


Eb=energy_per_bit(x);

for p=1:1

tframe=transmiter_21_nov_2014(x,nfr);

% transmitter part completed

%  channel

tframe1=adding_channel_21_nov_2014(tframe);

%  RECEIVER PART

snr=7:30;

% Caicauting noise variances

sigma=noise_variance_21_nov_2014(snr,Eb);

% Ber calculations

ber1(p,:)=receiver_21_nov_2014(tframe1,sigma,nfr,datalength,x);
```

```
end

for p=1:length(snr)

   ber(p)=(sum(ber1(:,p))/(40*100));
end


figure

semilogy(snr,ber);

xlabel(' Eb/N0 ');
ylabel(' BER ');
title('BER curve for 1 frame transmission along with Channel varing every 64 fads on
21_nov_2014');

% receiver part is completed
```

```
function Eb=energy_per_bit(x)


[a b]=size(x);


% sum of squares of each rows energy

for i=1:a
   sum1(i)=0;
   for j=1:b

      sum1(i)=sum1(i)+(x(i,j)*x(i,j));

   end
end

% average energy per bit  Eb

for i=1:a

   sum2(i)=sum1(i)/b;
end


Eb=sum(sum2(:))/a;

end
```

```
function tframe=transmiter_21_nov_2014(x,nfr)

pre_psc1=2*[ 1   -1   -1   -1   -1   -1   -1   -1   -1   -1   1   -1   -1   1   -1   -1   1   -1
-1   1   1   -1   1   -1   -1   1   1   -1   1   -1   1   1   1   1   1   -1   -1   1   1   -1
-1   -1   1   1   1   1   1   -1   -1   1   -1   -1   -1   1   1   1   -1   1   1   1   1   1
1   -1   -1   -1   -1   1   1   1   -1   -1   -1   -1   -1   -1   -1   1   1   1   1   1   1   1
1   1   1   -1   -1   -1   1   1   1   -1   -1   -1   1   -1   -1   1   1   1   -1   1   1   -1
-1   1   -1   1   -1   1   1   1   -1   1   1   1   1   -1   1   -1   1   -1   -1   -1   1   1
1   1   -1   1   -1   -1   1   -1   1   -1   1   -1   -1   -1   -1   -1   1   -1   1   1   1   1
1   1   1   1   -1   1   -1   1   -1   1   -1   1   -1   1   1   1   1   -1   1   -1   -1   -1
-1   1   1   1   -1   1   -1   -1   1   -1   -1   -1   1   1   -1   -1   1   -1   1   1   -1   1
-1   1   1   -1   -1   1   1   1   1   -1   1   -1   1   1   -1   -1   -1   1   1   -1   -1   1
1   1   1   1   1   -1   -1   1   -1   1   -1   1   -1   1   -1   -1   1   1   -1   -1   1   1
-1   -1   1   -1   1   -1   -1   1   1   1   1   1   -1   1   -1   -1   1   1];

 pre_psc=pre_psc1';

%chip sequance gentration

code=[-1 -1 1 1 1 -1 1 -1;
    -1 1 -1 -1 1 1 1 -1;
    1 1 -1 1 -1 -1 1 -1;
    1 1 1 -1 1 -1 -1 -1];

 chipseq=[];

 for i=1:4

    chipseq=[chipseq,code(i,:)];

 end


 chipseq=[chipseq chipseq];

% multiplication of frame with chip sequance

% data+chipsequance : dc

dc=zeros(4,(length(chipseq)*length(x(1,:))));


for j=1:nfr

for i=1:length(x(j,:))


 dc(j,(((64*(i-1))+1):(64*i)))=x(j,i)*chipseq;

end

end


% preamble+data+chipsequance : pdc
```

```
pdc=[];
for i=1:nfr
   pdc=[pdc pre_psc];
   pdc=[pdc dc(i,:)];
end
pre_psc=(pre_psc./2);

% up sampling  preamble+data+chipsequance+upsampling : pdcu
pdcu=zeros(1,(4*length(pdc(:))));
 k=1;
 for i=1:4:length(pdcu(:))
   pdcu(i)=pdc(k);
   k=k+1;
end
over_sampling_factor = 4;
rrc=convolution_1(pdcu);
% multipling with exponent signal transmited frame : tframe
tframe=rrc.*exp(1j*2*pi*50000*(1:1:length(rrc))/(length(chipseq)*over_sampling_factor));

end
```

```
function tframe1=adding_channel_21_nov_2014(tframe)
tframe1=[];
%Generate a length 64 channel
h1=randn(1,256);
%Generate a phase offset
theta=rand(1,1);
%Multiply phase offset with the channel
h=h1(1:256).*exp(1i*2*pi*theta);
%Multiply each set of 256 samples with the 256 channel samples
for i=1:256:length(frame)
   if((length(frame)-i)<256)
      x=length(frame)-i+1;
   else
      x=length(frame(i:i+255));
   end
   y=frame(1,i:(i+x-1)).*h(1:x);
  tframe1=[tframe1 y];
end
end
```

```
function sig=noise_variance_21_nov_2014(snr,Eb)
 for i=1:length(snr)
    p=(snr(i)/10);
    p1=(10)^(p);
    p2=(Eb/p1);
    n(i)=sqrt(p2);
 end
 sig=n(:);
end
```

```
function ber=receiver_21_nov_2014(tframe1,sigma,nfr,datalength,x)

 pre_psc1=2*[    1   -1   -1   -1   -1   -1   -1   -1   -1   -1    1   -1   -1    1   -1   -1    1   -1
-1    1    1   -1    1   -1   -1    1    1   -1    1   -1    1    1    1    1    1   -1   -1    1    1   -1
-1   -1    1    1    1    1    1   -1   -1    1   -1   -1   -1    1    1    1   -1    1    1    1    1    1
1   -1   -1   -1   -1    1    1    1   -1   -1   -1   -1   -1   -1   -1   -1    1    1    1    1    1    1    1
1    1    1   -1   -1   -1    1    1    1   -1   -1   -1    1   -1   -1    1    1    1   -1    1    1   -1
-1    1   -1    1   -1    1    1    1   -1    1    1    1    1   -1    1   -1    1   -1   -1   -1    1    1
1    1   -1    1   -1   -1    1   -1    1   -1    1   -1   -1   -1   -1   -1    1   -1    1    1    1    1
1    1    1    1   -1    1   -1    1   -1    1   -1    1   -1    1    1    1    1   -1    1   -1   -1   -1
-1    1    1    1   -1    1   -1   -1    1   -1   -1   -1    1    1   -1   -1    1   -1    1    1   -1    1
-1    1    1   -1   -1    1    1    1    1   -1    1   -1    1    1   -1   -1   -1    1    1   -1   -1    1
1    1    1    1    1   -1   -1    1   -1    1   -1    1   -1    1   -1   -1    1    1   -1   -1    1    1
-1   -1    1   -1    1   -1   -1    1    1    1    1    1   -1    1   -1   -1    1    1];


