University of Southern Queensland

Faculty of Engineering & Surveying

Software Simulator Development for Orthogonal Frequency Division Multiplexing (OFDM) Modulation

A dissertation submitted by

Barry Dunbar

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Electrical and Electronic)

Submitted: November, 2006

Abstract

Orthogonal Frequency Division Multiplexing (OFDM) modulation using varied inputs, code rates, modulation schemes and sub-carrier sizes over different fading and noisy channels were developed and tested in this thesis. The different code rates were the $\frac{1}{2}$, $\frac{2}{3}$ and $\frac{3}{4}$ whilst the different modulation schemes were Binary Phase Shift Keying (BPSK), Quadrature Phase Shift Keying (QPSK), 16-Quadrature Amplitude Modulation (16-QAM) and 64-Quadrature Amplitude Modulation. The different size sub-carriers were 64, 256, 512, 1024, 2048, 4096 and 8192. The different channels were the Rayleigh fading and the Rician fading channels. These aspects of the OFDM modulation system were developed using Matlab 7.1 as code written program utilising the toolboxes available.

The simulator was setup to be able to mirror all current implementations of the technology. Through the use of the simulator, a varied amount of system configurations were tested. The system arrangement that was found to be the most resilient to noise was a system using the three-quarter code rate, 64-QAM modulation and the 64 sub-carrier size.

University of Southern Queensland

Faculty of Engineering and Surveying

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Prof R Smith

Dean

Faculty of Engineering and Surveying

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Barry Dunbar

0050022993

_____

Signature

_____

Date

Acknowledgments

I would like to thank the people who encouraged and helped me along not only in finishing this thesis but throughout the course as a whole.

To my supervisor, Dr. Wei Xiang for the guidance throughout the project and constant emails making sure I was keeping on track.

To my brother, John, who showed the way and guided me along in the technical and detail areas.

To my wife's extended family, The Mackeys, for the many and varied social evenings around the chimenea.

To my brother-in-law, Patrick, for knowing what needs to be known when it comes to processing pictures.

Finally, to my loving wife, Katherine, who has watched, supported and kept the home fires burning. Her stoicism amidst the technical babble were a welcome relief and allowed me to think outside the square.

Barry Dunbar

University of Southern Queensland
November 2006

Contents

List of Figures

List of Tables

Nomenclature

16-QAM	16 point Quadrature Amplitude Modulation
64-QAM	64 point Quadrature Amplitude Modulation
A/D	Analogue to Digital
ADSL	Asynchronous Digital Subscriber Line
AGC	Automatic Gain Control
AM	Amplitude Modulation
AMPS	Analogue Mobile Phone System
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
CCK	Complementary Code Keying
CDMA	Code Division Multiple Access
CIF	Common Interleaved Frame
CS-CCA	Carrier Sense - Clear Channel Assessment
D/A	Digital to Analogue
dB	Decibel
DAB	Digital Audio Broadcast
DAS	Distributed Antennae System
DE	Doppler Effect
D-BPSK	Differential Binary Phase Shift Keying
D-QPSK	Differential Quadrature Phase Shift Keying
DVB	Digital Video Broadcast
DVB-H	Digital Video Broadcast - Handheld
EDTV	Enhanced Definition Television

EMI	Electro-Magnetic Interference
ERP	Extended Rate Preamble
FDD	Frequency Division Multiplexing
FDMA	Frequency Division Multiple Access
FFT	Fast Fourier Transform
FIC	Fast Information Channel
FLASH	Fast low Latency Access and Seamless Handoff
FM	Frequency Modulation
FSK	Frequency Shift Keying
GI	Guard Interval
GSM	Global Switching Mobile
GUI	Graphical User Interface
GUIDE	Graphical User Interface Development Environment
HDSL	High Digital Subscriber Line
HDTV	High Definition Television
HIPERLAN	High Performance Radio Local Area Network
IBC	In Building Coverage
ICI	Inter-Channel Intermodulation
IF	Intermediate Frequency
IFFT	Inverse Fast Fourier Transform
I/Q	In phase and Quadrature phase
IMD	Inter-Modulation Distortion
ISDN	Integrated Services Digital Network
ISI	Inter-Symbol Interference
ISM	Industrial, Scientific and Medical
LDTV	Limited Definition Television
LOS	Line of Sight
MCM	Multi-Carrier Modulation
MPEG	Moving Picture Experts Group
MSC	Main Service Channel
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access

PBCC	Packet Binary Convolutional Coding
QPSK	Quadrature Phase Shift Keying
RF	Radio Frequency
RGB	Red, Green and Blue
SDTV	Standard Definition Television
SFD	Start Frame Delimiter
SNR	Signal to Noise Ratio
TDD	Time Division Multiplexing
TDMA	Time Division Multiple Access
TPS	Transmitter Parameter Signalling
UHF	Ultra High Frequency
VHF	Very High Frequency
VDSL	Very high bit Digital Subscriber Line
WLAN	Wireless Local Area Network
WMAN	Wireless Metropolitan Area Network
WCDMA	Wideband Code Division Multiple Access

Chapter 1

Introduction

1.1 Introduction

As technology advances, the future impact on our lives is never fully realised by those implementing the new technology. These impacts can change the way we work, rest and play. Since the advent of the Internet towards the closure of the last century, there has been a veritable explosion in the number of people who have taken up the opportunity to become connected to a vast number of fellow humans, some known, but mostly unknown, from around the globe, transcending nationalities, cultures and the tyranny of distance. This Internet technology has enabled direct access, freely unfettered to different people a wealth of information and to business entities that they previously would have been unlikely or unable to access.

Allied to the fast pace of our modernlifestyles and modern society, this Internet revolution has also facilitated an explosion in the creative use and adoption of telecommunication products to achieve and maintain a state of perpetual connection. Included with this take-up, the increase in the use of wireless devices, which have the potential and ability to keep us in even greater touch, designed to the relentless trend towards freedom of usage. In order to try and meet the demand for more sophisticated, secure and faster connections, many different types of connection models have been introduced. Examples of these include ISDN, ADSL, GSM, TDMA, CDMA, WLAN and

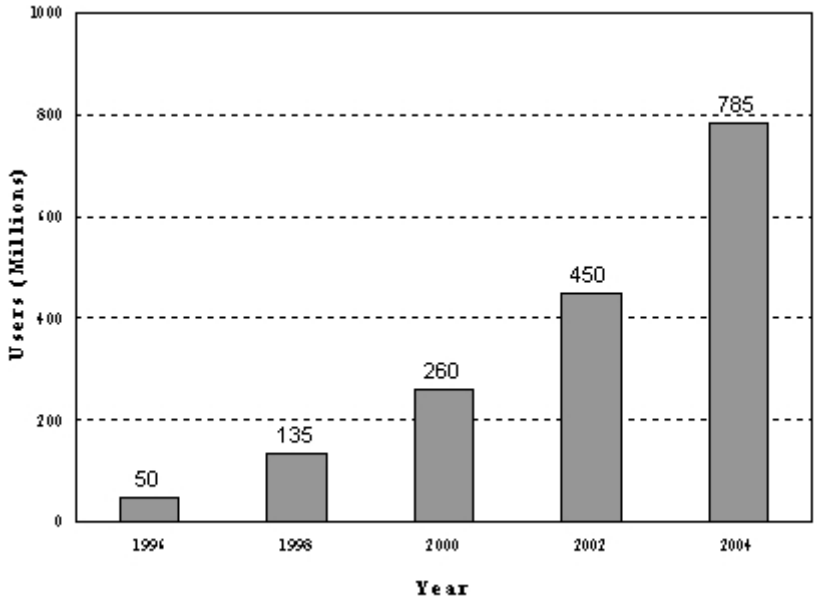

Figure 1.1: World-wide number of Internet users

Source: Engels, Figure 1.1, p1

HIPERLAN, just to name a few.

1.2 Project Aim

In recent years, the need for fast wireless communications has increased. This has led to a number of investigations as to how these services can best be provided. The aim of this project is to investigate the use of Orthogonal Frequency Division Multiplexing (OFDM) Modulation, as a vehicle for supplying these future needs, via the use of a simulation tool known as MATLAB

1.3 Project Objectives

The research efforts undertaken during this project are focused upon meeting various objectives. These project objectives are as follows:

- Research the methodology of Orthogonal Frequency Division Multiplexing (OFDM) Modulation.
- Examine a radio channel and discuss the impact on communication signals and systems.
- Design an Orthogonal Frequency Division Multiplexing (OFDM) Modulation simulator using MATLAB.
- Verify the simulation results matching the theoretical results of an Orthogonal Frequency Division Multiplexing (OFDM) transceiver system.
- Analyse the simulation results relative to different scenarios and rationalise these results with regard to network integrity.

Through research, design and analysis, simulation of the possibilities of wireless connectivity may demonstrate an ability in some instances to meet the needs of future users but an inability to do so in other instances.

1.4 Effects

All projects will have outcomes on various groups of society or surrounding environments to varying degrees. Some of these outcomes will be foreseen by the project engineers and therefore, will have mechanisms in place to minimise any damaging consequences. Other consequences may not be foreseen and so the engineers should endeavour to make the technical activity as safe as possible.

1.4.1 Sustainability

The choice of the materials used in the production of the transceiver need to be thoroughly investigated to keep the impact on future generations minimal. Some components used in the manufacture will be hazardous to the environment when the

transceiver's lifecycle has come to an end. The amount of original resources can be reduced with the use of recycling programs. Through recycling, some of these resources can be reused thus reducing the impact on a limited resource, Earth. The choice of components can decide the manufacturing process. If the components were safe to the environment, the manufacturing process would need to be investigated for its impact. This would include the degradation of the surrounding environment, such as contamination of waterways and nature parks, and the increase in ozone holes. Through the consideration of the impact that production, use and disposal of a communication transceiver has, in the planning stage, we guarantee the equitable growth and cost of the system across multiple generations [Johnston et al 1999 p476].

1.4.2 Safety

The implementation of any new engineering activity induces and introduces hazards and insecurities through the act of modernisation [Adams, 1998 p180]. The introduction of a new wireless transmission system creates an area where the Radio Frequency (RF) energy is higher than the standard allows. The transmitting antenna has to be situated so that special keys or tools are required to approach it. The antennae would be placed on a tall tower or over the edge of a building away from the general public. Maintenance staff would need access so they can ensure the most efficient use of the system. There is a trade-off between the general public and network operators. The transceiver design requires that when staff is handling it, they do not injure themselves. The weight and shape has to be designed so that the maintenance staff can carry the unit easily.

1.4.3 Risk Assessment

A Risk Assessment was conducted for a communications transceiver.

Table 1.1: Risk assessment

Hazard	People at Risk	Parts of the body	Risk Level	Categories	Short Term Controls	Long Term Controls
RF Exposure	Any person within the designated area where the RF level is greater than the standard allows. AS2772.	All	Moderate	Transceiver design, Location design	Measure reverse power, shutdown when above set limit.	Place transmitter aerial in an area where equipment is needed to access it e.g. tower
Transceiver too heavy	Installers and Maintenance staff	Back	Moderate	Transceiver design	Manual handling devices	Separate transceiver into less weighty components
Transceiver hot to touch	Installers and Maintenance staff	Hands	Low	Transceiver design	Cooling fans	Monitor temperature and shutdown when limit reached
Electric shock	Installers and Maintenance staff	All	Low	Transceiver design	Circuit breaker	Residual Current Device

1.5 Overview of the Dissertation

This dissertation is organized as follows:

Chapter 2 describes the basic problem of why this technology is needed and the design and application of the technology to overcome the difficulties of wireless transmission.

Chapter 3 discusses the radio environment and the impact it has on the different signals and systems associated with it.

Chapter 4 describes the current applications of OFDM technology and the different structures of each implementation.

Chapter 5 details the MATLAB model used to simulate OFDM transmission.

Chapter 6 reports the results of the simulator and relates these results to the impact they would have on network integrity.

Chapter 7 reports the conclusions of the dissertation. There is,also, a discussion on the suggestion for future work.

Chapter 2

OFDM Review

2.1 History

The concept of OFDM communications has evolved from the use of Frequency Division Multiplexing (FDM). Over a century ago, the use of telegraph employed more than one low rate signal being carried over a relatively wide bandwidth channel using separate carrier frequencies [Bahai et al, 2004 pp5-6]. These frequencies, or sub-carriers, were spaced far enough apart so they did not overlap. This allowed for filter design to be less specific and ensured that inter-channel interference was minimised. This system, compared to today's communication systems, was very inefficient. Through research during the 1950s and 1960s, a more efficient parallel data and FDM approach was developed.

With the new concept, the intention is for the sub-carriers to overlap each other. This could only be achieved if the sub-carriers each carry a signal rate, c, as well as be spaced a multiple of $\frac{1}{T}$ apart in frequency. This concept is called "orthogonal". Using this criterion, a more efficient use of the bandwidth could be employed, whilst still avoiding impulsive noise and the use of high-speed equalisation. Such efficiencies would save up to 50% of bandwidth [Prasad, 2004 p12]. In 1971, Weinstein and Ebert chose the use of the Discrete Fourier Transform (DFT) as part of the modulation / demodulation scheme. This allowed the implementation complexity of the equipment to be reduced

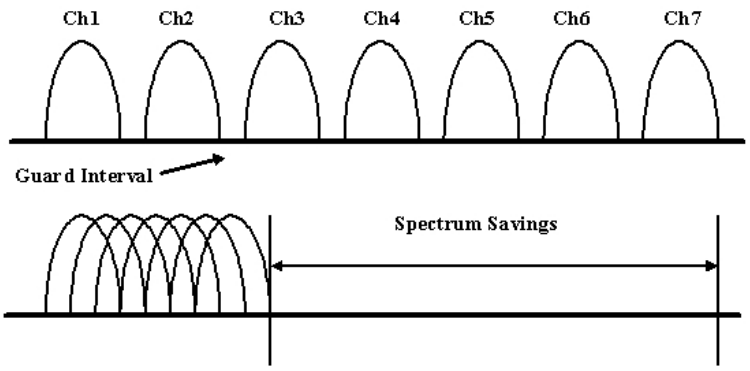

Figure 2.1: Concept of OFDM, (a) conventional frequency allocation and (b) OFDM frequency allocation

[Hanzo et al, 2002 p536], but due to manufacturing constraints, cost and size, usage of this idea would not occur. Then, during the 1980s and 1990s, the development of very large scale integrated chipsets that could implement the functions of OFDM has seen its use expanded considerably. OFDM is now being used or considered for use in:

- High speed modems
- Digital mobile communications
- High density recording
- Various Digital Subscriber Lines (HDSL, ADSL, VDSL, etc)
- Digital Audio Broadcasting (DAB) and
- High Definition Television(HDTV) Transmission

2.2 Design

The sub-carriers need to overlap themselves to ensure the maximum use of available spectrum. This type of hybrid communication came about from combining the two ideas of multi-carrier modulation (MCM) and Frequency Shift Keying (FSK) [Heiskala et al, 2002 p31]. However, using this type of communication creates problems at the

receiver end when the data needs to be processed from the received signal. One way of ensuring the relevant data can be extracted from the received signal is to ensure that all the sub-carriers are orthogonal. This is the basic principle for all Orthogonal Frequency Division Multiplexing (OFDM) Systems today. For maximum efficiency, a number of equal sized sub-carriers would divide the available spectrum. This would allow for parallel data transportation instead of the serial transport means. Using a number of sub-carriers allows for more, smaller rate carriers instead of one fast signalling rate. The benefit is a smaller channel bandwidth, which is less susceptible to noise or interference than wide channels [Armstrong, 2002 p37]. The channel response becomes more linear as the channel bandwidth becomes narrower.

2.3 Architecture

A communication system is made up of five mandatory components, a source, a transmitter, a receiver, a destination, and a channel. The source applies its information to the transmitter. Depending on the system type, analogue or digital, the information may be processed to ensure it is in the correct format for the transmitter. The transmitter sends user information signals over the channel, whilst the receiver picks the signal plus additive noise from the channel. The receiver passes on the decoded information to the destination where it may be converted to the appropriate signal type for its use. With a wireless communication system, air or space represents the channel as its medium. The transition through the wireless medium is not as smooth as wire-line transmission, and this poses problems, which need to be addressed by the receiver, to ensure the intelligence is decoded correctly. OFDM is primarily used in digital communication systems. Thus, if the input source signals are analogue, the information is converted to a digital stream through the use of an A/D converter. The binary stream is put through an OFDM transceiver to produce the OFDM signal.

Whilst in the transceiver, the binary stream is encoded, generally with a convolution encoder. The encoder produces a number of output bits for each input bit giving the message redundancy. Next, the encoded binary stream is put through an interleaver. The interleaver reduces the impact of deep fades in the wireless channel by moving

adjacent bits to non adjacent positions. Following the interleaver, the bit stream is modulated to symbols in the modulator functional block. The data is separated into streams that equal the amount of sub carrier channels. This is the serial to Parallel functional block. Here, pilot tones are added to the symbols and then they are modulated to OFDM symbols via Inverse Fast Fourier Transform (IFFT) block. The separate, parallel data streams are placed back into a serial data stream. The cyclic extension function block adds a cyclic extension, which is a copy of a number of sub carriers at the end of the OFDM symbol. This copy is placed at the start of the symbol. The signal converted to an analogue signal using an IQ modulation function block and the signal is up-converted to the required frequency for transmission. These stages were not added to the simulator as the simulator only tests the signal processing up to and including the intermediate frequency (IF) stage. The signal is put through a channel function block that adds White Gaussian noise (AWGN) as well as fading. The signal becomes corrupted and it is the purpose of the receiver to extract the wanted information from this signal. At the other side of the radio channel, the receiver processes the received signal, and converts it back to its original data. The imperfect channel would have changed the signal by adding noise and offsetting the phase and frequency. The receiver will need to adjust for these imperfections prior to signal demodulation using the synchronisation function block. Together with the synchronised signal and the timing signals, the cyclic extension is removed. Following the cyclic extension removal functional block, the serial data stream is split into many parallel data streams. The number is equal to the number of sub carrier channels as in the transmitter. The data streams are demodulated from OFDM symbols using the Fast Fourier Transform (FFT) function block. The resultant symbols are converted back into a serial signal stream where the pilot tones are removed. The pilot tones are compared with a set of pilot tones to find any frequency, phase and amplitude differences. The resultant timing signals are used to fine tune the data signal whilst it is being demodulated. The signal is converted to a binary stream in the demodulation function block and passed to the de-interleaver function block. Here the data stream is de-interleaved so the resultant data is now in the original order. The data is sent through a decoder function block where the original data is removed.