pre1=[pre_psc1 1];

pre_int=interp1(1:(length(pre1)),pre1,1:0.25:(length(pre1)));

pre=pre_int(1:1024);

rx_n=(tx_frame)+(sigma*randn(1,length(tx_frame)));

rx_con=conv((rx_n),rrc);

for i=1:1:length(rx_con)-1024;

peak(i)=timing(rx_con(i:i+1023),pre);

end

figure

 plot(peak)
%
[q pos]=max(peak)

 k=1;

 for i=pos:4:length(rx_con)

    r2(k)=rx_con(i);

     k=k+1;

  end

     H1=(r2(1:length(pre_psc1))./pre_psc1);

    H=zeros(1,64);

    l=0;

     for p=1:64:length(H1)
```

```
H(1,:)=H(1,:)+H1((1+((p-1)/64)*64):(64+((p-1)/64)*64));
 l=l+1;
end
H=(H/l);
datalength=40;
r_h=r2((length(pre_psc1)+1):((datalength*length(chipseq))+length(pre_psc1)));
r=[];
for s=1:64:length(r_h)
r((1+((s-1)/64)*64):(64+((s-1)/64)*64))=(r_h((1+((s-1)/64)*64):(64+((s-1)/64)*64)).*conj(H));
end
t=1;
for u=1:(length(chipseq)):length(r(:))
excchipseq=sum((r(u:(u+(length(chipseq)-1))).*chipseq));
if(excchipseq>0)
rcap1(t)=1;
else
rcap1(t)=-1;
end
t=t+1;
end
r3(1,:)=rcap1(:);
error=decodfeb232015(r3,x_data)
```

```
function maximum=timing(r1,pre)

k=1;

for i=1:(length(r1)/4)

sum1(k)=((abs(r1(i))*abs(pre(i)))+(abs(r1(i+64))*abs(pre(i+64)))+
(abs(r1(i+128))*abs(pre(i+128)))+(abs(r1(i+192))*abs(pre(i+192))));

sum2(k)=(((sum1(k)*sum1(k)))/((sqrt((abs((pre(i))^2))+(abs((pre(i+64)^2)))+
(abs((pre(i+128))^2))+(abs((pre(i+192))^2))))*(sqrt(abs((r1(i)^2))+(abs(r1(i+64)^2))+
(abs(r1(i+128)^2))+(abs(r1(i+192)^2))))));

k=k+1;

end

sum3=sum(sum2(:));

maximum=sum3;

end
```

```
function error=decodfeb232015(xcap,x_data)

count=0;

 for i=1:length(xcap)

if(xcap(i)~=x_data(i))

count=count+1;

 end

 end

error=count;

end
```

```
function v = read_complex_binary (filename, count)
m = nargchk (1,2,nargin);
if (m)
 usage (m);
end
if (nargin < 2)
count = Inf;
end
f = fopen (filename, 'rb');
if (f < 0)
v = 0;
else
t = fread (f, [2, count], 'float');
fclose (f);
v = t(1,:) + t(2,:)*i;
[r, c] = size (v);
v = reshape (v, c, r);
end
```

APPENDIX B
C++ and C Code for Transmitter

```
#include <uhd/utils/thread_priority.hpp>

#include <uhd/utils/safe_main.hpp>

#include <uhd/utils/static.hpp>

#include <uhd/usrp/multi_usrp.hpp>

#include <uhd/exception.hpp>

#include <uhd/types/device_addr.hpp>

#include <boost/program_options.hpp>

#include <boost/math/special_functions/round.hpp>

#include <boost/foreach.hpp>

#include <boost/format.hpp>

#include <boost/thread.hpp>

#include <boost/progress.hpp>

#include <boost/lexical_cast.hpp>

#include <boost/algorithm/string.hpp>

#include <iostream>

#include <csignal>

#include "parameters.h"

#include "transmitter.cpp"

#define PI 3.14159265

namespace po = boost::program_options;

/***********************************************************************
* Signal handlers
**********************************************************************/

static bool stop_signal_called = false;
```

```
void sig_int_handler(int){stop_signal_called = true;}

/***********************************************************************
* Main function
**********************************************************************/

int UHD_SAFE_MAIN(int argc, char *argv[]){

uhd::set_thread_priority_safe();

//variables to be set by po

std::string args, ant, subdev, ref, otw, channel_list;

size_t spb;

double rate, freq, gain, bw;

float ampl;

double time_to_send=0.05;

double t_inc=0.006;

//setup the program options

po::options_description desc("Allowed options");

desc.add_options()

("help", "help message")

("args", po::value<std::string>(&args)->default_value(""), "192.168.10.2")

("spb", po::value<size_t>(&spb)->default_value(1000))

("rate", po::value<double>(&rate), "10e6")

("freq", po::value<double>(&freq), "430e6")

("ampl", po::value<float>(&ampl)->default_value(float(0.3)), "0.7")

("gain", po::value<double>(&gain), "20")

("ant", po::value<std::string>(&ant), "0")

("subdev", po::value<std::string>(&subdev), "200")

("bw", po::value<double>(&bw), "0")
```

```
("ref", po::value<std::string>(&ref)->default_value("external"), "external")
("otw", po::value<std::string>(&otw)->default_value("sc16"))
("channels", po::value<std::string>(&channel_list)->default_value("0,1"),  "0,1")
("int-n", "tune USRP with integer-N tuning");
po::variables_map vm;
po::store(po::parse_command_line(argc, argv, desc), vm);
po::notify(vm);
//print the help message
if (vm.count("help")){
std::cout << boost::format("UHD TX Waveforms %s") % desc << std::endl;
return ~0;
}
//create a multi_usrp with two boards in the configuration
uhd::device_addr_t dev_addr;
std::cout << std::endl;
dev_addr["addr0"] = "192.168.10.3";
dev_addr["addr1"] = "192.168.10.4";
std::cout << boost::format("Creating the usrp device with: %s...") % dev_addr["addr0"] <<
 std::endl;
//multi_usrp::sptr dev = multi_usrp::make(dev_addr);
uhd::usrp::multi_usrp::sptr usrp = uhd::usrp::multi_usrp::make(dev_addr);
//detect which channels to use
//std::cout<<"usrp->get_num_mboards() = "<<usrp->get_num_mboards()<<"\n";
std::vector<std::string> channel_strings;
std::vector<size_t> channel_nums;
```

```
boost::split(channel_strings, channel_list, boost::is_any_of("\"","));
std::cout << "channel_strings.size()= "<< channel_strings.size() <<"\n";
for(size_t ch = 0; ch < channel_strings.size(); ch++){
size_t chan = boost::lexical_cast<int>(channel_strings[ch]);
//std::cout << "chan= "<< chan <<"\n";
//std::cout << "usrp->get_tx_num_channels() = " << usrp->get_tx_num_channels() <<"\n";
if(chan >= usrp->get_tx_num_channels())
throw std::runtime_error("Invalid channel(s) specified.");
else
channel_nums.push_back(boost::lexical_cast<int>(channel_strings[ch]));
}
//Lock mboard clocks for TX0
usrp->set_clock_source(ref,0);
//always select the subdevice first, the channel mapping affects the other settings
if (vm.count("subdev")) usrp->set_tx_subdev_spec(subdev);
std::cout << boost::format("Using Device: %s") % usrp->get_pp_string() << std::endl;
//Lock mboard clocks for TX1
usrp->set_clock_source("mimo", 1);
usrp->set_time_source("mimo", 1);
//always select the subdevice first, the channel mapping affects the other settings
//if (vm.count("subdev")) usrp->set_tx_subdev_spec(subdev);
std::cout << boost::format("Using Device: %s") % usrp->get_pp_string() << std::endl;
//set the sample rate
if (not vm.count("rate")){
std::cerr << "Please specify the sample rate with --rate" << std::endl;
```

```
return ~0;

}

std::cout << boost::format("Setting TX Rate: %f Msps...") % (rate/1e6) << std::endl;

usrp->set_tx_rate(rate);

std::cout << boost::format("Actual TX Rate: %f Msps...") % (usrp->get_tx_rate()/1e6) << std::endl
 << std::endl;

//set the center frequency

if (not vm.count("freq")){

std::cerr << "Please specify the center frequency with --freq" << std::endl;

return ~0;

}

////SET FREQUENCY FOR MASTER TRANSMITTER

std::cout << boost::format("Setting TX0 Freq: %f MHz...") % (freq/1e6) << std::endl;

uhd::tune_request_t tune_request(freq);

//if(vm.count("int-n")) tune_request.args = uhd::device_addr_t("mode_n=integer");

usrp->set_tx_freq(tune_request, 0);

std::cout << boost::format("Actual TX0 Freq: %f MHz...") % (usrp->get_tx_freq(0)/1e6) <<
 std::endl << std::endl;

////SET FREQUENCY FOR SLAVE TRANSMIITER ..  ...../////////////

std::cout << boost::format("Setting TX1 Freq: %f MHz...") % (429.6875e6/1e6) << std::endl;

uhd::tune_request_t tune_request1(430e6);

//std::cout << boost::format("Setting TX1 Freq: %f MHz...") % (430e6/1e6) << std::endl;

//if(vm.count("int-n")) tune_request.args = uhd::device_addr_t("mode_n=integer");

usrp->set_tx_freq(tune_request1, 1);

std::cout << boost::format("Actual TX1 Freq: %f MHz...") % (usrp->get_tx_freq(1)/1e6) <<
std::endl << std::endl;
```

```
for(size_t ch = 0; ch < channel_nums.size(); ch++) {

//set the rf gain

if (vm.count("gain")){

std::cout << boost::format("Setting TX Gain: %f dB...") % gain << std::endl;

usrp->set_tx_gain(gain, channel_nums[ch]);

std::cout << boost::format("Actual TX Gain: %f dB...") % usrp->get_tx_gain(channel_nums[ch])
 << std::endl << std::endl;

}

//set the IF filter bandwidth

if (vm.count("bw")){

std::cout << boost::format("Setting TX Bandwidth: %f MHz...") % bw << std::endl;

usrp->set_tx_bandwidth(bw, channel_nums[ch]);

std::cout << boost::format("Actual TX Bandwidth: %f MHz...") % usrp-
>get_tx_bandwidth(channel_nums[ch]) << std::endl << std::endl;

}

//set the antenna

if (vm.count("ant")) usrp->set_tx_antenna(ant, channel_nums[ch]);

}

boost::this_thread::sleep(boost::posix_time::seconds(1)); //allow for some setup time

size_t index = 0;

//create a transmit streamer

//linearly map channels (index0 = channel0, index1 = channel1, ...)

uhd::stream_args_t stream_args("fc32", otw);

stream_args.channels = channel_nums;

uhd::tx_streamer::sptr tx_stream = usrp->get_tx_stream(stream_args);

//allocate a buffer which we re-use for each channel
```

```
if (spb == 0) spb = tx_stream->get_max_num_samps()*10;

std::vector<std::complex<float> > buff(spb);

std::vector<std::complex<float> *> buffs(channel_nums.size(), &buff.front());

uhd::sensor_value_t ref_locked = usrp->get_mboard_sensor("ref_locked",0);

std::cout << boost::format("Checking TX: %s ...") % ref_locked.to_pp_string() << std::endl;

UHD_ASSERT_THROW(ref_locked.to_bool());

uhd::sensor_value_t mimo_locked = usrp->get_mboard_sensor("mimo_locked",1);

std::cout << boost::format("Checking TX: %s ...") % mimo_locked.to_pp_string() << std::endl;

UHD_ASSERT_THROW(mimo_locked.to_bool());

float *data;

data=transmitter(rate);

uhd::time_spec_t t_start = usrp->get_time_now();

//setup the metadata flags

uhd::tx_metadata_t md;

md.start_of_burst = true;

md.end_of_burst = true;//metadata has the info to detect bursts as in start and end flags
 also delays before sending

md.has_time_spec = true;

md.time_spec = t_start + uhd::time_spec_t((int)time_to_send,(time_to_send-(int)time_to_send));

//md.time_spec = uhd::time_spec_t(0.1);

//std::cout << boost::format("Setting device timestamp to 0.01...") << std::endl;

//usrp->set_time_now(uhd::time_spec_t(0.007));

std::signal(SIGINT, &sig_int_handler);

////////////data to be transmitted/////////////

long int t1=databpf+padded+paddedend;
```

```
t1=t1*64*osf;

int pulse_size=symd*osf+1;

t1=(t1+pulse_size-1)*nfrm*2;

long int k=0;

std::cout << "Press Ctrl + C to stop streaming..." << std::endl;

std::cout<<"Buffer size is :"<<buff.size()<<std::endl;