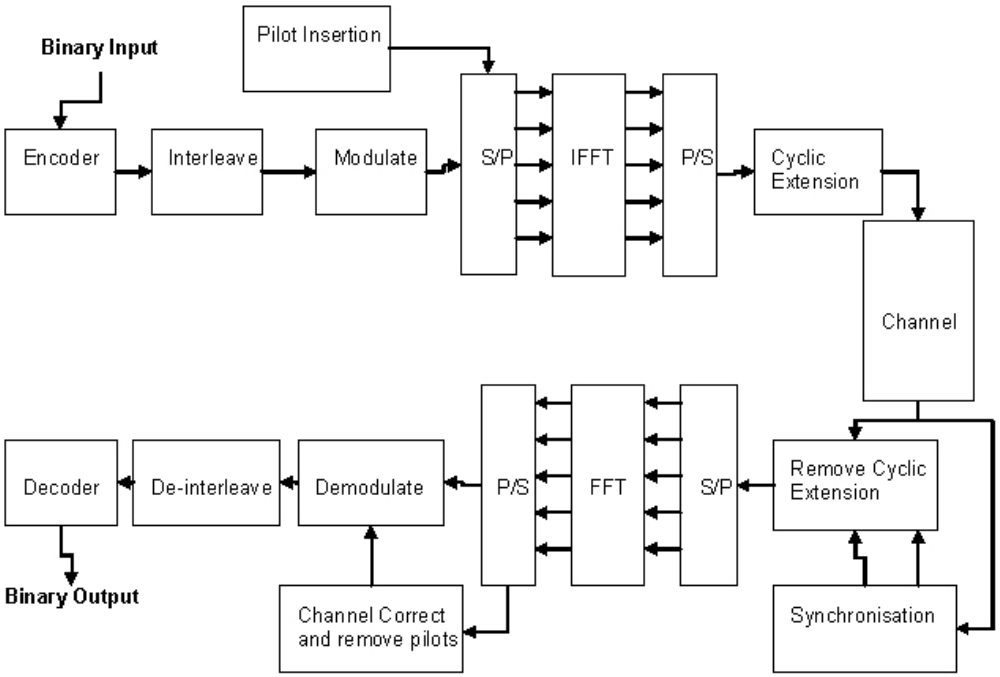

Figure 2.2: OFDM transceiver block Diagram

2.3.1 Encoding

The use of an error control coder enables the system to be more reliable. It can reduce Bit Error Rate (BER) for a given Signal-to-Noise Ratio (SNR). The cost to the system is that the signal has added redundancy resulting in a reduction in effective throughput. Depending on the chosen code rate, the original bit representation is increased accordingly. This results in the increase of the number of transmission bits in the original bit stream or message. The code rate is defined by the number of input bits, *ko*, being divided by the number of representative output bits, *no*. The code rate can be expressed as:

$$R_c = \frac{\kappa_o}{\eta_o} \tag{2.1}$$

There are different coding rates for different scenarios. The code rates,R, used in OFDM communication systems are $\frac{1}{2}$, $\frac{2}{3}$ or $\frac{3}{4}$. Whilst the $\frac{1}{2}$ rate is being used, two output bits

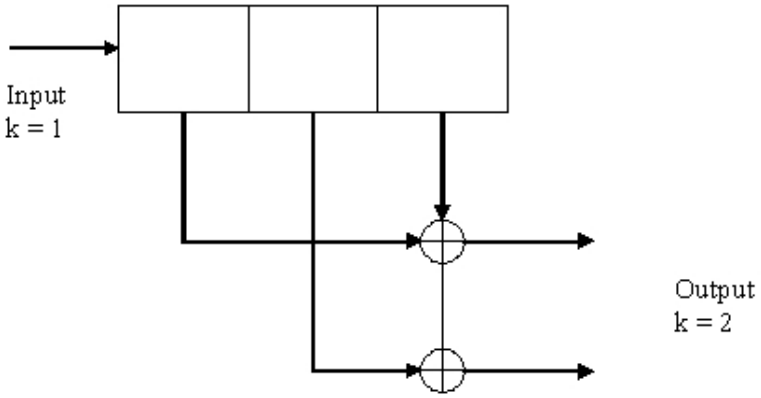

Figure 2.3: OFDM convolutional encoder diagram

represent one original input bit. This represents an increase of twice the length of the original message.

For the use of the $\frac{2}{3}$ rate, the $\frac{1}{2}$ rate has some bits omitted. When two bits are coded, making four output bits, one of the added bits is excluded leaving three bits representing the two original input bits, providing the $\frac{2}{3}$ rate. For the $\frac{3}{4}$ rate, the same principle is used. Six output bits represent three input bits but two output bits are discarded. The remainder four output bits represent three input bits or the $\frac{3}{4}$ rate. The omission of these bits is called "puncturing".

2.3.2 Interleaving

The use of an interleaver in OFDM communications systems is to try and offset the fading effects of the radio channel and its non-linear response across its frequency range. The sub-carriers in the OFDM symbol populate different frequencies. Due to the different frequencies, the sub-carrier's amplitude will be different. To reduce the impact the differing amplitude has on the coded stream, the bits are separated so that several sub-carriers are between original neighbouring bits when they are mapped to their respective sub-carriers. The coded bit stream will only have small separate bits missing instead of a large chunk, if any noise or interference occurs. The type of interleaver used in OFDM systems is generally a block interleaver. An example of a M * N block interleaver is:

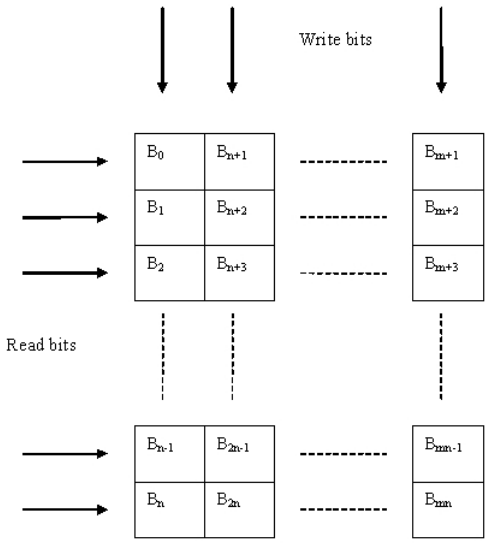

Figure 2.4: Block interleaver

As the bits are written into the block in columns, the system will read the bits out in rows.

2.3.3 Modulation

The coded bit stream is modulated into symbols to increase the efficiency of the communication system. Modulation of the signal changes the amplitude, phase and frequency of that signal. With OFDM, only the phase and amplitude is varied. The frequency is left constant to ensure the orthogonal aspect of the sub-carriers. The situation and application controls the type of modulation scheme chosen. Through the conversion of bits to symbols, a complex number represents one or more bit, depending on the scheme chosen. The modulation schemes used in OFDM communication schemes are BPSK, QPSK, 16-QAM and 64-QAM. Each scheme maps a certain number of bits to a symbol. This can be seen in their constellation maps. For BPSK, one bit represents a symbol whilst QPSK has two bits corresponding to the same symbol. 16-QAM has four bits equating to a symbol and 64-QAM has six bits per symbol. The Bit Error Rate (BER) increases for the same SNR level as the bit per symbol mapping criteria increases. The SNR needs to be higher so the removal of the bits in the receiver can be done effectively. This is due to the smaller phase difference that each modulation

scheme has when the number of points in the constellation map increases. As the number of points increases, the average power of the constellation increases as well. The average power equation:

$$P_{ave} = \frac{1}{M} \sum_{\kappa=1}^{M} \mid C_k \mid^2 \tag{2.2}$$

Where:

M is the number of points in the map and

C_k is the power of all the M points in the map.

From the formula above, it is very easy to see that a direct relationship exists between the number of points and the power of the signal. As the power of the signal can vary with the number of points, the probability of a bit error varies as well. The equation for the probability of an erroneous bit is:

$$P_b = Q\left(\sqrt{\frac{E_b}{N_o}}\right) \tag{2.3}$$

Where:

Q is the Probability Density Function of a zero mean, normal, random variable

E_b is the energy of the bit and

N_o is the power of the noise.

For BPSK, there are only two outcomes for a '0' and a '1'. These results are separated by 180^o. The constellation map for BPSK is:

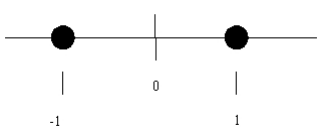

Figure 2.5: BPSK constellation map

When the data goes through the noisy channel, the data points will not be exactly on the constellation point as above. The difference between the exact and where it does occur is called the vector error. With BPSK, it does not have any imaginary factors as part of the complex number. This leaves the result to be on the real axis either side of the zero mark. The receiver can decide what the data point is supposed to be, either a '0' or a '1', depending on which side of the zero point the data is. For QPSK, there are four points in the constellation map. Each point has a real part and an imaginary part that makes up the complex number. This means, besides having points on either side of the zero line in the real dimension, it also has points either side of the zero line in the quadrature dimension. The area where a data point is, after being affected by noise, is only a quarter of the map if it is to be analysed as correct. If the data point has a phase change greater than 90^o, it will fall into a different quadrant, the receiver will interpret it as a different data point and an error will occur. To ensure errors are minimised, the SNR needs to be larger than for the BPSK scheme. The receiver must decide whether the data point is one of four points as opposed to one of two points in the BPSK modulation scheme. The constellation map for QPSK is:

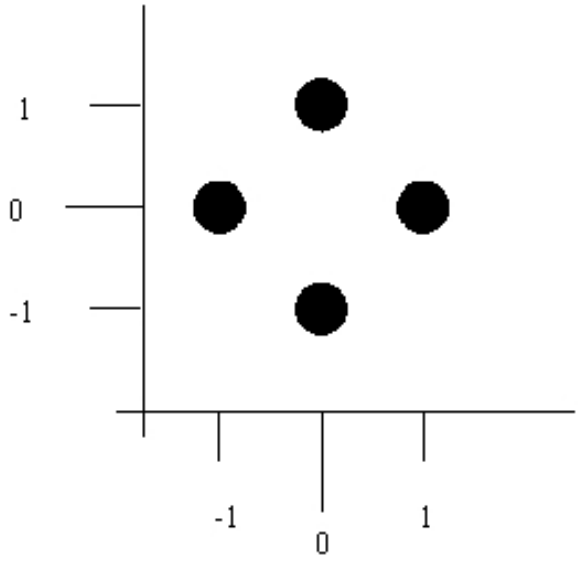

Figure 2.6: QPSK constellation map

The 16-QAM modulation scheme has sixteen points of which the receiver needs to ensure the data is correct. This allows for only a phase change of 45^o before it becomes an error. As well as signal phase changes to depict different data points, QAM schemes, also, changes the amplitude of the signal. Two aspects of the signal are needed to be correct to ensure the correct data is retrieved as opposed to one for the PSK schemes. The SNR of the signal needs to greater to enable the receiver to interpret the signal correctly. If the amplitude or phase varies, an error can result. The constellation map for 16-QAM is:

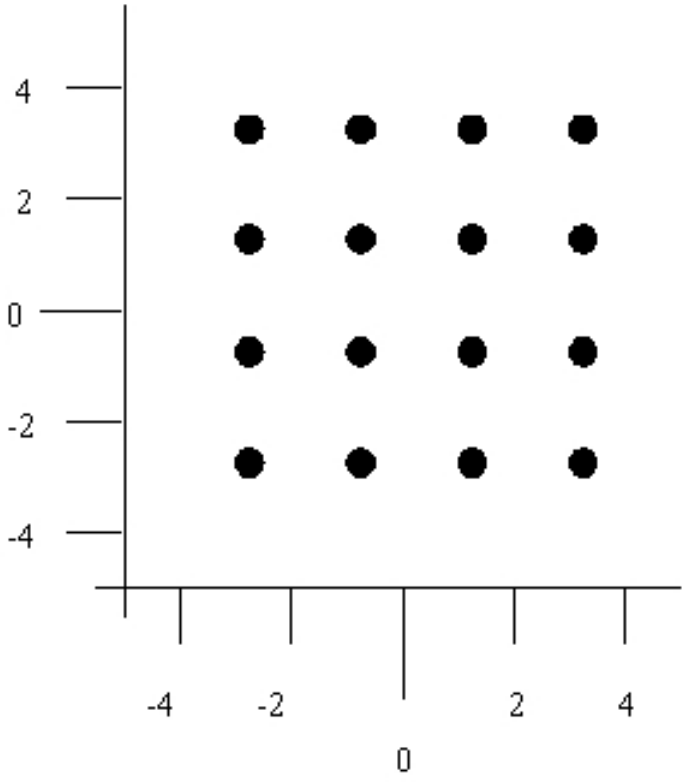

Figure 2.7: 16-QAM constellation map

The final modulation scheme used is 64-QAM. This scheme has sixty-four data points in its constellation map. The phase difference that the scheme allows before an error would occur is 22.5^o. This is half of the previous scheme, 16-QAM, so the data points need to be more precise to ensure the receiver correctly deciphers the signal. As well, there are four levels of amplitude the scheme uses. The signal will need a larger SNR to ensure the data is correctly extracted from the signal. The constellation scheme for

64-QAM is:

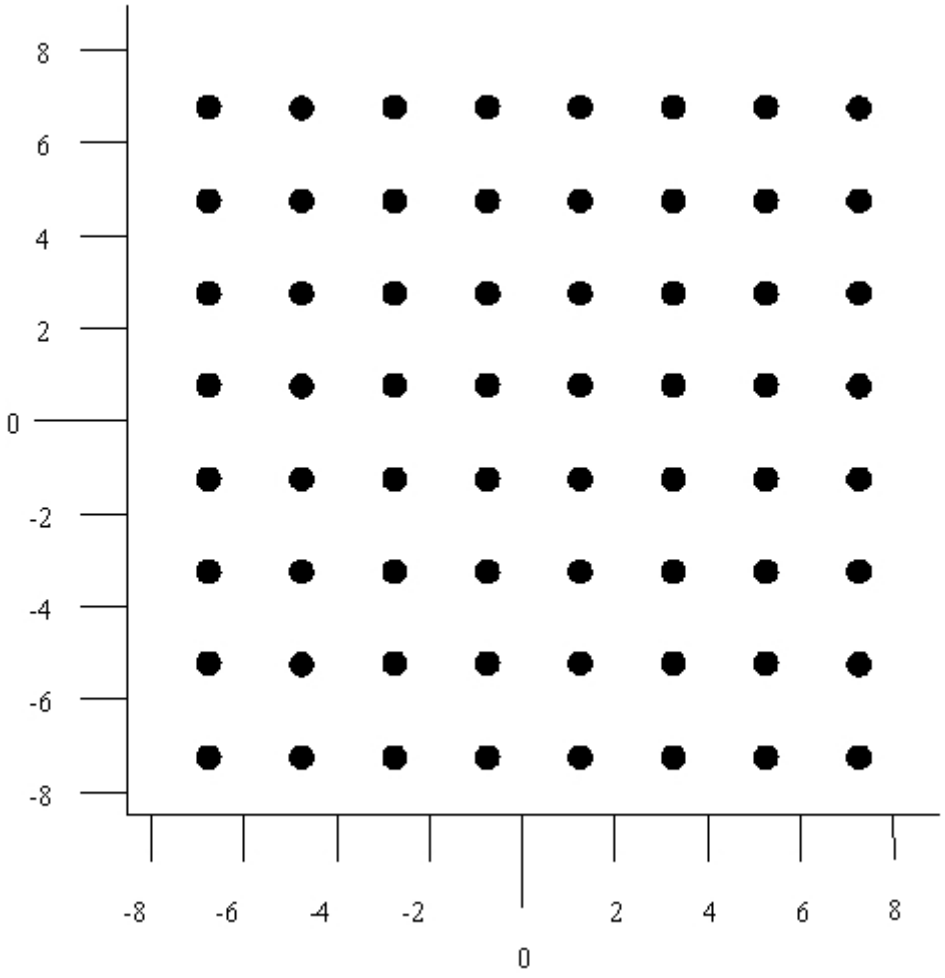

Figure 2.8: 64-QAM constellation map

2.3.4 Pilot Tone Insertion

The transmitted signal, passing through a wireless channel, will be affected by a frequency and phase shift as well as noise. To help a receiver to extract the data from the received signal, pilot tones are inserted into the OFDM signal. These signals are positioned throughout the OFDM symbol for the maximum effect of being able to help the receiver detect a change in the frequency or phase of the transmitted signal.

The synchronisation data the pilots carry, add to the overall data that is sent, but are

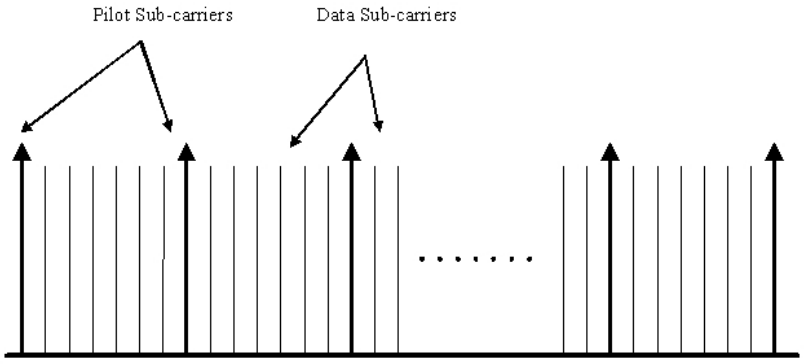

Figure 2.9: Pilot and data sub-carriers

considered overhead signals. The synchronisation of the receiver to the transmitted signal enables the receiver to extract the wanted data. The spacing of the pilot tones depends heavily on the type of wireless channel, its bandwidth and the coherence time. The more noise in the channel, the more help and time the receiver needs to synchronise to the received signal. However, as the pilot tones do not carry any information data, they add to the overall noise of the bandwidth and therefore decrease the SNR. The amount of pilot tones becomes a trade-off between channel performance and SNR loss. This becomes more apparent when the transmission is used for packet transfer. The delay due to the processing of the pilot tones slows the system and ultimately the data throughput. Through knowledge of the maximum delay spread, τ_{max}, the minimum spacing of the pilot channels can be calculated. The spacing needs to be less than the maximum delay to ensure the integrity of the data, that is:

$$Spacing \quad = \quad \frac{1}{\tau_{max}} \tag{2.4}$$

Where:

τ_{max} is the maximum delay spread.