//send data until the signal handler gets called

//boost::progress_timer timer;

while( not stop_signal_called)
{
        //fill the buffer with the waveform
        for (int n = 0; n < buff.size(); n++)
        {
                buff[n] =  std::complex<float>(*(data+k),*(data+k+1));
                k=k+2;
                if(k==t1)
                k=0;
        }

tx_stream->send(buffs, buff.size(), md);

//std::cout<<buff[0]<<std::endl;

md.start_of_burst = true;
md.end_of_burst = false;                // metadata has the info to detect bursts as in start and end flags
also delays before sending
md.has_time_spec = false;
md.time_spec = t_start + uhd::time_spec_t((int)time_to_send,(time_to_send-(int)time_to_send));
time_to_send+=t_inc;
}
std::cout<<buff.size()<<std::endl;
//std::cout<<buff[0]<<std::endl;
//send a mini EOB packet
md.end_of_burst = true;
tx_stream->send("", 0, md);
//finished
std::cout << std::endl << "Done!" << std::endl << std::endl;
return EXIT_SUCCESS;

}
```

parameters.h

```
int nfrm=1;
int databpf=20;
int padded=1; // no of bits padded at the beginning=4
int paddedend=6;
int osf=4;
int symd=8;
float roff=0.5;
float f1=156.25e3;
```

C-Code for Transmitter

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include"GetInformationBits.c"
#include"GetTheChipSeq.c"
#include"GetTheFrames.c"
#include"MultiplyFramesWithChipseq.c"
#include"Getupsampfrm.c"
#include"GetSquareRootRaisedCosinePulse.c"
#include"GetConvolvedOutput.c"
#include"MultiplyConvolvedFramesBySineAndCos.c"
#include"GetPreambleSeq64.c"
#include"MultiplyConvolvedFramesBySineAndCos1.c"
#include"MultiplyConvolvedFramesByCos.c"
#include"MultiplyConvolvedFramesBySine.c"

float* transmitter(double fs)
{

        int infbits,bpfrm,i,j;
        double num1,num2,x;
        int code[4][8]={
                                {-1,-1,1,1,1,-1,1,-1},
                                {-1,1,-1,-1,1,1,1,-1},
                                {1,1,-1,1,-1,-1,1,-1},
                                {1,1,1,-1,1,-1,-1,-1}
                        };
        int psize,flag;
        long int m,n;
        float gap,t;
        int *bits,*chipseq,*temp,*preamble;
        int *frame[nfrm];
        int *mulfrm[nfrm];
        int *upsampfrm[nfrm];
        float *srrcpulse,*temporary_sin,*temporary_cos;
        float *srrcpre[nfrm];
        float upsamp;
        float *frameh[nfrm],*framev[nfrm],*data;
        float tnsampfrm;
        infbits=nfrm*databpf;
        bits=GetInformationBits(infbits);

        chipseq=GetTheChipSeq(&code[0][0]);
        preamble=GetPreambleSeq();
```

```
	for(i=0;i<nfrm;i++)
	{
		frame[i]=GetTheFrames(bits,databpf,padded,paddedend,i);
	}
	bpfrm=databpf+padded+paddedend;
	for(i=0;i<nfrm;i++)
	{
		mulfrm[i]=MultiplyFramesWithChipseq(bpfrm,frame[i],chipseq,preamble,padded);
	}
	for(i=0;i<nfrm;i++)
	{
		upsampfrm[i]=Getupsampfrm(mulfrm[i],osf,bpfrm);
	}
	srrcpulse=GetSquareRootRaisedCosinePulse(roff,osf,symd);
	//temporary=srrcpulse;
	psize=symd*osf+1;
	upsamp=bpfrm*64*osf;
	for(i=0;i<nfrm;i++)
	{
		srrcpre[i]=GetConvolvedOutput(srrcpulse,upsampfrm[i],upsamp,psize);
	}
	tnsampfrm=upsamp+psize-1;
	for(i=0;i<nfrm;i++)
	{
		flag=0;

	frameh[i]=MultiplyConvolvedFramesBySineAndCos1(srrcpre[i],flag,f1,fs,osf,tnsampfrm);
		flag=-1;

	framev[i]=MultiplyConvolvedFramesBySineAndCos1(srrcpre[i],flag,f1,fs,osf,tnsampfrm);
	}
	data=(float*)calloc(2*tnsampfrm*nfrm,sizeof(float));
	FILE * pFile;
	pFile = fopen ("data_tx.txt","w");
	for(i=0;i<nfrm;i++)
	{
		temporary_sin=frameh[i];
		temporary_cos=framev[i];
		for(j=0;j<tnsampfrm;j++)
		{
			m=(i*2*tnsampfrm)+(2*j);
			*(data+m)=*(temporary_sin+j);
			*(data+m+1)=*(temporary_cos+j);
			fprintf (pFile,"%f\t%f\n",*(data+m),*(data+m+1));
		}
	}

	 return data;
}
```

```
int *GetInformationBits(int infbits)
{
        int k,i;
        int *bit=(int*)calloc(infbits,sizeof(int));
        if(bit==NULL)
        {
                printf("\nWe are out of memory.....");
                exit(1);
        }

        FILE *rp = fopen("data.txt", "r"); // read only

       if (rp == NULL)
       {
          printf("Error! Could not open file\n");
          exit(-1); // must include stdlib.h
       }

          for (i=0;i<infbits;i++)
          {
              k=fscanf(rp," %d",(bit+i));
          }

    fclose (rp);

    return bit;
}
```

```
int *GetTheChipSeq(int *code)
{
        int i;
        int *c,*temp;
        int *chipseq=(int*)calloc(64,sizeof(int));
        if(chipseq==NULL)
        {
                printf("\nWe are out of memory.....");
                exit(1);
        }
        temp=chipseq;
        c=code;
        for(i=0;i<32;i++)
        {
                *chipseq=*code;
                code++;
                chipseq++;
        }
        for(i=0;i<32;i++)
        {
                *chipseq=*c;
                c++;
                chipseq++;
        }
        return temp;
}
```

```
int *GetTheFrames(int *bits,int databpf,int padded,int paddedend,int frameno)
{
        int *frame,*temp;
        int bpfrm=databpf+padded+paddedend;
        int i,j;
        frame=(int*)calloc(bpfrm,sizeof(int));
        if(frame==NULL)
                {
                        printf("\nWe are out of memory.....");
                        exit(1);
                }
                temp=frame;

        for(j=0;j<bpfrm;j++)
        {
        if(j<padded)