Taking a typical mobiles base station whose coverage area has been set for approximately 5 km, the maximum delay is round trip time divided by the speed of light, c.

The delay for the site is:

$$Delay = \frac{(5000 \times 2)}{3 \times 10^8}$$

$$= 33.33us$$

With a delay of 33us, the minimum spacing is 30 KHz and also having knowledge of the sub-carrier frequency spacing, the position and number of pilot sub-carriers can be found.

The pilot tones have complex data values of 1 + j0 and -1 + j0. As seen, they only have a real part and no imaginary part. These values are stored in the receiver as well for comparison. Through comparison of the pilot tones in the receiver with those received as part of the OFDM packet, an estimation of the phase shift, frequency and timing offsets and amplitude change can be deduced. This estimation can be used to compensate for these alterations to the signal.

2.3.5 IFFT

Once the pilots have been inserted into the data symbols, the data is put through an Inverse Fast Fourier Transform (IFFT). This maps the complex data symbols to a Time Domain OFDM symbol. The OFDM symbol is made of a number of discrete, base-band and orthogonal sub-carriers, which carry the data symbols and other, required timing information. Not all sub-carriers are used for data and pilot information. There are some sub-carriers that are used as guard barriers and preamble at the start and finish of the OFDM symbol. To enable the most efficient use of the IFFT function, the number of sub-carriers is kept to a power of two, namely 2^n. Prior to the advent of Digital Signal Processors (DSPs), which allow the IFFT to be performed in a single chip, a bank of mixers, oscillators and filters performed this function. This amount of equipment meant that the size and weight were limiting factors in the use of this type of communication.

2.3.6 Fourier Transform

With the analysis of communication systems, the understanding of the relationship that signals have in both the time and frequency domains becomes very important. Many systems studied in the frequency domain are given an input as a random, time varying signal. One of the tools used to provide the relationship is the Fourier Transform (FT). Given a periodic, complex input signal, with a period of T_o, is:

$$x\left(t\right) \quad = \quad Ae^{j2\pi f_o t} \tag{2.5}$$

Where:

A is the amplitude of the signal and

f_o is the frequency of the signal in Hertz.

The orthogonal augmentation of the signal is represented by the Fourier series coefficients. The Fourier series coefficients of the signal $x\left(t\right)$ are:

$$x_n \quad = \quad \frac{1}{T_o}\int_{\alpha}^{\alpha+T_o} x\left(t\right)e^{-j2\pi t\frac{n}{T_o}}\,dt \tag{2.6}$$

Where:

T_o is the period of the signal

α is a constant indicating the lower boundary and

$x\left(t\right)$ is the input signal.

Taking note of the term:

$$\frac{n}{T_o} \quad = \quad nf_o \tag{2.7}$$

Where:

T_o is the period of the signal and

f_o is the frequency of the signal.

and that the fundamental frequency is f_o, the coefficients represent the harmonics of the fundamental frequency. The coefficients are generally complex values regardless of the signal being real or complex.

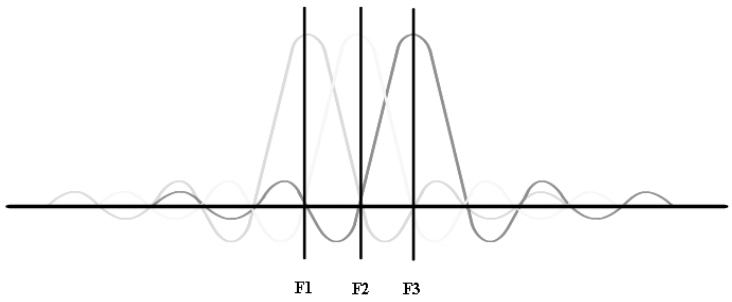

Figure 2.10: Orthogonal sub-carriers

2.3.7 Cyclic Extension

A cyclic extension is added to the OFDM symbol to ensure the sub-carriers remain orthogonal to each other. The extension is a repeat of the later section of the proceeding OFDM symbol but is added to the front of the next OFDM symbol.

The length of the extension is chosen so that the maximum multipath delay incurred in the radio channel is smaller than the extension length. If the delay was to become larger than the extension, inter-symbol interference (ISI) would occur. The sub-carriers closest to the ends of the symbol would be affected first and as the delay increases, more and more sub-carriers become affected thus rendering the system inoperable. The orthogonal sub-carriers are, also, affected which leads to inter-carrier interference (ICI) [Engels, 2003 p37].

The efficiency and SNR of the signal becomes compromised when the length of the

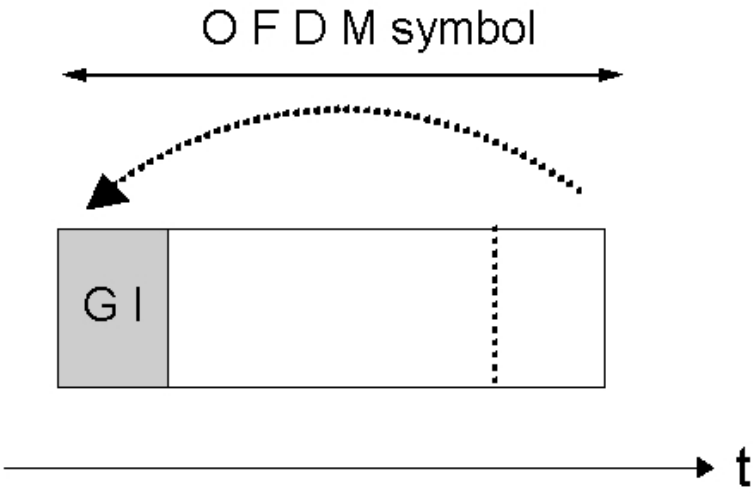

Figure 2.11: Guard interval

cyclic extension is made too large. As the data in the extension is not being used for information, the efficiency of the system is being reduced. The transmitted energy of the extension adds to the noise level, therefore leading to a reduction in the SNR. The size of the extension needs to be carefully chosen for the most optimum performance in the given environment. The extension length can be as low as 10% of the useful information block length if they are long information blocks i.e. $\geq$ 128 sub-carriers [Hanzo et al, 2003 p42].

2.3.8 Transmission

The OFDM symbol is now converted to a serial data stream, modified to an analogue signal and up-converted to the required frequency for transmission. The OFDM symbol can be written as:

$$s(t) = \Re\left\{\sum_{i=-\frac{N_s}{2}}^{\frac{N_s}{2}-1} x_{i,k}\, e^{j2\pi\left[f_c+\frac{i}{T_{fft}}(t-kT)\right]}\right\}, kT \leq t \leq kT+T \qquad (2.8)$$

Where:

N_s is the number of sub-carriers

$x_{i,k}$ is the complex data and pilot signal

f_c is the OFDM centre frequency

i is the sub-carrier index

k is the transmitted symbol index

T is the OFDM symbol period and

T_{fft} is the effective OFDM symbol period.

The signal is emitted into the wireless channel. A continuous, transmitted OFDM sequence can be expressed as:

$$s_{rf}(t) = \sum_{k=-\infty}^{\infty} S_{rf,k}\ (t - kT) \tag{2.9}$$

Whilst in the channel, the sequence will be affected by noise and other interferences. For effective removal of the data from the signal, the receiver will need to account for the effect on the signal.

2.3.9 Receiver

The OFDM receiver implements the reversal of the processing that occurred in the transmitter with some extra timing functions. The receiver needs to be synchronised in both frequency and phase with the received signal to ensure the correct bit stream data is extracted. After the Radio Frequency (RF) signal has been reduced to its base-band signal, the receiver searches for a pre-defined sequence or preamble. This sets the boundaries for each OFDM transmission. Once the boundaries have been found, the frequency offset needs to be found. This is done through a channel estimate using the pilot tones in the OFDM symbol. This is a coarse correction with fine-tuning occurring after the Fast Fourier Transform (FFT) function. The cyclic extension is removed prior to the FFT. The FFT is utilised and the corrections are implemented to the incoming signal. The signal and receiver are now fully synchronised to each other. The signal is

demodulated, de-interleaved and decoded. The final product is the digital bit stream. If the result needs to be an analogue signal, it is further processed through a Digital to Analogue Converter (DAC).

2.4 Chapter Summary

Understanding the structure of an OFDM transmission system aids design to make the most efficient use of the wireless channel and to further develop the technology into a more flexible tool in communications. Although the process of transmitting an OFDM signal may seem to be very complex, the advantages of a lower error rate, with the use of many of the functional blocks, at a lower SNR level outweigh the constraints.

Chapter 3

Radio Environment

3.1 Propagation

Radio waves travel through a medium, usually free space or air, and become attenuated. This reduction of signal power is due to many signal and system parameters. When a signal is being transmitted, the radiated energy is in different forms and in different directions. Some of the energy goes towards the sky and is used in long distance transmissions. These are sky waves. Some of the energy is directed, so it is parallel to the Earth's surface. These are direct or space waves and are used in short transmissions. The last type of signal propagation is the type where the energy is directed at the ground and is reflected so it 'hops' around the Earth. These are ground waves and are used for both short and long distance transmissions. The type of transmission waves is mainly determined by the antennae array and the frequency. Multiple access wireless systems used in Australia use frequencies below 10 GHz. The transmission waves for these systems are made up of direct / space and ground waves as the two ends, transmitter and receiver, usually can be seen by each other. This is called Line of Sight propagation (LOS).

As the frequency gets higher, the waves become more like sky waves and can penetrate the ionosphere and are used for transmission in space or beyond. For systems that use the LOS principle, the signal is attenuated by distance and frequency. As the

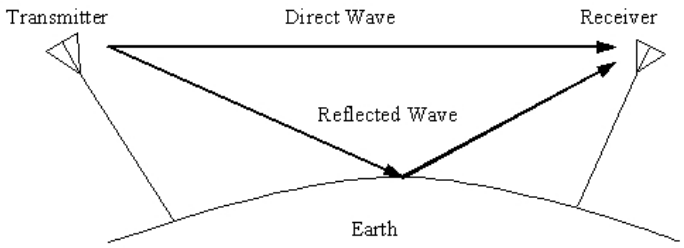

Figure 3.1: Line of Sight (LOS)

distance gets longer, the signal gets proportionally weaker. As frequency gets higher, the loss is proportionally greater as well. The loss is accounted for through the Earth's atmospheric contents including water vapour, various gases molecules and free electrons. The atmosphere contains a high proportion of water and oxygen and the loss is high when the water and oxygen absorption is at its peak of around 21 GHz and 60 GHZ respectively [Young, 1985 p681]. As the frequency gets lower, the free electrons play a larger part in the loss factor of the path loss.

The LOS path loss formula is:

$$Path\ Loss \quad = \quad 32.4 + 20\ \log(f)\ +\ 20\ \log(d) \tag{3.1}$$

Where:

Path Loss is in dB
frequency f is in MHz and
distance d is in km.

The above formula states the relationship between distance (d), frequency (f) and the channel loss. This is used extensively to design radio systems and to find their coverage footprint. Other factors can add to the propagation loss. These can be heavy precipitation e.g. snow, rain or from tree foliage growing in front of the antennae thus causing signal degradation. These issues can be allowed for by ensuring the receive signal is always greater than the noise level of the receiver. This is called Fade Margin.

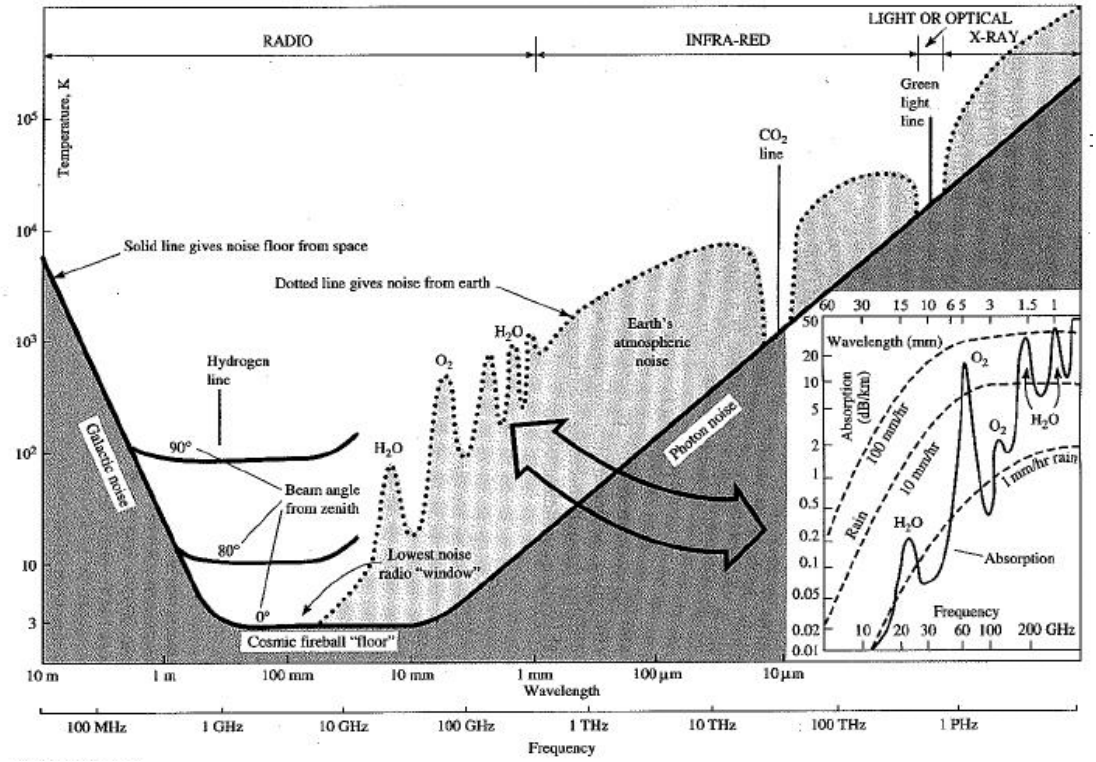

Figure 3.2: Attenuation from radio to x-rays, insert shows weather effect

Source: Kraus et al, Figure 5-59, p336

3.2 Multipath

The transmitted signal will reach the receiver at different times and signal strengths to each other. These signals would have come directly from the transmitter or have been bounced off nearby buildings or obstructions [Armstrong, p16].

As the distance travelled is different, their time of arrival and signal strength will vary. This affects the phase of the signal with respect to each of the different received signals. If any signals are completely out of phase, destructive signal addition occurs and the received signal will be cancelled. As the received signals have different strengths and phases, a signal is usually recovered. The phase difference should be kept to a minimum so the received signal is at a maximum after constructive signal addition has occurred. This phenomenon is called Rayleigh Fading. Rayleigh fading can cause fading to occur very quickly and within a wavelength [Miller, 1993 p323]. For mobile users where

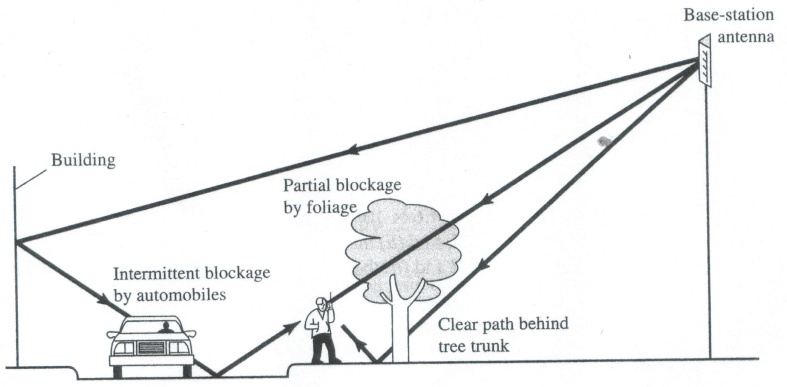

Figure 3.3: Multipath

Source: Kraus et al, Figure P5-11-20, p352

the transmitted signal is around the 1 GHz, the wavelength is less than a metre so small movements can cause a connection to struggle to maintain connectivity. In urban landscapes where there are a lot of obstacles, this type of fading occurs regularly and can be combated by having a significant fade margin of 20 dB or more or the use of diversity [Miller, 1993 p323]. Some of the different types of diversity are frequency and space diversity.

3.2.1 Frequency Diversity

Frequency diversity uses different frequencies with matching transmitters and receivers to combat fading. All radio channels suffer from a spectral response that is not linear across its range. At various frequencies, the response can be very poor. This occurs because the frequencies destructively cancel each other out. At other frequencies, they add to each other. This type of response causes narrow band transmissions to be affected more greatly than wide band transmissions. If a null in the spectrum occurs at the wanted frequency, the complete signal can be lost. The use of multiple frequencies is incorporated into OFDM modulation [Chuang, 2000 p4]. It uses many narrow band sub carriers and if one or more of the sub carriers has a poor response, the system still allows data to be transferred using the other frequencies. The loss of the signal power of one or more sub-channels can be overcome with the signal power of the remaining

sub-channels instead of a complete loss of signal.

3.2.2 Space Diversity

Space diversity employs the use of multiple receivers, spaced many wavelengths apart. Using antennae set apart by many wavelengths, the receive signal between the two antennae would not have a null or large phase difference. This allows for resultant higher average signal strength as the received signals add together. Many of Australia's network providers use external configurations where multiple antennae are used for space diversity as well as redundancy for when problems exist in the external plant. If a problem does exist in the external equipment, only mobile capacity is affected. Coverage is still provided with the use of the remaining external equipment.

3.2.3 Delay Spread

When multipath occurs, the received signal contains a direct route signal plus many reflected signals. The reflected signals have travelled a longer route so the signal has some delay compared to the direct route signal. Also, with different frequencies, the delay is not linear. As the frequencies increase, the delay also increases. With all these delays in the receive signal, the signal is spread over a wider time frame. The time between the first and last significant received signal is called delay spread [Intini, 2000 p20]. This phenomenon causes inter-symbol interference (ISI) if it is large enough to affect the received signal. With OFDM Modulation, ISI is negated with the use of a cyclic extension. This is a partial repeat of the data being sent. If the delay spread is smaller than the cyclic extension, no ISI should occur. If it is larger, then ISI becomes an issue that needs to be addressed. The length of the extension is chosen to be enough to combat the greatest delay that would be encountered, thus keeping ISI under control. Another way of keeping delay spread under control is to reduce the data rate. By doing this, the symbol has a longer time period between different states and therefore can be more resilient to the delay. This slows the systems throughput if it only has one channel. OFDM modulation reduces the data rate but only to each sub-channel. The overall data rate is a combination of the sub carrier's rate multiplied by the number of

sub carriers utilised. The original data rate can still be supported and the impact of the delay spread can be negated at the same time.