        {
                *frame=1;
                frame++;
        }
        else if(j>=(bpfrm-paddedend))
        {
                *frame=0;
                frame++;
        }
        else
        {
                *frame=*(bits+(frameno*databpf+j-padded));
                frame++;
        }
        }
   return temp;
}
```

```
int *MultiplyFramesWithChipseq(int bpfrm,int *frame,int *chipseq,int *preamble,int padded)
{
	int j,k,*address,t,m;
	int *temp,*mulfrm=(int*)calloc(bpfrm*64,sizeof(int));
		if(mulfrm==NULL)
		{
			printf("\nWe are out of memory.....");
			exit(1);
		}
	address=chipseq;
	temp=mulfrm;
		for(j=0;j<bpfrm;j++)
		{
			chipseq=address;
			t=*frame;
			if(j<padded)
			{
				for(k=0;k<64;k++)
				{
					m=*(preamble+j*64+k);
					*mulfrm=t*m;
					mulfrm++;
				}
			}
			else
			{
				for(k=0;k<64;k++)
				{
					m=*chipseq;
					*mulfrm=t*m;
					mulfrm++;
					chipseq++;
				}
			}
			frame++;
		}
	return temp;
}
```

```
int *Getupsampfrm(int *mulfrm,int osf,int bpfrm)
{
	int i,j,k,t,m;
	int *temp,*upsampfrm=(int*)calloc(bpfrm*osf*64,sizeof(int));
	if(upsampfrm==NULL)
		{
			printf("\nWe are out of memory.....");
			exit(1);
		}
		temp=upsampfrm;
	m=(int)bpfrm*64;

	for(j=0;j<m;j++)
	{
		t=*mulfrm;
		*upsampfrm=t;
		upsampfrm++;
		mulfrm++;
		for(k=0;k<(osf-1);k++)
		{
			*upsampfrm=0;
			upsampfrm++;
		}
	}
	return temp;
}
```

```
#define pi 3.14159265
float *GetSquareRootRaisedCosinePulse(float roff,int osf,int symd)
{
	int i,psize;
	float a,t,gap,num1,num2;
	float *temp,*srrcpulse;
	a=roff;
	gap=1.0/osf;
	num1=1.0/(4.0*a);
	num2=-1.0/(4.0*a);
	psize=symd*osf+1;
	srrcpulse=(float*)calloc(psize,sizeof(float));
	if(srrcpulse==NULL)
		{
			printf("\nWe are out of memory.....");
			exit(1);
		}
		temp=srrcpulse;
	t=(float)-symd/2;
	for(i=0;i<psize;i++)
	{
		if(t==0.0)
		{
			*srrcpulse=(1.0-a)+4.0*a/pi;
		}
		else if(t==num1 || t==num2)
		{
			*srrcpulse=a/(float)sqrt(2)*((1.0+2.0/pi)*(float)sin(pi/(4.0*a))+(1.0-
2.0/pi)*(float)cos(pi/(4.0*a)));
		}
		else
		{
			*srrcpulse=((float)sin(pi*t*(1.0-a))+4.0*a*t*(float)cos(pi*t*(1.0+a)))/
(pi*t*(1.0-(4.0*a*t)*(4.0*a*t)));
		}

		*srrcpulse=(*srrcpulse)/osf;
		t=t+gap;
		srrcpulse++;
	}
	return temp;
}
```

```
float *GetConvolvedOutput(float *srrcpulse,int *fupsapm,float upsamp,int psize)
{
	int i,j,num1;
	float *temp,*srrcpre;
	float num2;
	srrcpre=(float*)calloc(upsamp+psize-1,sizeof(float));
	if(srrcpre==NULL)
		{
			printf("\nWe are out of memory.....");
			exit(1);
		}
	temp=srrcpre;
		for(i=0;i<upsamp;i++)
		{
			for(j=0;j<psize;j++)
			{
				num1=*(fupsapm+i);
				num2=*(srrcpulse+j);
				*(srrcpre+i+j)=*(srrcpre+i+j)+(float)(num1*num2);
			}
		}
	return temp;
}
```

```
float *MultiplyConvolvedFramesBySineAndCos(float *srrcpre,int flag,float f1,double fs,int
osf,float tnsampfrm)
{
	float t,gap,*temp,*mulfrm;
	int i,j;
	float num1,x,n;
	mulfrm=(float *)calloc(tnsampfrm,sizeof(float));
	if(mulfrm==NULL)
	{
		printf("\nWe are out of memory.....");
		exit(1);
	}
	t=1;
	gap=1;
	n=f1/fs;
	n=2*M_PI*n;
	for(i=0;i<tnsampfrm;i++)
	{
		x=t*n;
		x=x-(flag*M_PI/2);
		num1=sin(x);
		*(mulfrm+i)=(float)(*(srrcpre+i)*num1);
		t=t+gap;
		//if(t>99)
		//t=1;
	}
	return mulfrm;
}
```

```
int *GetPreambleSeq()
{
	int i;
	int preamble[64]={1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,1,-1,-1,1,-1,-1,1,1,-1,1,-1,-1,1,1,-1,1,-
1,1,1,1,1,1,-1,-1,1,1,-1,-1,-1,1,1,1,1,1,1,-1,-1,1,-1,-1,-1,1,1,1,-1,1,1,1,1,1,1,-1};


	int *preamble1=(int*)calloc(64,sizeof(int));
	if(preamble1==NULL)
	{
		printf("\nWe are out of memory.....");
		exit(1);
	}
	for(i=0;i<64;i++)
	{
		preamble[i]=preamble[i];
		*(preamble1+i)=preamble[i];
	}

	return preamble1;
}
```

```
float *MultiplyConvolvedFramesBySineAndCos1(float *srrcpre,int flag,float f1,double fs,int
osf,float tnsampfrm)
{
	float t,gap,*temp,*mulfrm,n;
	int i,j;
	float num1,x;
	mulfrm=(float *)calloc(tnsampfrm,sizeof(float));
	if(mulfrm==NULL)
	{
		printf("\nWe are out of memory.....");
		exit(1);
	}
	t=1;
	gap=1;
	n=f1/fs;
	for(i=0;i<tnsampfrm;i++)
	{
		x=2*3.1415*t*n;
		x=x-(flag*M_PI/2);
		num1=cosf(x);
		*(mulfrm+i)=*(srrcpre+i)*num1;
		t=t+gap;
	}

	return mulfrm;
}
```

```
float *MultiplyConvolvedFramesByCos(float *srrcpre,int flag,float f1,double fs,int osf,float
tnsampfrm)
{
	float t,gap,*temp,*mulfrm,n;
	int i,j;
	float num1,x;
	mulfrm=(float *)calloc(tnsampfrm,sizeof(float));
	if(mulfrm==NULL)
	{
		printf("\nWe are out of memory.....");
		exit(1);
	}
	t=1;
	gap=1;
	n=f1/fs;
	for(i=0;i<tnsampfrm;i++)
	{
		x=2*3.1415*t*n;
		//x=x-(flag*M_PI/2);
		num1=cosf(x);
		*(mulfrm+i)=*(srrcpre+i)*num1;
		t=t+gap;
	}
	printf("\n fs is %lf \n",fs);
	//n=(f1/fs);
	printf("\n n is %f \n",n);
	x=2*3.1415*50*n;
	printf("\n\n 50 th cos element is  %f \n\n",*(mulfrm+49));
	x=2*3.1415*50*n;
	printf("\n x is %f \n",x);
	printf("\n\n cos(x) is  %f \n\n",cosf(x));
	return mulfrm;
}
```

```
float *MultiplyConvolvedFramesBySine(float *srrcpre,int flag,float f1,double fs,int osf,float
tnsampfrm)
{
	float t,gap,*temp,*mulfrm;
	int i,j;
	double num1,x;
	mulfrm=(float *)calloc(tnsampfrm,sizeof(float));
	if(mulfrm==NULL)
	{
		printf("\nWe are out of memory.....");
		exit(1);
	}
	t=1;
	gap=1;
	for(i=0;i<tnsampfrm;i++)
	{
		x=2*M_PI*t*(f1/fs);
		//x=x-(flag*M_PI/2);
		num1=sin(x);
		*(mulfrm+i)=*(srrcpre+i)*(float)num1;
		t=t+gap;
	}
	return mulfrm;
}
```