3.2.4 Doppler Effect

The frequency of the reflected received signal can vary if there is motion between the transmitter and the receiver [Miller, 1993 p544]. In mobile communications, the user is generally moving towards or away from the radio base station. This affects the received signal frequency. As with a motorcycle that is approaching you as a bystander, the motor sounds different when it is driving away from you. The frequency of the motor is higher as it approaches you and lower as it drives away. This change in frequency is called the Doppler Effect and has to be accounted for when synchronising the receiver to the transmitted signal. The amount of frequency shift caused by the Doppler Effect is:

$$\Delta f = \frac{2\ v}{\lambda} \tag{3.2}$$

Where:

v is velocity in metres per second and

λ is the wavelength in metres.

For OFDM modulation, the frequency shift can cause a loss of orthogonality which result in errors occurring in the data transmission.

3.3 Noise

Noise is unwanted signals that appear at all receivers' input when the wanted signal is not there. Noise can also drown out the wanted signal at the receiver if it is strong enough. The level of noise directly impacts the SNR and therefore the capacity of the

channel. Many processes have to be employed to ensure the noise effect is kept to a minimum and our desired signal is extracted. The effect the noise can have on the received signal is to change the amplitude of the signal in both the real and imaginary parts as well as the phase of the signal. These aspects are the two parameters in OFDM transmission can change the received data from being correct to an erroneous bit.

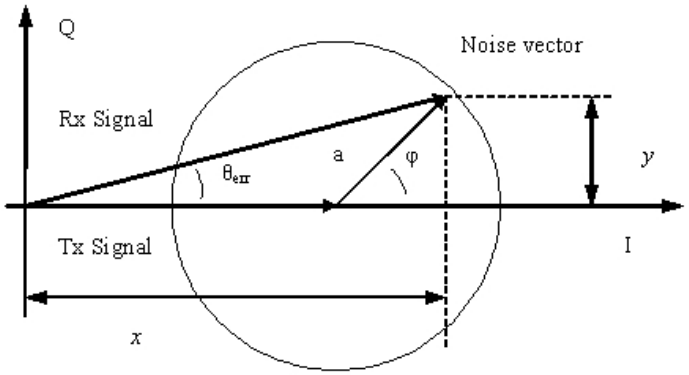

Figure 3.4: Noise vector effect

The noise can come from many sources but are classed as two different types. They are external and internal noise.

3.3.1 External Noise

The external noise can be either natural or man-made. The naturally occurring noise comes from events that happen in and around the Earth's atmosphere. These mainly happen when storms are occurring and cause problems for receivers close to the storm. This can be seen when watching television and lightning strikes nearby. The picture suddenly becomes full of static when the strikes occur. The strike emits frequencies across a broad spectrum but as the frequency increases the effect diminishes very quickly. Also, if there is heavy rainfall between the transmitter and the receiver, the signal can be degraded. The other type of naturally occurring noise that affects radio transmission comes from events in space. The most commonly known effect is solar noise. This occurs approximately every 11 years, when sun spot activity reaches a

peak, and is very unpredictable. Other cosmic noise occurs from the stars but as their distances are very large, the effect is very minimal. The Earth's atmosphere helps to absorb a lot of this cosmic noise but there is still enough getting through to make radio transmissions unpredictable. Man-made noise from other mechanical devices causes the most impact on radio transmissions. This type of noise is produced by switch gear operating machinery, ignition systems, electrical motors, mast head amplifiers, lights, other communication systems and any equipment that switches power. The equipment causing the noise may have become faulty and is transmitting it in all directions surrounding the equipment or may not have any electro magnetic interference (E.M.I) barriers to stop the noise from emanating from it. This has created a lot of debate in the last decade and manufacturers are taking E.M.I issues very seriously now when designing new products. Many faulty mast head amplifiers transmit out of the antennae and cause high levels of noise that interfere with communications systems. As wireless communications becomes more prevalent, the spectrum becomes more and more congested with radio signals that interfere with other communications systems. This creates more noise and therefore other systems have to transmit at higher levels to keep their connectivity. This adds more noise and the process keeps escalating. As this is occurring more in urbanised areas, very sensitive equipment is located in remote areas where the man made noise is at its lowest. This type of equipment is used in satellite and space exploration communications.

3.3.2 Internal Noise

Internal noise is present in all equipment. This is produced by the components in the receiver. The main types of internal noise are thermal, shot, flicker, burst and transmit time noise. Thermal noise is caused by the electrons interacting with ions at the atom level. The vibrations, caused between an atom and an ion interacting, creates heat which varies the rate of electron flow. With varying rates of electron flow or current, voltages are produced. This is the thermal noise voltage [Young, 1990 p118]. The frequency content of this noise is extremely broad and is affected by temperature. This type of noise can be referred to as White noise, as well, due to the colour white having all the colours of the spectrum. This noise is proportional to the bandwidth of the

system used so the wider the bandwidth, the larger the noise level. The noise level is:

$$N_o \quad = \quad \kappa TB \tag{3.3}$$

Where:

κ is Boltzmann's constant $(1.38 \times 10^{-23}\ J/K)$

T is the temperature in Kelvin (*usually* 290 *indicates ambient temperature*) and

B is the bandwidth in Hertz.

The noise level of any system can be determined by the bandwidth of the system. It is always advisable to keep the bandwidth as small as possible. Noise floor is the most important factor that limits the usefulness of any receiver. When designing a communication network, an indication of the level of noise is important. The formula for thermal noise is used extensively to gauge the noise floor of the system. For a 20 MHz bandwidth system, such as WLAN, the noise floor is:

$$N_o\,(dBm) \quad = \quad 10\ log\ \left(1.38 \times 10^{-23}\ times\ 290\ \times\ 20 \times 10^6\right)\ +\ 30$$

$$= \quad -101\ dBm$$

Once designers know the noise floor and using the path loss formula, they are aware of how well a communications network will meet the needs of the users. With all receivers the first stage has the most impact on how the noise will affect the signal level. The choice of components is very important if the receiver has any chance of extracting the data from the signal. These components are manufactured so they produce the minimum amount of noise possible. Shot noise occurs when a bias current in a discrete component is not in a steady motion as it crosses the potential junction. As with all electronic components, a standard or similar P-N junction exists. The junction is a barrier to the electron flow until a bias current is applied that allows the electrons to flow. As the electrons flow, they flow in a random fashion due to the direction and distance being different for all electrons. The anode of the semi conductor heats up

due to the electrons flowing to this electrode and the name "shot" is derived from the electrons bombarding the anode. With transistors and other multi junction devices, the shot noise is increased based on the number of junctions that exist and the amount of bias current needed. The shot noise and thermal noise add to each other to give an overall noise effect in the receiver. This noise is purely random in its nature and is flat across the frequency spectrum. Flicker noise is associated with a bias current in semi conductors and is caused by the defects in the manufacturing of the electronic junction. The noise is proportional to the bias current but as frequency increases, the noise level decreases. As the response is not flat across the frequency spectrum, the noise is referred to as Pink noise. The flicker noise is random and is very device specific depending on the number of defects occurring in the manufacturing process. Burst noise is very similar to flicker noise but it is caused by the contamination of the ions used in the production of the semi conductor junction. It is proportional to the bias current but decreases more quickly as frequency increases. The bias current increases suddenly for short periods of time before returning to its original level. This is also known as Popcorn noise due to the popping sound it produces if it is amplified. The burst noise is random and like flicker noise is very device dependent due to the manufacturing process. Burst and flicker noise are not a problem for applications in the wireless data communications industry as the frequencies used are generally in the Very High Frequency (V.H.F), Ultra High Frequency (U.H.F) or above frequency ranges. Transmit-time noise occurs when the wireless application is close to the maximum boundary of the chosen components. When the period of the frequency becomes similar to the time taken to cross the semi conductor junction, some of the signal can become diverted to other junction points [Miller, 1993 p11]. This increases the current in other areas of the semi conductor, therefore increasing the level of noise. This is only a problem if the chosen components are not suitable for the application, or higher than expected frequencies are occurring. A well designed receiver should allow for any frequencies that are seen at the receiver and any frequencies that should not be there are filtered out prior.

3.3.3 Signal to Noise Ratio (SNR)

SNR indicates the relationship of the wanted signal to the noise level of the receiver. Given the noise floor using the formula for thermal noise, any signal received that is above this level is the SNR of that signal. This is an indication of the quality of the wanted data prior to it being processed. The higher the SNR of the received signal will give better quality of the processed data. If the SNR is too low, the noise will degrade the signal to the point where it will be unusable. Although a good SNR figure is needed to ensure the data being extracted from the receiver is the wanted information, the noise figure the receiver has to be considered. The receiver's noise figure is an indication of the impact its own internal noise will have on the signal. Although ideal devices inject no noise into the signal, it has been found that in practical situations, noise is universal and needs to be considered. The noise figure of a receiver is always designed for a very low result so the additive noise is minimised. In communication systems where the frequencies used are above 1 GHz, the noise figure value is replaced with a value called Equivalent Noise Temperature (T_{eq}). The T_{eq} value highlights a greater variation in the noise level and it is easier for designers to work with to produce working circuits, especially in systems based in space [Young 1990 p746].

3.4 Interference

Interference is an unwanted signal, causing errors with the wanted signal of our communications system. There are four main types of interference. These are Inter-Symbol Interference (I.S.I), Narrowband, Wideband and Intermodulation.

3.4.1 Inter-Symbol Interference (I.S.I)

ISI is caused between the different symbols in the data stream of digital communication systems. As the data is transferred through a non linear phase response radio channel, some frequencies of the symbol are lost. The effect is a rounding of the pulse and the next pulse is affected because the response is slower than the transmission rate. As the

transmission rates get higher, the effect gets greater till errors of an un-sustainable level occur and the data is lost. To compensate for this type of interference, a filter that is matched to the channel is used. One of the most commonly used filter types is the Raised Cosine filter. One of the main reasons ISI is an issue is that the channel cannot handle the higher transmission rates. As the rates get higher, they start to approach the channel capacity. Shannon stated that the channel capacity is:

$$C = BW\ log_2\,(SNR\ +\ 1) \tag{3.4}$$

Where:

C is the channel capacity,

BW is the bandwidth and

SNR is the signal to noise ratio (not in dB).

When the transmission rate approaches the channel capacity, the number of errors starts to increase. The formula shows that if the spectrum is noisy, a reduction of the bandwidth to enhance the SNR also reduces the channel capacity. The capacity becomes a trade-off between bandwidth and SNR. Another tool to help keep the integrity of the system in noisy environments is to employ the use of error correction coding. OFDM modulation can use Forward Error Coding and Interleaving to improve the SNR and offset the non-linearity of the channel.

3.4.2 Narrowband Interference

Narrowband interference is interference that is very narrow in relation to the transmitted signal. It can reside in the receiver's spectrum and only cause problems to part of the signal. The communication system can still operate but at a reduced efficiency. The efficiency is dependant on how much impact the interference has. It is usually created by external sources. For OFDM modulation, it has very good narrowband rejection facilities due to the many sub carriers it employs. If the narrowband interference causes problems with some of the sub carriers, these sub carriers are blocked from being

used and the data is still carried on the unaffected sub carriers. A typical example of narrowband interference is mast head interference. This happens when the masthead amplifier for a household television system goes faulty. It transmits out a very narrow frequency, similar to a Delta function, equal to the internal oscillator's frequency. As the day goes on, the frequency shifts with the heat of the sun. It creates problems for users of Australia's CDMA and GSM mobile phone networks. As the interference gets wider, more and more sub carriers are affected till the system becomes inoperable.

3.4.3 Wideband Interference

Wideband interference causes the communications systems to become unusable till the interference is removed. They have bandwidths that are similar to or larger than the communication system that is being used. The sources of these types of interference can be classed in two areas. They are Interference and Intermodulation Distortion (IMD). As with narrowband interference, wideband interference is interference that is usually created by external sources. An example of this is faulty fluorescence lights. The pulses from these lights create a broad band signal that moves through the receiver's spectrum making it unusable.

3.4.4 Intermodulation

Intermodulation occurs when two frequencies add or subtract together causing resultant frequencies to interfere with the communication system. These products occur at non linear junctions, such as poorly designed mixers and combiners, faulty connections or antennae. These second or third order products usually fall outside narrowband systems but when a broadband system is being used, they fall inside the receive spectrum. The products ($2f_1 - f_2$, $2f_1 + f_2$, $2f_2 - f_1$ and $2f_2 + f_1$) have amplitude that can interfere with the system if it is not carefully designed. In systems where the desired frequency is close to another frequency that is much higher than the desired frequency, the unwanted frequency can still get into the receiver and mix with the wanted frequency. This causes mixing products that can interfere with the information signal. As well, systems that use the same antennae system, such as, In-Building Coverage Distributed

Antennae Systems (IBC / DAS), the mixing of two transmit signals can occur in the DAS components resulting in a frequency near the wanted receive frequency. To combat this occurring, an intermodulation level of -140dBc needs to be attained when using carriers around + 43dBm (20W).

3.5 Chapter Summary

A wireless channel is a very complex segment of the communication system that impacts the quality of the signal immensely. Through the impact of noise, in its various forms, the signal gets degraded. Having the signal and system robust enough to offset the impact makes it very complex. Different tools can be utilised to enable the signal to be received in a state where the information can be extracted. A system designer would need to be aware of the constraints that noise has on the signal to ensure that a properly designed system is implemented to cope in the extreme and varying conditions.

Chapter 4

Current Implementations

OFDM modulation is very flexible and is being used in many applications now. It can be adapted to any situation so maximum efficiency can be attained. Some of these applications are ADSL, DAB, DVB, WLANs, WMANs and FLASH.

4.1 ADSL

OFDM modulation is used to provide high speed data connectivity across the standard copper cabling that goes to every phone in every household. Whilst telephone cable was traditionally used for adhoc, voice connections, Asymmetric Digital Subscriber Line (ADSL) is a continually connected, ready to go service. The biggest advantage for ADSL is most of the infrastructure is already in place, thus providing a huge cost saving in supplying the product. With ADSL, the transmit speeds vary with the type of traffic. The faster speeds are used for downlink traffic, whilst slower speeds are used for uplink traffic. The traditional services of the line are still kept and the new ADSL services do not interfere with this service. There are many variations of ADSL ranging from the standard ADSL to ADSL4 which can provide speeds up to 52 Mbps across a 3.75 MHz bandwidth. [DSL White Paper, 2005 p7] For standard ADSL, the spectrum is broken into various sections. The original spectrum (0 to 4 KHz) is reserved for voice calls whilst the spectrum above this frequency is used for data. The signal is split into

255 sub carriers, or bins. They are spaced 4.3125 KHZ apart. The frequency spectrum of a copper line used for ADSL is from DC to 1.104MHZ. The first 4 KHz is reserved for voice calls, followed by an unused guard band from 4 KHz to 25 KHz. This spectrum comprises of 7 bins which are disabled. The spectrum from 25 KHz to 138 KHz is utilised for Uplink and comprises of 24 bins. The following spectrum, from 138 KHZ to 1,104 KHz, is used for Downlink and comprises of 224 bins. More bins are reserved for downlink traffic as experience has shown that there is a substantial amount more downlink traffic than uplink traffic.

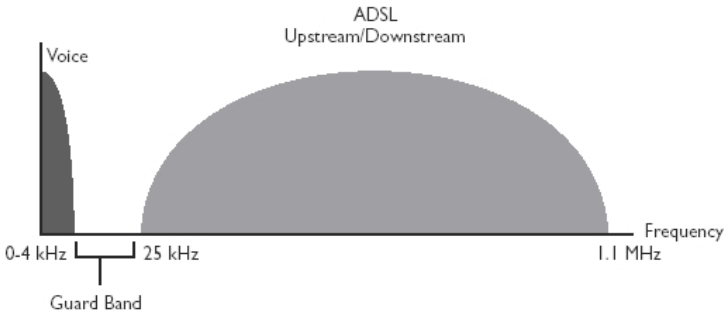

Figure 4.1: ADSL frequency allocation

Each bin is continually monitored and when the SNR level falls below 6 dB, the bin is blocked. This ensures the most efficient use of the connection at all times.

4.2 DAB

The Digital Audio Broadcast (DAB) standard was designed and implemented for the distribution of digital audio services. The system is very flexible as it can be used in different network configurations due to the four different frame structures it can utilise. They are 96 ms (Transmission Mode I), 24 ms (Transmission Mode II and III) and 48 ms (Transmission Mode IV). Mode I is to be used in the Band I, II and III of the UHF Band. Mode II and IV are to be used in Bands I to V which exist over both VHF and UHF bands. It is also available for use between 1,452 and 1,492 MHz (L - Band) for satellite transmission. Mode III is to be used to transmit in any configuration as long as the frequency range is below 3 GHz. The standard also allows

the use of cable as a delivery medium. The chosen mode for the cable transmission can be any of the four modes, but Mode III is preferred as its transmission frequency is very flexible. The frequency bands, VHF and UHF, are currently being used to transmit Amplitude Modulation (AM) services, Frequency Modulation (FM) Services and Analogue television. The new systems must not interfere with these current signals and systems. For all transmission modes, the channel width is 1.536 MHz and is known as the DAB block [DAB Standard, 2006 p168]. The different modes utilise different FFT sizes. For Mode I, the FFT size is 2048 otherwise known as 2K Mode. Mode II uses a FFT size of 512 whilst Mode III uses a size of 256. Mode IV uses 1024 sub carriers and is known as 1K Mode. The number of data sub carriers used is of the FFT size. The size of the cyclic extension, or guard interval, is of the FFT size for all transmission modes. A DAB transmission consists of numerous frames. Each frame is a combination of a synchronisation channel, a Fast Information Channel (FIC) and a Main Service Channel (MSC). The combinations of these sections total the frame time length that indicates the particular mode in operation. Each section is made up of a number of OFDM symbols. The total number of OFDM symbols varies for each mode. For Mode I, II and IV, the number of OFDM symbols used is 77. For Mode III, the number of symbols is 153. In each mode, the first two OFDM symbols are used for the synchronisation sub frame. Of these two symbols, the first symbol is known as the NULL symbol. The Null symbol has a different length to the rest of the OFDM symbols. For Mode I, the Null symbol length is 1.297 ms. For Mode II, it is 324 us. For mode III, it is 168 us and for Mode IV, the Null symbol length is 648 us. The length of the OFDM symbols used throughout the rest of the frame is approximately 4 % shorter at 1.246 ms, 312 us, 156 us and 623 us respectively. The increased length of the Null symbol is to ensure the start of frame is made correctly. Following the Null symbol is the Phase Reference symbol. This symbol sets the reference of the demodulator for the following OFDM symbol. The type of modulation used is Differential Quadrature Phase Shift Keying (D-QPSK). Following the synchronisation section is the FIC section. For Modes I, II and IV, there are three OFDM symbols allocated. For Mode III, eight OFDM symbols are used. The FIC block contains control information regarding parameters of the frame for the proper demodulation of the received signal and regional area information. This includes the time, date, country and network parameters that can be used to provide supplementary information or services. These can include regular traffic updates for

localised areas. The remaining 72 (Mode I, II and IV) and 144 (Mode III) OFDM symbols are used for Common Interleaved Frames (CIF). The CIF frames hold all the information of audio, data and services that are broadcasted.