APPENDIX C
C++ and C code for Receiver

```
#include <uhd/utils/thread_priority.hpp>

#include <uhd/utils/safe_main.hpp>

#include <uhd/usrp/multi_usrp.hpp>

#include <boost/program_options.hpp>

#include <boost/format.hpp>

#include <boost/thread.hpp>

#include <iostream>

#include <complex>

#include <fstream>

#include<stdio.h>

namespace po = boost::program_options;

/***********************************************************************
* Signal handlers
**********************************************************************/

static bool stop_signal_called = false;

void sig_int_handler(int){stop_signal_called = true;}

/***********************************************************************
* Main function
**********************************************************************/

int UHD_SAFE_MAIN(int argc, char *argv[]){

 uhd::set_thread_priority_safe();
```

```
std::string args, file, ant, subdev, ref;
size_t total_num_samps;
double rate, freq, gain, bw;
std::string addr, port;
//setup the program options
po::options_description desc("Allowed options");
desc.add_options()
("help", "help message")
("args", po::value<std::string>(&args)->default_value(""), "192.168.10.2")
("nsamps", po::value<size_t>(&total_num_samps)->default_value(1000), "500")
("rate", po::value<double>(&rate)->default_value(10e6/16), "10e6")
("freq", po::value<double>(&freq)->default_value(0), "480e6")
("gain", po::value<double>(&gain)->default_value(10), "10")
("ant", po::value<std::string>(&ant), "RX2")
("subdev", po::value<std::string>(&subdev), "200")
("bw", po::value<double>(&bw), "0")
("ref", po::value<std::string>(&ref)->default_value("external"), "external")
po::variables_map vm;
po::store(po::parse_command_line(argc, argv, desc), vm);
po::notify(vm);
//print the help message
if (vm.count("help")){
std::cout << boost::format("UHD RX to UDP %s") % desc << std::endl;
return ~0;
```

```
}

//create a usrp device

std::cout << std::endl;

std::cout << boost::format("Creating the usrp device with: %s...") % args << std::endl;

uhd::usrp::multi_usrp::sptr usrp = uhd::usrp::multi_usrp::make(args);

if (vm.count("subdev")) usrp->set_rx_subdev_spec(subdev); //sets across all mboards

std::cout << boost::format("Using Device: %s") % usrp->get_pp_string() << std::endl;

//Lock mboard clocks

usrp->set_clock_source(ref);

//set the rx sample rate

std::cout << boost::format("Setting RX Rate: %f Msps...") % (rate/1e6) << std::endl;

usrp->set_rx_rate(rate);

std::cout << boost::format("Actual RX Rate: %f Msps...") % (usrp->get_rx_rate()/1e6) << std::endl
<< std::endl;

//set the rx center frequency

std::cout << boost::format("Setting RX Freq: %f Mhz...") % (freq/1e6) << std::endl;

usrp->set_rx_freq(freq);

std::cout << boost::format("Actual RX Freq: %f Mhz...") % (usrp->get_rx_freq()/1e6) << std::endl
<< std::endl;

//set the rx rf gain

std::cout << boost::format("Setting RX Gain: %f dB...") % gain << std::endl;

usrp->set_rx_gain(gain);

std::cout << boost::format("Actual RX Gain: %f dB...") % usrp->get_rx_gain() << std::endl <<

std::endl;

//set the IF filter bandwidth
```

```
if (vm.count("bw"))
{
std::cout << boost::format("Setting RX Bandwidth: %f MHz...") % bw << std::endl;
usrp->set_rx_bandwidth(bw);
std::cout << boost::format("Actual RX Bandwidth: %f MHz...") % usrp->get_rx_bandwidth() <<
std::endl << std::endl;
}
//set the antenna
if (vm.count("ant"))
{
usrp->set_rx_antenna(ant);
std::cout<<"antenna selection\n";
}
boost::this_thread::sleep(boost::posix_time::seconds(1)); //allow for some setup time
//Check Ref and LO Lock detect
std::vector<std::string> sensor_names;
sensor_names = usrp->get_rx_sensor_names(0);
if (std::find(sensor_names.begin(), sensor_names.end(), "lo_locked") != sensor_names.end()) {
uhd::sensor_value_t lo_locked = usrp->get_rx_sensor("lo_locked",0);
std::cout << boost::format("Checking RX: %s ...") % lo_locked.to_pp_string() << std::endl;
UHD_ASSERT_THROW(lo_locked.to_bool());
}
sensor_names = usrp->get_mboard_sensor_names(0);
if ((ref == "mimo") and (std::find(sensor_names.begin(), sensor_names.end(), "mimo_locked") !=
```

```
 sensor_names.end())) {
uhd::sensor_value_t mimo_locked = usrp->get_mboard_sensor("mimo_locked",0);
std::cout << boost::format("Checking RX: %s ...") % mimo_locked.to_pp_string() << std::endl;
UHD_ASSERT_THROW(mimo_locked.to_bool());
}
if ((ref == "external") and (std::find(sensor_names.begin(), sensor_names.end(), "ref_locked") !=
sensor_names.end())) {
uhd::sensor_value_t ref_locked = usrp->get_mboard_sensor("ref_locked",0);
std::cout << boost::format("Checking RX: %s ...") % ref_locked.to_pp_string() << std::endl;
UHD_ASSERT_THROW(ref_locked.to_bool());
}
int num=0;
//create a receive streamer, host type does not matter
uhd::stream_args_t stream_args("fc32","sc16");
uhd::rx_streamer::sptr rx_stream = usrp->get_rx_stream(stream_args);
//issue stream command
uhd::stream_cmd_t stream_cmd(uhd::stream_cmd_t::STREAM_MODE_START_CONTINUOUS);
stream_cmd.num_samps = total_num_samps;
stream_cmd.stream_now = true;
usrp->issue_stream_cmd(stream_cmd);
//std::cout<<"nm_tx_samples :"<<
//loop until total number of samples reached
size_t num_acc_samps = 0; //number of accumulated samples
uhd::rx_metadata_t md;
```

```
std::vector<std::complex<float> > buff(rx_stream->get_max_num_samps());

float real1,imag1;

FILE * pFile;

pFile = fopen ("data_rx.bin","wb");

fseek(pFile, 0, SEEK_SET);

while(num_acc_samps<3e7){

size_t num_rx_samps = rx_stream->recv(&buff.front(), buff.size(), md);

for(int i = 0; i<buff.size(); i++)

{
                        real1=real(buff[i]);

                        imag1=imag(buff[i]);

                        //fprintf (pFile,"%f\t%f\n",real1,imag1);

                        fwrite(&real1, sizeof(float), 1, pFile);

                        fwrite(&imag1, sizeof(float), 1, pFile);

}

switch(md.error_code){

case uhd::rx_metadata_t::ERROR_CODE_NONE:

break;

case uhd::rx_metadata_t::ERROR_CODE_TIMEOUT:

if (num_acc_samps == 0) continue;

std::cout << boost::format("Got timeout before all samples received, possible packet loss, exiting
loop...") << std::endl;

goto done_loop;

default:

std::cout << boost::format("Got error code 0x%x, exiting loop...") % md.error_code << std::endl;
```

```
goto done_loop;
}
num_acc_samps+=num_rx_samps;
}done_loop:
fclose(pFile);
std::cout << std::endl << "Done!" << std::endl << std::endl;
return EXIT_SUCCESS;
}
```