4.3 DVB

The Digital Video Broadcast (DVB) standard has been designed for the transmission of digital terrestrial television broadcasting. This involves the services of Limited Definition (LDTV), Standard Definition (SDTV), Enhanced Definition (EDTV) and High Definition Television (HDTV). The frequency spectrum of this standard will be in the Very High Frequency (VHF) and Ultra High Frequency (UHF) bands. These bands are currently being used to transmit analogue television and must not be interfered with. The channel widths are 6, 7 and 8 MHz. A 5 MHz channel width has been made available for handheld terminals (DVB-H) [DVB-H Standard, 2004 p10]. There are two FFT sizes, 2048 (2K mode) and 8192 (8K mode). An extra FFT size, 4096 (4K mode) is added for handheld operation. The 2K mode is used in areas where the network coverage is small and the 8K mode is for large coverage areas. For the different operational modes, a flexible, cyclic prefix length ranging from $\frac{1}{4}$ to $\frac{1}{32}$ can be combined with different types of modulation. The available modulation types are QPSK, 16-QAM and 64-QAM and non-uniformed 16-QAM and 64-QAM. Using the above parameters, a bit rate ranging from 4.98 Mbits/s to 31.67 Mbits/s can be achieved [DVB Standard, 2004 p40]. With the added flexibility of a two tiered, hierarchical code, maximum frequency efficiency in any network configuration can be maintained at all times. The two levels of transmission, high priority and low priority, allow independent streams of the same or different data to be received by the user. A program can be transmitted in a high definition mode, which would be more susceptible to radio channel effects, whilst being transmitted in a standard definition mode. If the signal becomes degraded by the radio channel effects, the receiver can switch to the standard definition mode so the user still has reception. If the effects become large enough, reception is lost. The data frame consists of many super frames. A super frame is four OFDM frames. Each OFDM frame consists of 68 OFDM symbols. Each symbol represents 6817 sub carriers (8K mode) or 1705 sub carriers (2K mode). Each OFDM frame contains a data section and

a cyclic extension section. Throughout the data section, there are permanent and variable positioned pilot sub carriers as well as Transmitter Parameter Signalling (TPS) carriers. These pilots are used to synchronise the receiver to the incoming received signal. The TPS carriers contain data that informs the receiver what settings in the receiver need to be allocated to properly demodulate the data. These parameters refer to the coding, the constellation spacing, hierarchy information, guard interval length, transmission mode, frame number, cell identification and modulation type used. Given the TPS data, the receiver knows enough information about the transmitted signal that zero padding is not used. This allows optimum data transfer. The TPS sub carriers are D-QPSK modulated and are transmitted with the mean energy level of all data sub carriers. The input data stream can come directly from a MPEG-2 coded television signal. The data is placed into fixed packets of 188 bytes. It is coded and interleaved using a Reed Solomon code prior to it being convolutional coded and interleaved again. It is mapped to the constellation based on the transmitter parameter α. These parameters make up the TPS values that are transmitted with the data.

4.4 802.11a Wireless LAN

The 802.11a specification refers to Wireless LANs being used in the 5 GHz U-NII frequency spectrum. There are three separate frequency spectrum (5.15 - 5.25 GHz, 5.25 - 5.35 GHz and 5.725 - 5.825 GHz). The lower and middle spectrums have 8 channels across their total 200 MHz whilst the upper spectrum has 4 channels in it. Each channel is 18 MHz wide at the reference level, and is separated by 20 MHz spacing. Each channel uses 48 data sub-channels, 4 pilot channels and twelve zero set sub-channels. The total number of sub carrier channels is 64 which are a 2's multiply to allow for a more efficient Inverse Fast Fourier Transform (IFFT) function. Given the number of sub carriers is 64 and the channel is spaced 20 MHz apart, the individual sub carriers are spaced at 312,500 Hz apart. This resembles the current GSM standard of 200 KHz. A cyclic extension of 16 sub carriers is appended to the front of the signal frame to make an OFDM data frame. The data sub carriers are Gray-coded modulated with four different types of modulation schemes depending on the application. They are BPSK, QPSK, 16-QAM and 64-QAM. The binary data stream is encoded using three

different convolutional coding rates. They are $\frac{1}{2}$, $\frac{2}{3}$ and $\frac{3}{4}$. Using these coding rates coupled with the different modulation schemes, a variety of transmission rates ranging from 6 Mbits/s to 54 Mbits/s can be achieved [802.11a Std, 2003 p3]. The transmission rate can be higher, up to 72 Mbits/s, but without the use of the coding process. To ensure the data is more resilient to channel non-linearity, the data is also interleaved. The data stream that is sent out by the transmitter has been encoded, interleaved, modulated, had pilots inserted, IFFT function conducted on the data symbols, a cyclic extension is added before it is modulated onto the carrier frequency. Prior to the transmitter sending the data frames out, some synchronisation frames are sent. These frames set up the receiver so it is synchronised with the received signal. The first frame is a repetition of ten frames to allow signal detection, Automatic Gain Control (AGC), coarse frequency adjustment and timing synchronisation. The second frame is used to fine frequency adjustment and to help in the channel offset estimation. It comprises of two long training frames with their associated cyclic extension. The following frame has parameters about the data frames. The pilots that are inserted into the data frames are used to make the coherent detection more robust to phase noise and frequency offsets. The transmitter power levels are restricted to certain levels. They are 40 mW (+16 dBm) for the lower spectrum, 200 mW (+23 dBm) for the middle spectrum and 800 mW (+29 dBm) for the upper spectrum. From these power levels, it is quite evident that this system is used as a Pico or Micro wireless system that only covers an area up to a couple of hundred metres from the transmitter. Coupled with the receiver sensitivity ranging from -82 dBm to -65 dBm, it would only be used in indoor applications. The receiver uses the Carrier Sense Clear Channel Assessment (CS-CCA) to sense any signal above -82 dBm and will enter a 'busy' state till the signal disappears. This is similar to computers on a wired network when transmission is occurring.

4.5 802.11g WIFI

The 802.11g specification refers to Wireless LANs being used in the 2.4 GHz ISM frequency spectrum and is the next technological step after the 802.11b standard. The 802.11b standard is used for slower transmission rates in the same spectrum. The spectrum is from 2.4 GHz to 2.4835 GHz, a range of 83.5 MHz. Within this spectrum,

there are 14 separate channels of 22 MHz wide. Their centre frequency is 5 MHz apart starting at 2.412 GHz which means they overlap each other. Within Australia, only channels 1 to 11 are supported. Various countries support different numbers of channels from the list of 14. When multiple cells are used in a single area, adjacent or overlapping frequencies would operate with minimum errors if the channel frequencies chosen are separated by 5 channels (+ 25 MHz). As with the 802.11a standard, each channel uses 48 data sub-channels, 4 pilot channels and 12 zero set sub-channels giving a total of 64 sub-channels. The cyclic extension is a replica of the 802.11a standard which consists of 16 sub carriers. This is appended to the front of the signal frame to make an OFDM data frame. The data sub carriers are similarly Gray-coded modulated, but instead of four modulation types, there are six different types of modulation schemes that are used with OFDM modulation. They are CCK, PBCC, BPSK, QPSK, 16-QAM and 64-QAM. The D-BPSK and D-QPSK modulation schemes that are involved in this standard are not used in conjunction with OFDM modulation and therefore have not been mentioned. The type of scheme chosen is dependent on the application. There are four convolutional code rates, none, $\frac{1}{2}$, $\frac{2}{3}$ and $\frac{3}{4}$ used to encode the data stream. The additional none code rate has been added to the three used previously in the 802.11a standard. The highest data transmission speed has not been improved on but there has been extra, slower speeds introduced. These rates, 1 Mbit, 2 Mbits, 5.5 Mbits and 11 Mbits do not use OFDM modulation for their data transfer. They use D-BPSK, D-QPSK, CCK and PBCC. The transmitter will emit synchronisation frames prior to emitting data frames. With the different types of rates and transmission speeds, there are three different synchronisation preambles. They are Long Preamble, Short Preamble and Extended Rate Preamble - OFDM (ERP-OFDM) [802.11g Std, 2003 p18]. The long preamble consists of 144 bits of which 128 scrambled, "all ones" bits are used for synchronising the receiver. The remaining 16 bits, Start Frame Delimiter (SFD), are used to indicate the start of frame. The short Preamble consists of 72 bits. The first 56 bits are used to synchronise the receiver and they are scrambled "all zeros". The remaining 16 bits are the SFD but are orientated in reverse to the SFD used in the long preamble. The use of the short preamble is to minimise network overheads and increase data throughput. The coding of the preamble is done with the use of the 1 Mbits Barker Spreading code. For users whose equipment does not use this code, the long preamble is used instead. The ERP-OFDM preamble is the same as the 802.11a

preamble, that is, it uses two frames. The first synch frame is a repetition of ten frames to allow signal detection, Automatic Gain Control (AGC), coarse frequency adjustment and timing synchronisation. The second frame is used to fine frequency adjustment and to help in the channel offset estimation. It comprises of two long training frames with their associated cyclic extension. Due to the different national authorities, there is a huge variation in the transmitter power levels that network providers are allowed to operate at. Some of the more familiar levels are 1000 mW (+30 dBm) in the USA, 100 mW (+20 dBm) in Europe and its associated partners, Australia included, and 10 mW/MHz (+10 dBm) in Japan. This aspect restricts the coverage of the transmitter to small areas. To make them profitable, these units are mainly placed in areas that have high populations such as metropolitan areas and major centres. As with the 802.11a standard, the receiver uses the Carrier Sense Clear Channel Assessment (CS-CCA) to sense any signal above -76 dBm and will enter a 'busy' state till the signal disappears.

4.6 802.16a Wireless MAN

The 802.16 specification refers to Broadband Wireless Access Systems being used in the frequency band from 10 GHz to 66 GHz where Line of Sight (LOS), low multipath and multi-vendor systems can operate. This standard is otherwise known as Wireless MAN. The standard can be used for frequencies below 10 GHz, but employs extra functionality to offset the channel effects. The frequency range below 10 GHz is optional, as that spectrum is not available in all countries. With the spectrum above 10 GHz, some part of that spectrum is available in all countries, therefore allowing manufacturers to provide a product range that can be used in a multitude of countries. This keeps costs down, which the producers find appealing and therefore the standard is accepted for use throughout the globe. The standard is very adaptable and can be utilised in a frequency division duplex (FDD) mode or a Time Division Duplex (TDD) mode. The channel width used in the USA and associated countries are 1.5, 3, 6 and 12 MHz whilst Europe and their associated countries use channel widths of 1.75, 3.5, 7, 8, 14 and 28 MHz. With the differing size of channel widths, there is a range of associated symbol rates. These range from 1.4 to 26.4 MSymbols/second depending on the modulation chosen. The different types of modulation, used on the data, are QPSK, 16-QAM and 64-QAM.

The modulation scheme is chosen to ensure the fastest throughput on each link. The size of the Fast Fourier Transform (FFT) can be adaptable based on the size of the cyclic prefix but the higher the FFT value, the more sensitive the system is to oscillator phase noise. The range of FFT values are from 64 to 2048 (2K mode) sub carriers. A, IEEE recommended, FFT size is 2048 (2K mode) as it is a compromise between multipath, frequency accuracy and pulse shape. Depending on the size of spectrum used, the maximum distance that the system can be used before delay starts to cause unacceptable errors is up to 50 km. This is comparable to the current GSM standard, but a little short of the CDMA standard. The sizes of the cyclic prefix, in relation to the OFDM frame, are $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$ and $\frac{1}{32}$. The different modes the standard can operate in are TDD and FDD. Whilst in TDD mode, the different users are determined by Time Division Multiple Access (TDMA). TDMA divides the sub carrier into different timeslots, which is very similar to how the GSM standard operates. Whilst in FDD mode, the different users are differentiated by Orthogonal Frequency Division Multiple Access (OFDMA). This is a sub-set of OFDM. Each cluster of 64 sub carriers is grouped together to form a sub-channel. As the FFT size increases, the number of sub-channels increases to a maximum of 32, when the FFT size is 2048 (2K mode). This allows 32 simultaneous users. The benefit to the user is that they only need to transmit on their allocated sub-channel, which for the 2K mode, is $\frac{1}{32}$ of the frame period. This is a huge saving in battery use. Also, the user block size stays consistent at 64 sub carriers. The system operates very similar to the 802.11a standard and therefore is backward compatible with that network. On a frame basis, the sub-channels are allocated different sub carriers. The allocation algorithm rotates the sub carriers so that when the number of rotations equals the sub-channel total, it starts again. The allocation algorithm is not used on the 64 FFT mode frame as it only has one sub-channel. The rotation of sub-channels helps offset any radio channel fade that affects the signal. To help keep synchronisation of the signal with the receiver, two modes of sub-channel synchronisation are utilised. In the first mode, some of the sub-channels are allocated as Ranging sub-channels. These sub-channels carry no user data and are used solely for the receiver to synchronise with the received signal. In the second mode, the ranging signals are allocated sub carriers throughout the data sub-channels. This allows for the emission of both data and synchronisation signals simultaneously. As well as the ranging signals, the frames have pilot carriers placed throughout. Some

of these pilots are permanently placed on the same sub carrier whilst other pilots are continuously varied over a cycle of every 4^{th} symbol.

4.7 FLASH

Fast Low Latency Access with Seamless Handoff (FLASH) OFDM uses a fast hopping sequence which spreads the user across multiple sub carriers. This has been aimed at Mobile Data Communications specifically. A user that is normally assigned the same number of and positioned sub carriers is now assigned different positioned sub carriers on every frame. This is very similar to how CDMA operates. The benefits of "spreading" are frequency diversity and interference averaging. Having multiple users assigned to different sub carriers allows each user to not cause any interference to any other user and be resilient to deep fades. This concept is used in TDMA with separate timeslots. The idea of Flash OFDM is to bring the benefits of both CDMA and TDMA together in the same technology, and to be at least three times more efficient than CDMA [OFDM for Mobile Data Communications, 2006 p17]. Coupled with the benefits of these two technologies, Flash OFDM keeps the signalling IP from end user to end user. This reduces the processing time or latency of the system. For systems that need fast throughput, such as video conferencing and live gamers, this idea becomes very attractive. The channel bandwidths for this system are 1.5MHz and 5MHz. This is comparable to the spectrum bands used in CDMA and WCDMA which are candidates for second and third generation technologies. The system uses spectrum below the 3GHz frequency. This caters for the majority of mobile communications.

4.8 Chapter Summary

The many and varied implementations of OFDM modulation show the flexibility it has to meet the requirements of current and future applications. With the increasing number of users wanting various applications, OFDM modulation is at the forefront ready to be utilised in any form. From cable fed ADSL to Flash, the different types of implementations provide different needs to the user although the transmission tech-

nology is the same. This allows the maximum efficiency of the transmission channel regardless of the application.

Chapter 5

Matlab Model

5.1 Chapter Overview

The object of the Matlab simulator is to simulate the operation of an OFDM system. The data, as a bit stream will be configured and sent through the transmitter system. This will involve the bit stream being encoded, interleaved, modulated, pilots inserted, placed through the IFFT function and finally a cyclic extension or guard interval added. The signal emanating from here will replicate the Intermediate Frequency (IF) signal. The IF signals are put through a channel where it is affected by Additive White Gaussian noise. The simulation of the Radio Frequency (RF) stage which involves the IQ Modulator and power amplifier was not simulated. The simulation used Matlab Student Version 7.1 as the simulation software. It is not known if the simulator would operate using previous versions of Matlab. The Simulink block functionality was not used. The code was written using the various toolboxes that Matlab has at its disposal.

5.2 Preparation

Prior to starting the simulation code, Matlab had to be researched to find how it can be utilised to run the program. The function of Matlab that allows simulations, to run with input from the user, is the Graphical User Interface Development Environment

(GUIDE). Guide allows the production of a program that has input functions, such as buttons and pull down menus, allowing the user to input into the program. It also allows the set-up of graphs to show the output of the program. Guide lets the programmer set out the Graphical User Interface (GUI) for the easiest operation. The Gui is linked to a program code file where the software developer can add different code so when the user activates a particular function, that code is operated on. Knowing the different functional blocks in an OFDM transceiver, the Gui was divided into separate sections. Each section represented a component of the transceiver system. The three main sections were Transmitter, Channel and Receiver. Each section was broken down into smaller blocks. The codes for these blocks was written and then added together.

5.3 Initialisation

The simulator is initialised when the file, OFDM_simulator.m, has been run in the Matlab environment. Ensuring that the Matlab current directory holds all the associated files, the simulator should initialise and execute with the user's input. Once the above has run, the GUI is loaded with the default parameters. Using Guide, Matlab's GUI program, the interface was setup automatically. The pull down menus, text boxes, panels and various buttons were loaded. These had to be setup when building the GUI, but they became automatic when they were saved in the *m* file. Various parameters were setup when the simulator's opening function was executed. These various parameters were updated into the GUI structure, handles. The code to execute this function is:

```
guidata(hObject, handles);
```

5.4 Input

For the use of the simulator, there is a text box where the user enters the packet amount in bits and the type of data input. There are three types of data input. They are an all '1' bit stream, a random bit stream of either '0's or '1's and a bit stream from a

video source.