C- Code for Receiver

```
#include<stdio.h>
#include<stdlib.h>
#include<complex.h>
#include<math.h>
#include"GetPartialValues.c"
#include"GetTheChipSeq.c"
#include"GetSquareRootRaisedCosinePulse.c"
#include"GetConvolvedOutput.c"
#include"GetPreambleSeq64.c"
#include"GetInterPreambleSeq64.c"
#include"AllocateMemory.c"
#include"FrameSynchr64.c"
#include"ChannelEstimation64.c"
#include"ChipExt.c"
int main()
{
        /////////Parameters//////////////
        int nbits=20;
        int input[nbits],*in_data;
```

```
long n=4e7,i;                          //no of samples to be read
int framecheck=5000,num_frame=2000;//number of samples from received data to be read
float *rx_data,biterr=0;               //received data
float *real1,*imag1;                   //real and imaginary parts
float *srrc,roff=0.5;                  //SRRC pulse
int symd=8,osf=4,psize;
int code[4][8]={
                                       {-1,-1,1,1,1,-1,1,-1},
                                       {-1,1,-1,-1,1,1,1,-1},
                                       {1,1,-1,1,-1,-1,1,-1},
                                       {1,1,1,-1,1,-1,-1,-1}
                };
        int *chipseq,*pre1;
        float *pre;
        float *real_conv,*imag_conv;        //Convolved data
        float *channel_real,*channel_imag;  //Channel coefficients
        FILE *rp1,*rp,*chan_file;
        int pos;                            //start of the frame
        int bit[nbits];                     //Exctracted bits
        int error,k,erroredframe=0;
        int fram_err[num_frame];
/////////////// Input data ////////////////////
        rp1 = fopen("data.txt", "r");           // read only
    if (rp1 == NULL)
    {
       printf("Error! Could not open file\n");
       exit(-1);                            // must include stdlib.h
```

```
}

for (i=0;i<nbits;i++)

{

fscanf(rp1,"%d",&input[i]);

}

fclose (rp1);

in_data=input;

printf("Input data is \n");

for(i=0;i<nbits;i++)
        {
                printf("%d ",*(in_data+i));
        }
printf("\n");

        /////////////// Received data ////////////////

rx_data=(float *)calloc(n,sizeof(float));

if(rx_data==NULL)
{
                printf("\nWe are out of memory");
                exit(1);
}

rp = fopen("data_rx.bin", "rb");              // read only

if(rp==NULL)
      {
            printf("\nFile can not be opened\n");
            exit(-1);
      }

fseek(rp, 6e6, SEEK_SET);

for(i=0;i<n;i++)

      {
            fread((rx_data+i),sizeof(float),1,rp);
      }
```

```
 fclose(rp);
printf("\n");
//////////////////////Get real part of of data/////////
real1=GetPartialValues(rx_data,n,0);
//////////////Get imaginary part ////////////////////
imag1=GetPartialValues(rx_data,n,1);
//////////////////////////Gete the chip sequence ////////////////////////////
chipseq=GetTheChipSeq(&code[0][0]);
//////////////////////Square root raised cosine/////////////////////////////////
srrc=GetSquareRootRaisedCosinePulse(roff,osf,symd);
psize=symd*osf+1;                                   //length of srrc pulse
printf("\n SRRC pulse is\n");
for(i=0;i<33;i++)
{
        printf("%ld) %f__ ",i+1,*(srrc+i));
}
//////////////////////Preamble sequence /////////////////
pre1=GetPreambleSeq();
//////////////////////Interpolated preamble sequence//////////////////////////
pre=GetInterPreambleSeq(pre1,srrc);
/////////////////Convolution of real part/////////////////
real_conv=GetConvolvedOutput(srrc,real1,n/2,psize);
printf("\n");
```

```
//////////////////Convolution of Imaginary part//////////////////

imag_conv=GetConvolvedOutput(srrc,imag1,n/2,psize);

//////////////Memory allocation for channel//////

channel_real=AllocateMemory();

channel_imag=AllocateMemory();

chan_file = fopen ("channel_rx.bin","wb");

if(chan_file==NULL)
        {
                printf("\nWe are out of memory");
                exit(1);
        }

fseek(chan_file, 0, SEEK_SET);

for(k=0;k<num_frame;k++)
        {
              {
                  printf("\n framenumber : %d",k);
              }

////////////////////////Frame detection////////////////////////////////////

pos=FrameSynchr(real_conv,imag_conv,pre);

printf("Pos: %d", pos);

///////Channel Estimation ////////////////////////////////////

ChannelEstimation(real_conv,imag_conv,channel_real,channel_imag,pre1,pos);

                for(i=0;i<64;i++)
                {
                        fwrite(channel_real+i, sizeof(float), 1, chan_file);
                        fwrite(channel_imag+i, sizeof(float), 1, chan_file);
                }

//////////////////////Chip Exctraction ////////////////

real_conv=real_conv+pos+256;

imag_conv=imag_conv+pos+256;
```

```
float *GetPartialValues(float *rx_data,long n,int flag)
{
	int i;
	float *num_seq=(float *)calloc(n/2,sizeof(float));
	if(num_seq==NULL)
	{
		printf("\nwe are out of memory!!!!\n");
		exit(1);
	}
	for(i=0;i<n/2;i++)
	{
		*(num_seq+i)=*(rx_data+2*i+flag);
	}
	return num_seq;
}
```

```
int *GetTheChipSeq(int *code)
{
	int i;
	int *c,*temp;
	int *chipseq=(int*)calloc(64,sizeof(int));
	if(chipseq==NULL)
	{
		printf("\nWe are out of memory.....");
		exit(1);
	}
	temp=chipseq;
	c=code;
	for(i=0;i<32;i++)
	{
		*chipseq=*code;
		code++;
		chipseq++;
	}
	for(i=0;i<32;i++)
	{
		*chipseq=*c;
		c++;
		chipseq++;
	}
	return temp;
}
```

```
#define pi 3.14159265

float *GetSquareRootRaisedCosinePulse(float roff,int osf,int symd)

{
	int i,psize;

	float a,t,gap,num1,num2;

	float *temp,*srrcpulse,total_sum=0;

	a=roff;

	gap=1.0/osf;

	num1=1.0/(4.0*a);

	num2=-1.0/(4.0*a);

	psize=symd*osf+1;

	srrcpulse=(float*)calloc(psize,sizeof(float));

	if(srrcpulse==NULL)

		{
			printf("\nWe are out of memory.....");
			exit(1);
		}
	temp=srrcpulse;

	t=(float)-symd/2;

	for(i=0;i<psize;i++)

	{
		if(t==0.0)
		{
			*srrcpulse=(1.0-a)+4.0*a/pi;
		}
		else if(t==num1 || t==num2)
		{
*srrcpulse=a/(float)sqrt(2)*((1.0+2.0/pi)*(float)sin(pi/(4.0*a))+(1.0-2.0/pi)*(float)cos(pi/(4.0*a)));

		}
		else
		{
```

```
*srrcpulse=((float)sin(pi*t*(1.0-a))+4.0*a*t*(float)cos(pi*t*(1.0+a)))/(pi*t*(1.0-
(4.0*a*t)*(4.0*a*t)));

}
                total_sum=total_sum+(*srrcpulse);
                t=t+gap;
                srrcpulse++;
}
        total_sum=sqrtf(total_sum);

        for(i=0;i<psize;i++)
        {
              *(temp+i)=(*(temp+i))/total_sum;
        }
        return temp;
}
```

```
float *GetConvolvedOutput(float *srrcpulse,float *fupsapm,long n,int psize)
{
        int i,j;
        float *temp,*srrcpre;
        float num2,num1;
        srrcpre=(float*)calloc(n+psize-1,sizeof(float));
        if(srrcpre==NULL)
                {
                        printf("\nWe are out of memory.....");
                        exit(1);
                }
        temp=srrcpre;
                for(i=0;i<n;i++)
                {
                        for(j=0;j<psize;j++)
                        {
                                num1=*(fupsapm+i);
                                num2=*(srrcpulse+j);
                                *(srrcpre+i+j)=*(srrcpre+i+j)+(float)(num1*num2);
                        }
                }
        /*for(i=0;i<n+psize-1;i++)
        {
                printf("%f ",*(srrcpre+i));
        }*/

        return temp;
}
```

```
int *GetPreambleSeq()
{
        int i;
        int preamble[64]={1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,1,-1,-1,1,-1,-1,1,1,-1,1,-1,-1,1,1,-1,1,-
1,1,1,1,1,1,-1,-1,1,1,-1,-1,-1,1,1,1,1,1,1,-1,-1,1,-1,-1,-1,1,1,1,-1,1,1,1,1,1,1,-1};


        int *preamble1=(int*)calloc(64,sizeof(int));
        if(preamble1==NULL)
        {
                printf("\nWe are out of memory.....");
                exit(1);
        }
        for(i=0;i<64;i++)
        {
                preamble[i]=preamble[i];
                *(preamble1+i)=preamble[i];
        }

        return preamble1;
}


float *GetInterPreambleSeq(int *pre1,float *srrc)
{

float *pre=(float*)calloc(256,sizeof(float));

int k,i,count=0;

float *pre2;

        if(pre==NULL)
        {
                printf("\nWe are out of memory.....");
                exit(1);
        }
k=0;

for(i=0;i<64;i++)
{
 *(pre+k)=*(pre1+i);
 *(pre+k+1)=((0.75*(*(pre1+i)))+(0.25*(*(pre1+i+1))));
*(pre+k+2)=((0.5*(*(pre1+i)))+(0.5*(*(pre1+i+1))));
*(pre+k+3)=((0.25*(*(pre1+i)))+(0.75*(*(pre1+i+1))));

  k=k+4;
}

for(i=0;i<256;i++)
 {
  if(*(pre+i)==0)
  {
    *(pre+i)=0.01;
  }
 }

return pre;
 }
```

```
float *AllocateMemory()
{
	float *channel=(float *)calloc(64,sizeof(float));
	if(channel==NULL)
	{
		printf("\nWe are out of memory");
		exit(1);
	}
	return channel;
}
```