5.4.1 Input Amount

The amount the user would like to enter is entered in the text box. The amount must be positive. There is no upper limit set to the packet size. The value entered is added to the GUI structure, handles, so it is available for all other functions. The code used is:

```
pval = get(hObject,'String');
pakval = str2double(pval);
handles.pakNumVal = pakval;
```

5.4.2 All '1's bit stream

The use of an all '1' bit stream is a common industry standard that is used to test a communication system. This is commonly called a loop back test. The system is set up so the output of the last functional block, prior to the radio frequency stage, is looped back into the input port. If any errors exist, the amount is recorded and the system is broken down to find the source of the errors. The 'Leap frog' rule is used to trace the error source. When using the Leap frog rule, the loop back is placed on the output of the first block. The data is checked. This step ensures the quality of data entering the system. As each block is checked for conformity, the loop back is moved towards the final stage. Various stages may be 'leap frogged' until the faulty block is found. This test is an invasive test and therefore the system is inoperable whilst in the test mode. For an invasive test, the output result could be analysed. The expected result is all '1's and any errors and be picked up very easily. The production of the bit stream in the simulator occurs using the code:

```
dataorig = ones(s,1);
```

When the Bit Error Rate (BER) is zero, the test is complete and the system is working as it should. The random bit stream test is employed to further test the system.

5.4.3 Random bit stream

A random bit stream comprising of either '0's or '1's, is used as the input to further test the system's BER response. The random bit stream assesses the system more thoroughly than the previous all '1's examination. The receiver decision algorithm must decide whether the data is either a '0' or a '1'. This analysis highlights any system deficiencies that the all '1's exam does not pick up. The test is an invasive assessment as well. The random bit stream is created using the code:

```
dataorig = randsrc(s,1,[0 1]);
```

At the end of the assessment, the result is compared to the input and the errors are read. The leap frog rule is used till the faulty block is found and rectified. The test cannot be conducted without knowledge of the input bit stream. As this is random, this needs to be found prior to every test. This may be difficult if different sections exist at different locations. For wireless communication systems, this is usually the case. At the completion of the test with no errors, the system is deemed to have no detrimental impact on the data. A final assessment of sending through data of a defined structure or picture is the last system test.

5.4.4 Video

This is the final input data type that the system is assessed on. It comprises of a movie file, clock.avi. The code used to call the movie file into the simulator is:

```
mov = aviread('clock.avi');
fileinfo = aviinfo('clock.avi');
movfra = fileinfo.NumFrames;
```

The file is converted into a bit stream prior to transmission through the simulator. The file comprises of a cdata matrix and an associated colour map. The cdata matrix is a matrix of integers that represents each pixel in the picture. The matrix element is an 8 bit integer. The integer indicates the colour of the pixel. The total choice of colours for an 8 bit integer is 256 colours. The colour map comprises of 4 columns. The first column

refers to the colour integer, 1 through to 256. The following 3 columns indicate the amount of red, green or blue (RGB) levels in the pixel colour. The matrix and colour maps are converted to a binary bit representation and input into the transceiver. The code used to convert the matrix and colour map is:

```
[xmov,movMap] = frame2im(mov(imov));
xmovb = de2bi(xmovab,8);
movMapaa = movMapa * 255;
movMapb = de2bi(movMapaa,8);
movbin = [xmovc movMapc];
```

Once the bit stream has been through the simulator, the system BER is measured. The resultant bit stream is converted back to a movie file, rxclock.avi. The movie file can be played by any media player to show the effect the system has on the quality of the picture. The movie comprises of 12 frames. Each frame is shown once per second. The picture is a clock face where a hand moves from the number 1 to the number 12 in a circular motion. The file shows the effect of noise as the SNR is decreased for each frame. Through comparison with the line graph output, a subjective relationship can be made with the resultant movie.

5.5 Encoding

The output of the Input Select block becomes the input stream to the OFDM transmitter. The first block in the transmitter is the Forward Error Correction (FEC) Encoder. The encoder outputs a number of output bits for each input bit. The number of output bits is dependent on the code rate chosen. There are three different code rates used. They are $\frac{1}{2}$, $\frac{2}{3}$ and $\frac{3}{4}$. The encoder used in the simulator was a convolutional encoder that is the model for the WLAN standard. The polynomials used are 7 as the constraint, 133 and 171 as the generator matrix feedback entries. The constraint is the number of bits in the shift register and the matrix entries indicate the bits that are used to add together to get the output bit. The outputs of the generator have a bit delay due to the shift register. These outputs are represented by:

$$g_1(D) = D + D^3 + D^4 + D^6 + D^7 \quad (5.1)$$

$$g_2(D) = D + D^2 + D^3 + D^4 + D^7 \quad (5.2)$$

Where:

D represents the unit time shift delay of the shift register

The generator as shown:

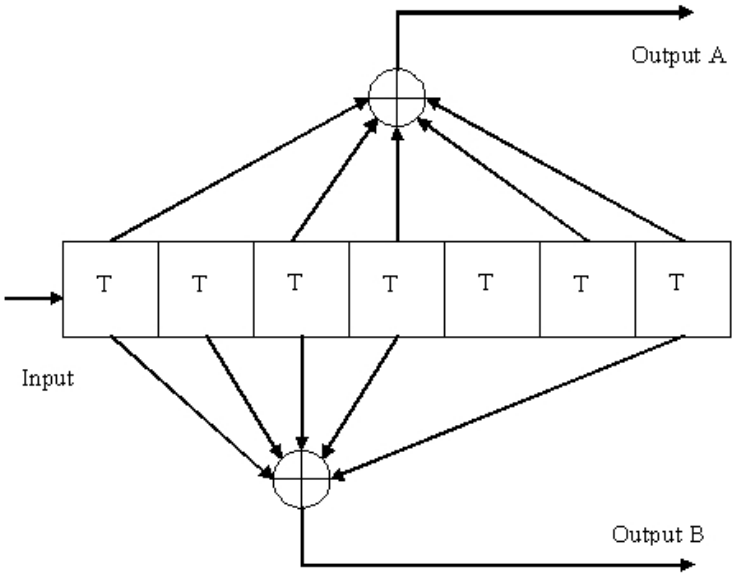

Figure 5.1: Simulator convolutional encoder

The output bit A shall precede the output bit B. The use of bit puncturing is employed to gain higher code rates.

5.5.1 Half rate Code Rate

The half rate outputs two bits for every input bit. This is the standard output from the encoder. The output bits are determined by the input bit plus the contents of the

shift register. These bits may be the starting state of the shift register or the previous seven bits of the data stream. The code used to encode the bit stream with the code rate is:

```
t = poly2trellis(7,[133 171]);
tcode = convenc(codeIn,t);
```

5.5.2 Two Thirds Code Rate

The two thirds code rate uses the half rate as previous but, omits every fourth bit. The pattern is two input bits long making four output bits. When the bit has been omitted, the output has three output bits for every two input bits.

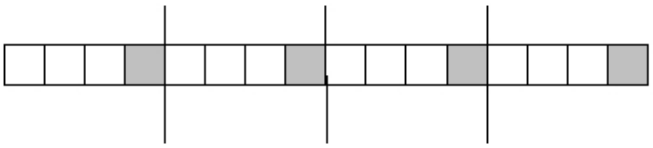

Figure 5.2: Two-thirds code rate output

The code used to generate the two thirds code rate is:

```
t = poly2trellis(7,[133 171]);
tcode = convenc(codeIn,t);
punct23code = tcode;
punct23code(4:4:end)=[];
```

5.5.3 Three Quarters Code Rate

As with the previous code rate, the three-quarters code rate is the half rate code with some bits punctured. The puncturing pattern is three input bits long. This leaves six output bits prior to puncturing. The code omits every fourth and fifth bit so after

three bits have entered the function, four output bits are produced. This represents four output bits for every three input bits.

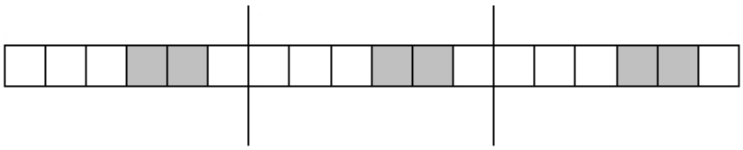

Figure 5.3: Three-quarters code rate output

The code that produces the three-quarters code rate was:

```
t = poly2trellis(7,[133 171]);
tcode = convenc(codeIn,t);
punct34code = tcode;
punct34code(4:6:end)=[];
punct34code(4:5:end)=[];
```

5.6 Interleaving

The data stream from the encoder block is put into the interleaving function. The function of this block is to spread the data bits over the sub carrier channels. This is to offset any deep fades that occur in the wireless channel. The wireless channel is a wideband channel and rarely encounters a flat, consistent response across the entire spectrum. As deep fades affect more than one sub carrier channel, a block of data is affected. By spreading adjacent bits across the channels, the bits affected by fades will be isolated, once the bits have been re-arranged in their proper order. The effect of the fade is reduced. The interleaver function used in the simulation is a random process, randperm. The bits are placed in a block equal to the number of sub carriers multiplied by the number of bits per symbol. The number of bits per symbol is dependent on the modulation type used to transfer the data. The types of modulation are BPSK, QPSK, 16-QAM and 64-QAM. If the WLAN standard is being simulated, the number of sub-carriers is forty eight and therefore the block lengths are 48, 96, 192 and 288 bits respectively. The length of the block is referred to as the interleaving depth. This

parameter defines the delay the interleaver introduces to the system. As the depth gets larger for the different modulation schemes employed, the delay gets larger accordingly. The length of the data stream is padded out so it becomes a multiple of the block length. This ensures there are complete blocks even at the end of the data stream. The integers in the permutation matrix range from 1 to the block length. These integers are randomly rearranged. The bits in the data block are rearranged according to the permutation matrix. The permutation matrix is carried through to the receiver so the de-interleaving function can occur. The receiver requires the matrix as it is different for each data transfer. The code used to generate the interleaved data using a random block permutation is:-

```
q = randperm((NoSub*NoBits)).';
newintrlvd = intrlv(levMess,q);
```

5.7 Modulation

Once the data has been through the interleaving process, it is placed into the Modulation function. The modulation function is to convert the binary bit stream into complex symbols so that these information symbols can be transmitted through a channel. The complex symbols are RF expressions for the data stream and are generally expressed as:

$$s(t) \quad = \quad A(t)\, e^{(w_c t + \theta(t))} \tag{5.3}$$

Where:

A is the signal amplitude,

w_c is the carrier frequency in radians and

$\theta(t)$ is the phase.

The radian carrier frequency is calculated from the equation:

$$w_c = 2\,\pi f_c \tag{5.4}$$

Where:

f_c is the carrier frequency in hertz.

The types of modulation used in the simulator are BPSK, QPSK, 16-QAM and 64-QAM. Added to these types, is a choice of no modulation which was used as a control to compare all the different types for their impact. The choice of no modulation means that the data out is a replica of the data in. No processing of the data occurs with this choice.

5.7.1 BPSK Modulation

BPSK modulation involves placing 1 bit of the data stream onto a RF carrier. The data stream is checked to ensure there are enough bits to make complete symbols. When the end of the data stream is reached, there may not be enough bits to make a symbol, so extra bits are padded onto the length. With BPSK modulation, the number of symbols is the same as the number of bits. This is due to the 1 bit per symbol parameter of BPSK. The code used to check the symbol number is:

```
M=2;
k = log2(M);
input_sym_len = inModLengthOrig/k;
sym_len_rnd  = ceil(input_sym_len);
inputLength = sym_len_rnd * k;
msgExtBitsLen = inputLength - inModLengthOrig;
msgExtBits = ones(1,msgExtBitsLen);
newMess = [mod_input msgExtBits];
newMessLen = length(newMess);
newMess = newMess';
```

Although the code is not used as there is a direct relationship, it was still added to the function in case future use called for data changes. The data is converted from binary bits to a decimal representation. The data stream is not changed in any way, as the decimal range for BPSK is from 0 to 1. It is still added for program conformity. The code for the conversion is:

```
msg_sym = bi2de(msg_sym_reshape.', 'left-msb');
```

Once the data has being changed to decimal notation, it is modulated using the code:

```
msg_tx = pskmod(msg_sym,M);
```

A scatter plot produced from Matlab shows the correct constellation map for BPSK modulation.

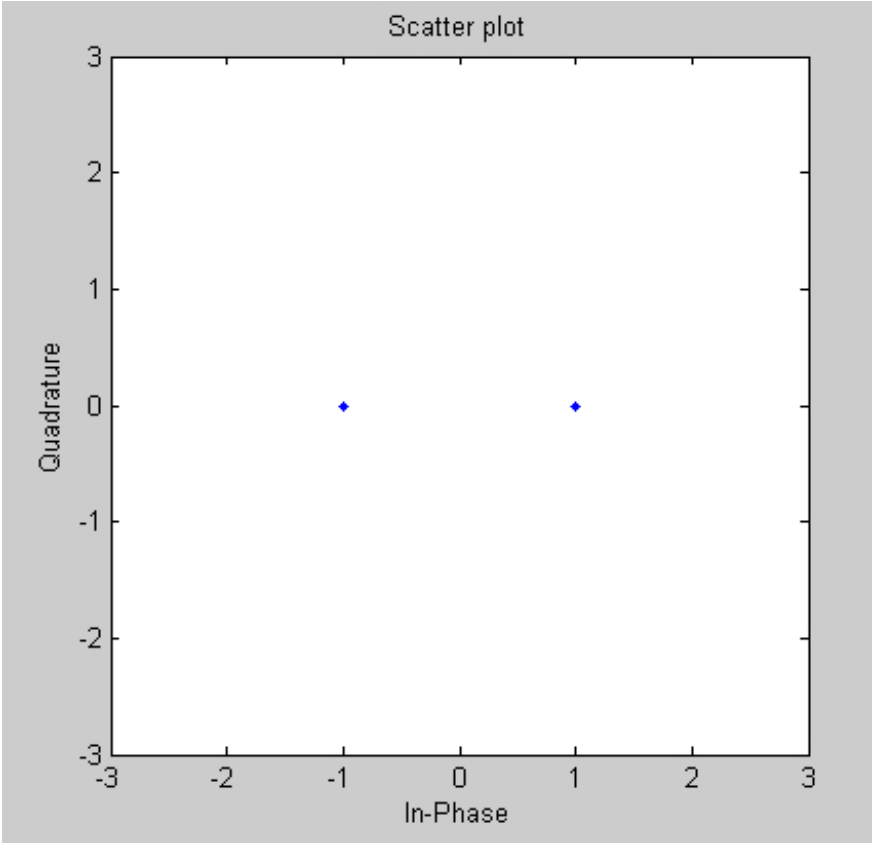

Figure 5.4: BPSK simulator scatter plot

5.7.2 QPSK Modulation

QPSK modulation involves placing two bits onto a RF carrier. The same process of checking is carried out to ensure there are enough bits to make complete symbols. For QPSK, the number of bits has to be an even number. If the original data length is an odd number, the data stream is padded with an extra '1'. It is converted to a decimal number that ranges from 0 to 3. The binary range is 00 to 11. The code to convert and modulate is the same. The difference is the modulation parameter M where for BPSK, the value is 2 and for QPSK, it is 4. A scatter plot is produced to ensure the code matches the operation. The scatter plot is:

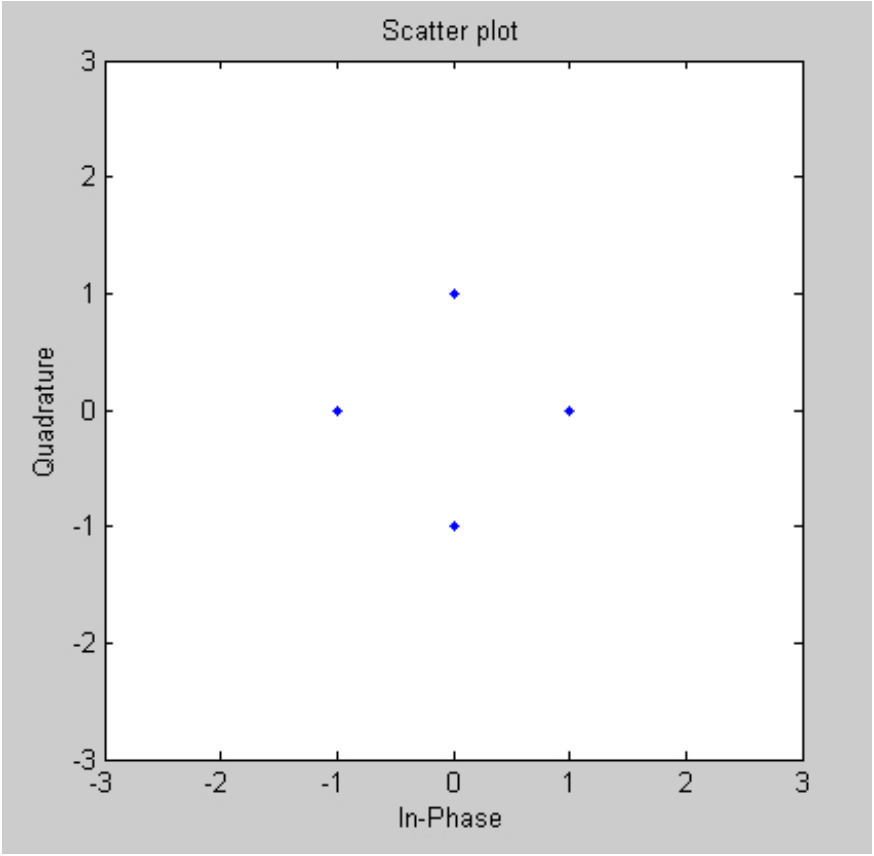

Figure 5.5: QPSK simulator scatter plot

5.7.3 16-QAM Modulation

16-QAM modulation involves the placing of 4 bits on a RF carrier. The process of checking the data length to ensure it is a multiple of 4 and padding it with '1's till it is, is very involved. It is converted to decimal whose range is now from 0 to 15 (0000 to 1111). As the modulation scheme is not a PSK scheme, the code to modulate is:

```
msg_tx = qammod(msg_sym,M);
```

The modulation parameter M is 16 and the associated scatter plot is:

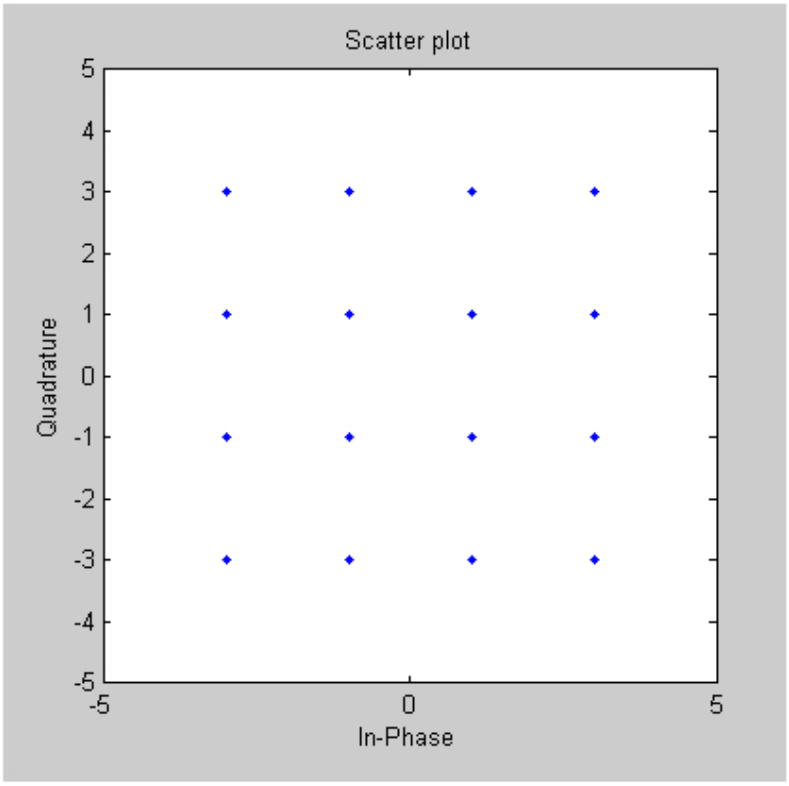

Figure 5.6: 16-QAM simulator scatter plot

5.7.4 64-QAM Modulation

64-QAM modulation enables 6 bits of data to be placed on each RF carrier. The process of ensuring the data length is a multiple of six is used prior to decimalisation and modulation. The range of decimal values for 64-QAM is from 0 to 63 (000000 to

111111). The scatter plot is:

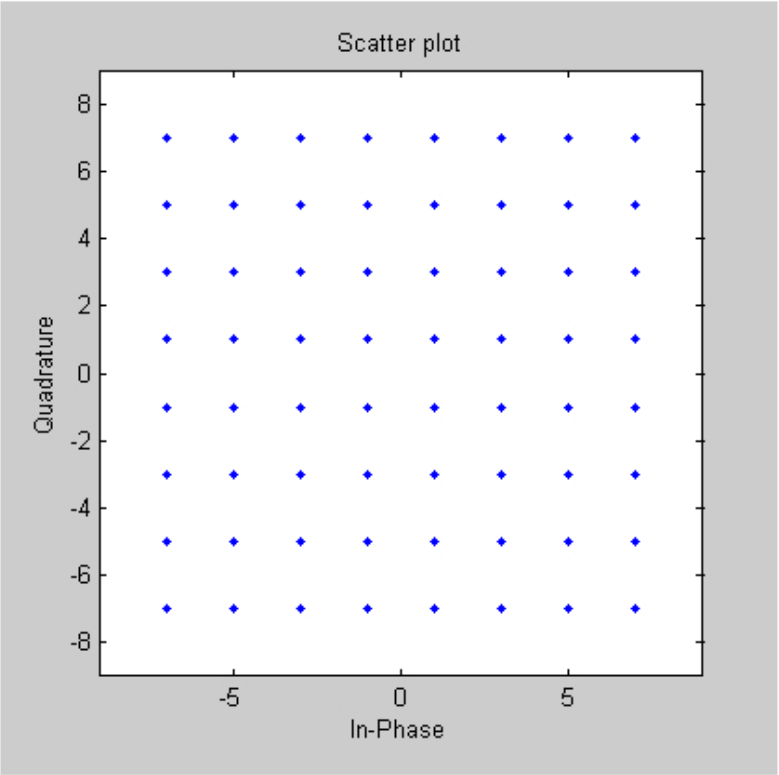

Figure 5.7: 64-QAM simulator scatter plot

5.8 IFFT, Pilot Insertion and Cyclic Extension

To ensure the correct data is extracted from the received signal, pilot sub-carriers are placed throughout the OFDM symbol. The complex symbols, from the modulation function block, are measured to ensure its length is a multiple of the data sub-carriers. If the length is not, the symbol stream is padded with '1's. The length is made a multiple of the data sub-carriers so that the output of the IFFT function block is a number of complete OFDM symbols. The code that ensures the length is correct is:

```
sym_len_col = (length(ifft_input)/noDatCar);
sym_len_rnd  = ceil(sym_len_col);
inpifftLen = sym_len_rnd * noDatCar;
ifftExtBitsLen = inpifftLen - ifftLenOrig;
```

```
ifftExtBits = ones(1,ifftExtBitsLen);
ifft_input = ifft_input';
newifftMess = [ifft_input ifftExtBits];
newifftLen = length(newifftMess);
newifftMess = newifftMess';
```

Once the symbol stream length is correct, the symbols are assigned to the sub-carrier index. The code for the 64-IFFT is:

```
ifftSymPil([7:11,13:25,27:32,34:39,41:53,55:59],:) =
    ifftSym([1:5,6:18,19:24,25:30,31:43,44:48],:);
```

The various pilot tones, zero pads and DC pad are assigned their relevant sub-carrier index. The code for the 64-IFFT is:

```
ifftSymPil([1:6],:)    =  0;
ifftSymPil([12],:)     =  1;
ifftSymPil([26],:)     =  1;
ifftSymPil([33],:)     =  0;
ifftSymPil([40],:)     =  1;
ifftSymPil([54],:)     = -1;
ifftSymPil([60:64],:)  =  0;
```

The OFDM data symbols have now been assigned the proper data symbol to their respective sub-carrier index and they are transformed using the IFFT function. The code is:

```
ifftSymOut = ifft(ifftSymPil);
```

The cyclic extension is added to the OFDM data symbol to make the complete OFDM symbol. The code that incorporates the cyclic extension for the 64-IFFT is:

```
ifftSymPrefix([1:16,17:80],:) = ifftSymOut([49:64,1:64],:);
```

The OFDM symbols are serialised and waiting for the preamble to be added before it is transmitted. The code for the other IFFT is exactly the same except the assignment

of the sub-carrier index is larger to incorporate the bigger IFFT sizes. All the program code is located in Appendix B.

5.9 Preamble

The preamble is used to allow the receiver and the received symbol stream to synchronise to each other prior to the data. This allows the receiver to gain some knowledge of the impact the channel has had on the transmitted signal. The receiver has a set of known OFDM symbols that it compares the received preamble with. Through the comparison process, the phase and amplitude changes can be estimated and offset. The simulator's preset preamble is aligned with the 802.11 preamble and is the same for all the different scenarios. This consists of 10 short training symbols followed by 2 long training symbols. The code to produce the preamble is:

```
p16([1:16],:) =  pshortifft([1:16],:);
peramsh = [p16 p16 p16 p16 p16 p16 p16 p16 p16 p16];
plongifftPrefix([1:16,17:80],:) = plongifft([49:64,1:64],:);
peramblelo([1:16,17:32,33:96,97:160],:) =
    plongifft([33:48, ... 49:64,1:64,1:64],:);
preamblefull = [peramblesh peramblelo];
```

The preamble is added to the OFDM symbols at the start of the OFDM transmission. The code is:

```
frameout = [preamble ifftoutput];
```

The parameter frameout is the complete OFDM transmission and it is sent into the wireless channel to the receiver.

5.10 RF/IQ Modulation

The RF / IQ Modulation stage of the OFDM simulator was studied and tested. The functional diagram for this stage was:

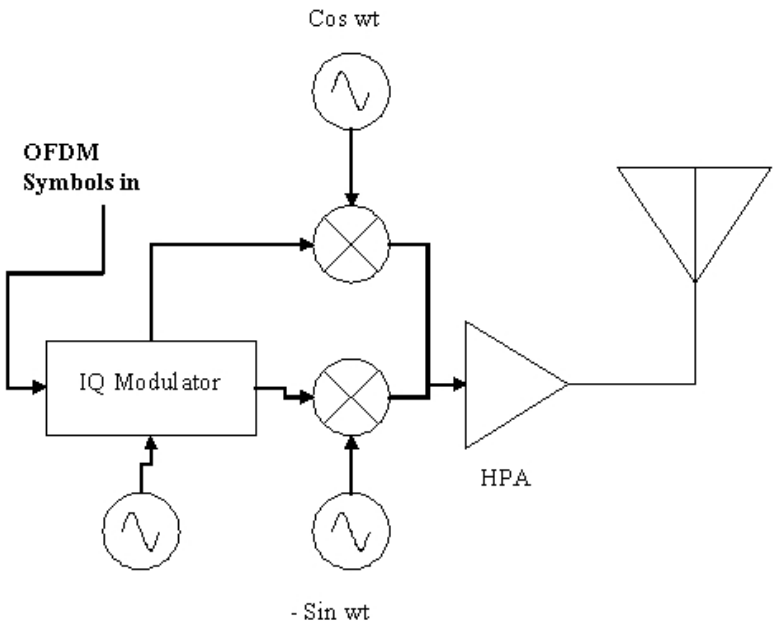

Figure 5.8: Simulator RF stage diagram

The simulator code did not work as the receiver received every OFDM symbol as an error. Whilst simulators generally only test to the IF or baseband signal, the application of the RF stage would highlight the technology at different frequencies. A more in depth study of radio simulation practises is needed and it is beyond my current knowledge base.

5.11 Channel

The channel was set up to create noise and to induce fading. The transmitted signal would be affected by a Rayleigh or Rician fading channel and white Gaussian noise would be added. The user can choose which fading channel type they prefer but the white Gaussian noise is always utilised. Noise is considered universal and to not have noise included in the simulation is not a true indication of the systems behaviour. The capability to alter the value of the signal Eb/No is to test the signal under varying conditions. A graph could be produced to highlight these effects. The channel code is:

```
fadedSig = filter(chan, channin);
```

```
channout = awgn(fadedSig,snr);
```

5.12 Receiver

The receiver is to detect the start of the transmission, synchronise and offset any changes made to the signal and remove any information from it. It is basically the reverse of the transmitter with extra timing and frequency functions. The preamble is removed and is changed from the complex Cartesian values to complex polar values. The known preamble is compared with the received preamble and the difference is the channel impact on the signal. These values become the channel estimate to adjust the received signal values with so as to offset the channel impact. The code is:

```
rxIpream = real(rxIQpre);
rxQpream = imag(rxIQpre);
[rxTHETA,rxRHO] = cart2pol(rxIpream,rxQpream);
preamresrho = txRHO ./ rxRHO;
preamresthet = txTHETA - rxTHETA;
```

5.13 Output

The output of the simulator is in a matrix as well as a graph. The test results always compare the simulation parameters to see what the BER will be with a set value for SNR. The BER values are recorded as the SNR is varied. This shows the response the simulator has in varying conditions. The test results can be chosen depending on the user's choice. The choices are single test, code rate comparison, modulation type comparison and video test results. The single rate test tests the simulator as set by the user. It outputs a graph to highlight the system's response. It does not compare to a baseline measure. The code rate comparison compares the different code rates used in the simulator with each other and a baseline, non-encoded signal. It highlights how well the system setup compares using different rates. The modulation type comparison compares the different types of modulation schemes with each other and as well as a

base-line non-modulated signal. The video test gives a real situation demonstration of what the graphical figures would look like if used. A graph showing the system response is given and the user can make a subjective decision on how well the system responds to the given setup.

5.14 Chapter Summary

The implementation of the OFDM simulator in Matlab was very involved and complex. By breaking the system down into functional blocks, the simulation could be achieved. Being familiar with the code and the many different commands make the task of coding a lot easier and this would come with experience. Whilst the system was being constructed, the output of the functional blocks was being compared with the theoretical results, to highlight the quality of the code written. This allowed for the detection of errors before the simulations were started. The RF stage was found to not be functioning correctly and was removed. A deeper knowledge of Matlab and its code commands would solve this issue.

Chapter 6

Results and Discussions

6.1 Results

6.1.1 Simulation GUI Screen

Once the simulator has been started, the user will be prompted to input the system parameters for their OFDM transmission. The GUI screen is:

Figure 6.1: Simulator GUI screen

6.1.2 Typical test result screens

The simulator can give the results of a system simulation in a number of screens. The choice is decided by the user. The choices of screen results are a single test result, a code rate comparison test result, a modulation type comparison test result and a sub-carrier size comparison test result. The user can choose the size of data that they would like to send in bits, the type of data, either a random bit stream or an all '1's bit stream. They which type of encoding, either none, $\frac{1}{2}$, $\frac{2}{3}$ or $\frac{3}{4}$. The modulation type is chosen as either none, BPSK, QPSK, 16-QAM or 64-QAM. The size of the sub-carrier is chosen as either 64, 256, 512, 1024, 2048, 4096 or 8192. The screens come with an associated matrix of the values of the system simulation BER. A single test screen shot is:

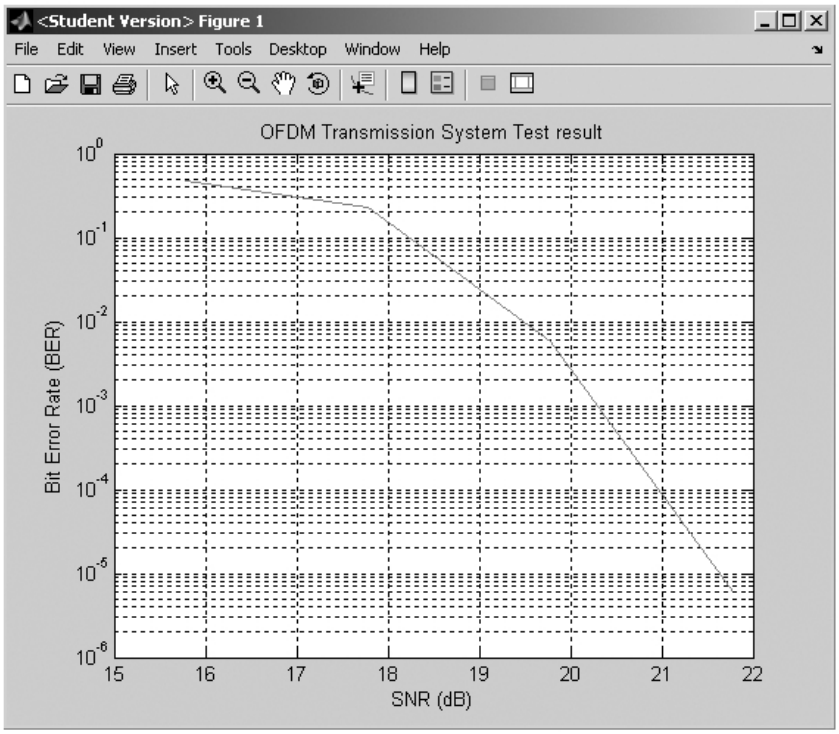

Figure 6.2: Typical single test output

A typical code rate comparison screen shot is:

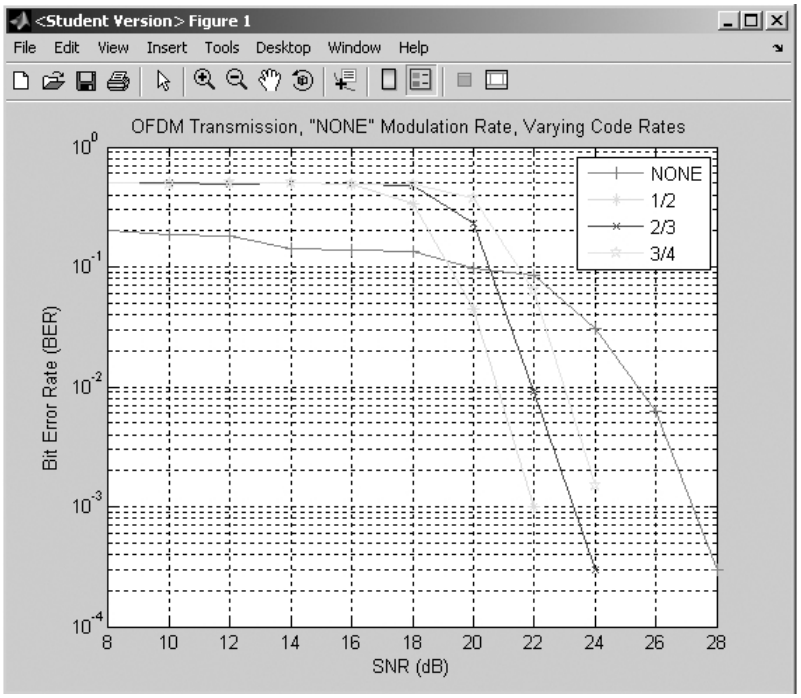

Figure 6.3: Typical code rate comparison test output

A typical modulation scheme comparison screen shot is:

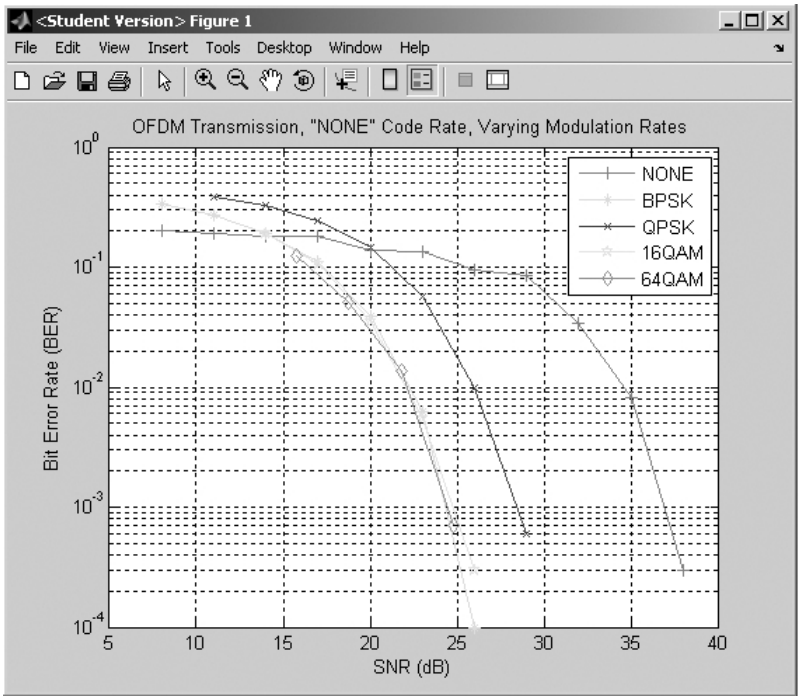

Figure 6.4: Typical modulation rate comparison test output

A typical sub-carrier size comparison screen shot is:

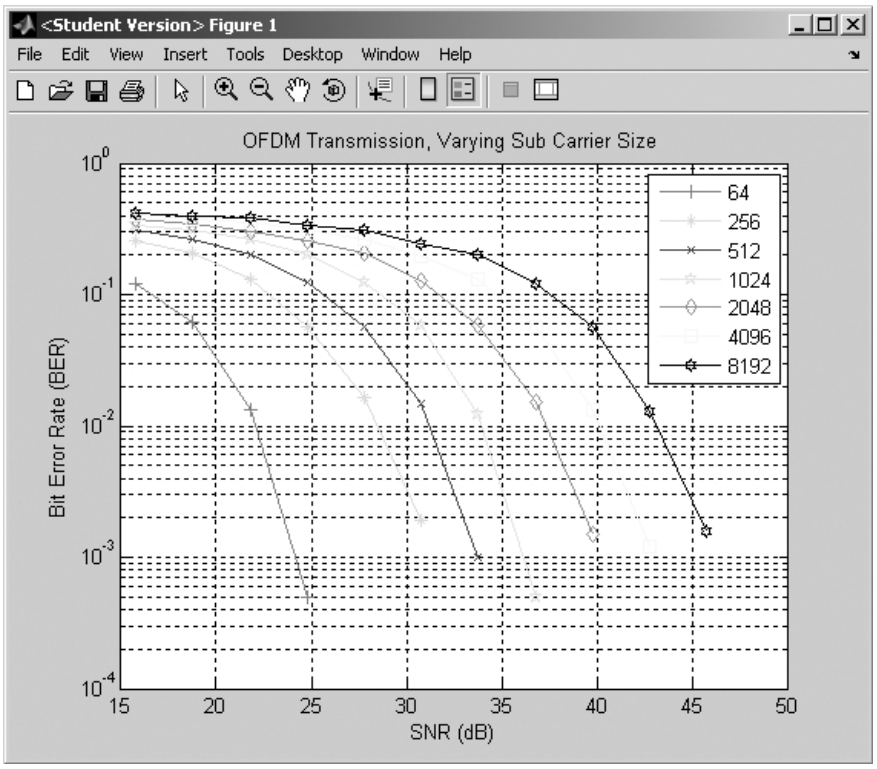

Figure 6.5: Typical sub-carrier size comparison test output

6.1.3 Single system test result

For a given scenario, the BER of the system is found with a varying level of SNR. The values are placed in a graph for easy interpretation of the figures. The performance of any type of transmission can be found. If the user needs to know how well a system would work at a particular BER, this graph will highlight the performance.From the graphical result below, it can clearly be seen that as the power of a signal increases, the error rate decreases. There is a power level where the error rate flattens out and becomes constant. This seems to occur at approximately 15 dB. The results are left in the matrix, bitout. A graphical view of the System Test result is:

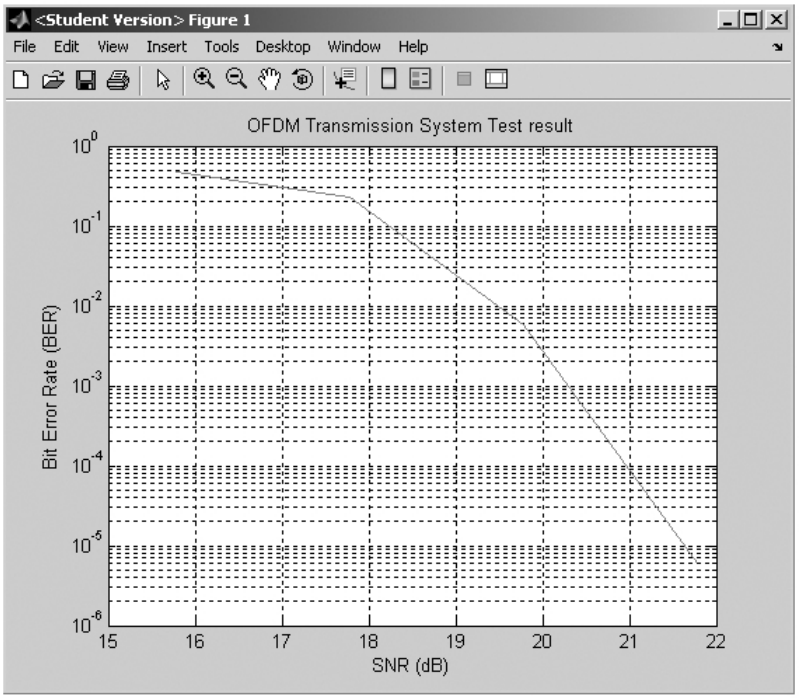

Figure 6.6: Single test output

6.1.4 Code rate comparison test result

The code rate test compares the settings the user has defined for their OFDM transmission with the different types of code rates to find the optimum code rate. The same channel and data parameters, that the user chose, are used except the code rate is varied for all types. From the graphical results below, a none coded signal has a BER of 0.01 at a level of approximately 41 dB. A similar signal that is coded with the three quarter rate has a BER of 0.01 at approximately 36 dB. A maximum difference of 5 dB was found as all the other code rates fell between these two extremes. At about 25 dB the BER level levels out for all types of code rates. In noisy areas where the E_b/N_o of the signal is below this level, the use of a code rate adds no extra benefit. A test result for varying code rates is:

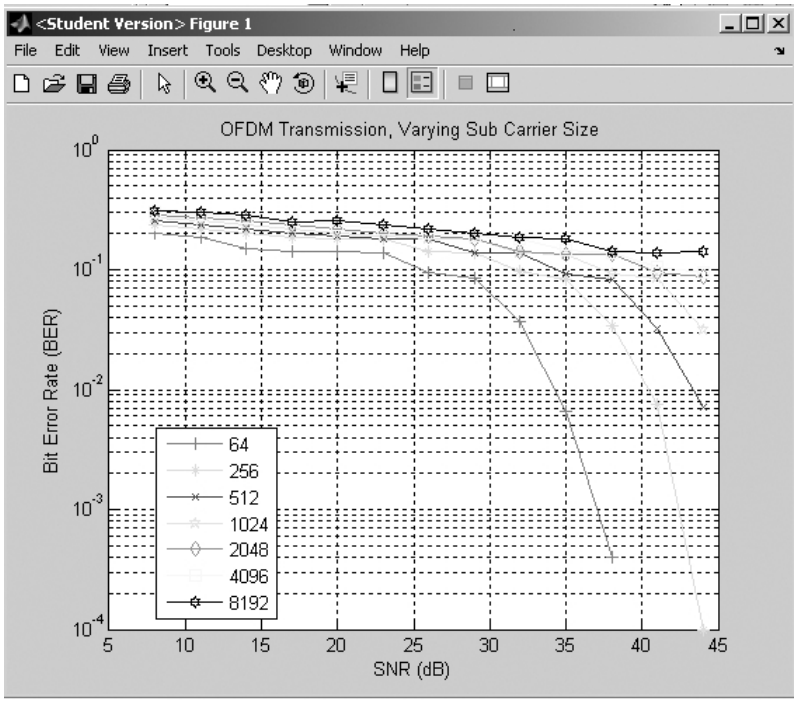

Figure 6.7: No code test output

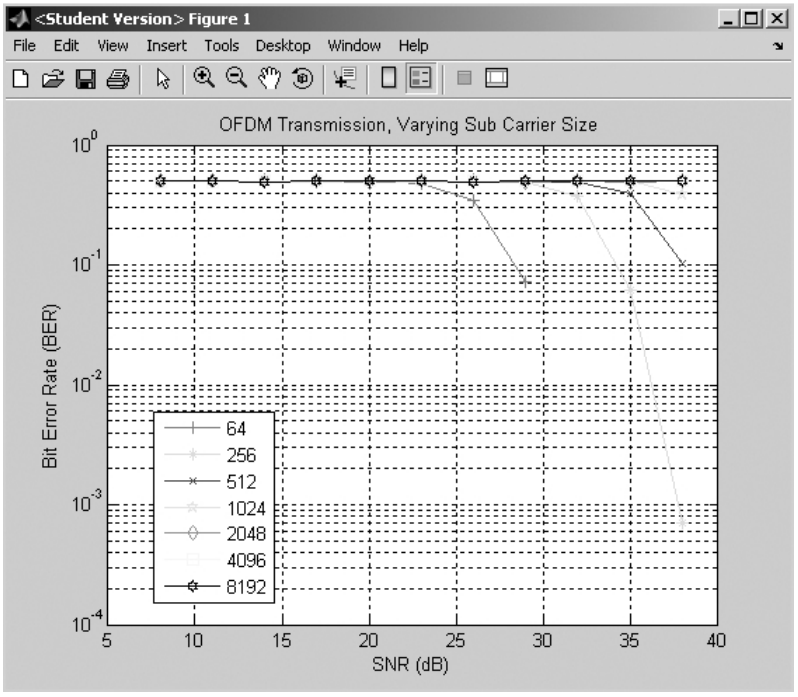

Figure 6.8: Three quarter code test output

6.1.5 Modulation type comparison test result

A modulation comparison screen shot is:

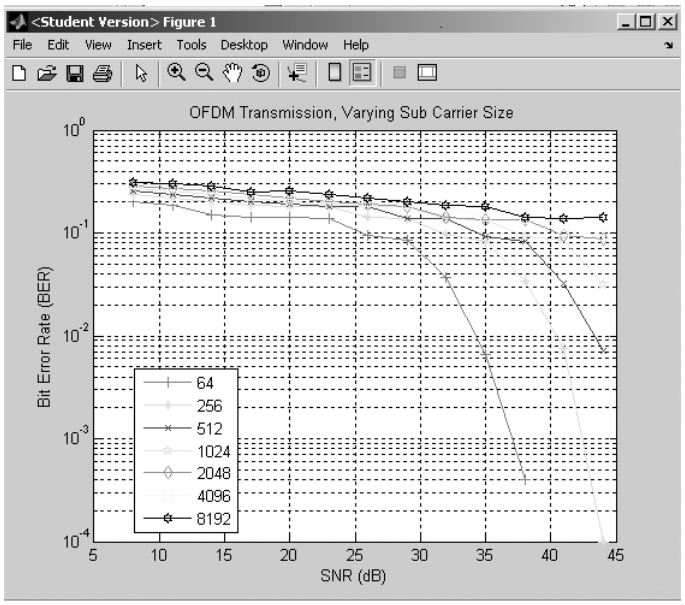

Figure 6.9: No modulation test output

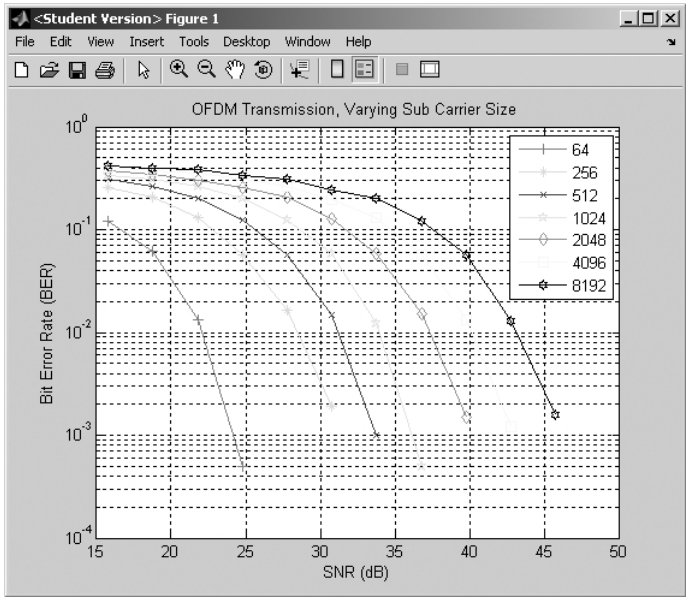

Figure 6.10: 64-QAM modulation test output

The modulation type comparison test takes the user defined parameters for their OFDM

transmission and varies the modulation type so that the optimum modulation scheme can be found. The result is a graph that compares the different schemes against each other and a base-line non-modulated signal. From the results above, a non-modulated signal has a result that is approximately 12 dB than a signal that is modulated. This is constant regardless of the different carrier size. This shows the use of a modulation scheme is necessary for data transmission. The values are stored in a matrix, bitout.

6.1.6 Sub-carrier size comparison test result

The sub-carrier size comparison test compares the different sizes of sub-carriers that are available whilst using the user defined system parameters. The different sizes are 64, 256, 1024, 2048, 4096 and 8192 sub-carriers. The result is a graph that can be interpreted very easily to find the most optimum sub-carrier size. From the graphs below, the 64 sub-carrier transmission has the best result at 22 dB. For the transmission with 8192 sub-carriers, the best result is 44 dB for the same modulation scheme. The 64 sub-carrier system has a better response by approximately 22 dB regardless of the modulation scheme used. The values are, also, stored in a matrix, bitout. A typical sub-carrier size comparison screen shot is:

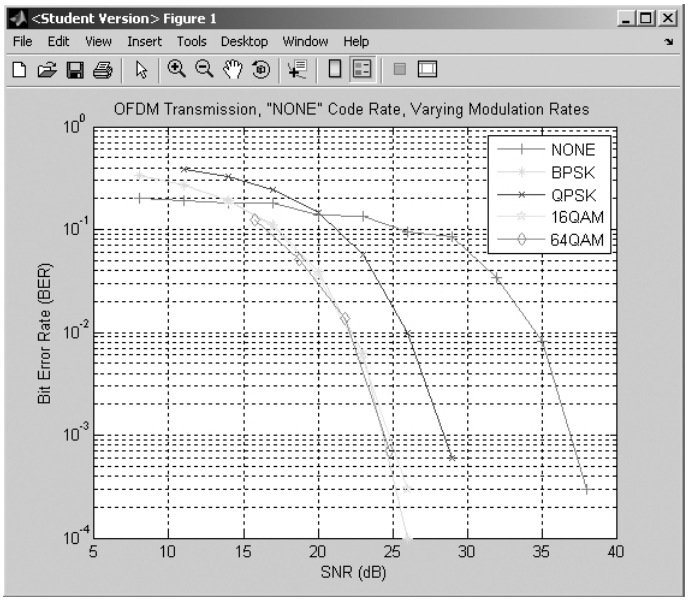

Figure 6.11: Sub-carrier size 64 test output

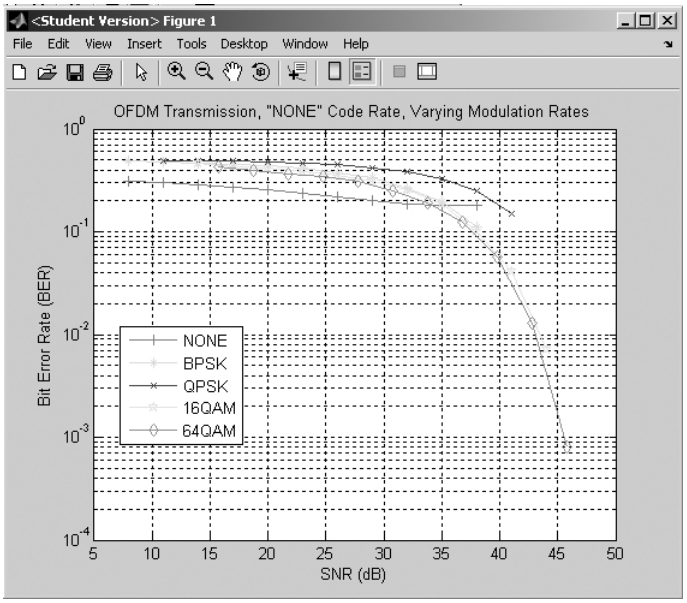

Figure 6.12: Sub-carrier size 8192 test output

6.1.7 Video test result

The video test result is a movie file output based on a movie as its input.

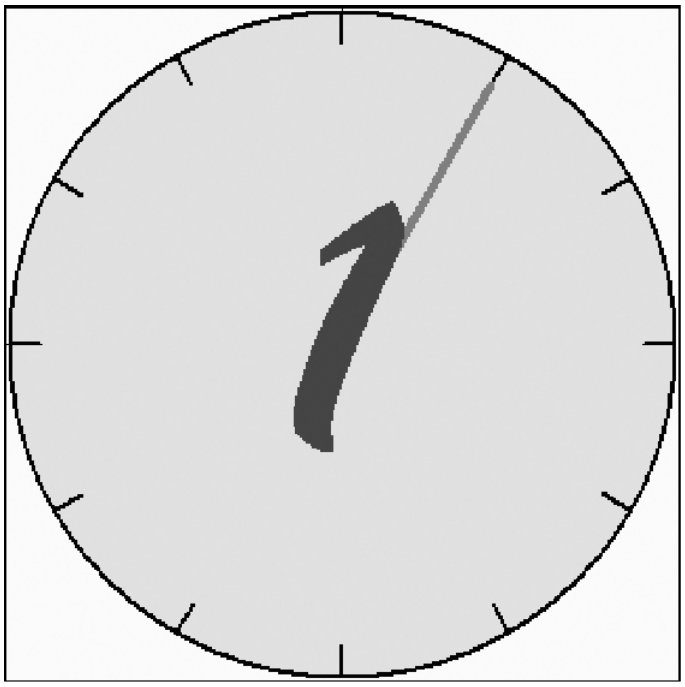

Figure 6.13: Video test input, clock.avi

The output movie would highlight the differences between its picture and colour quality and the input movie's picture and quality. The output movie, rxclock.avi, is written to the working directory of the simulator. A media program, such as Windows Media Player, is needed to play the movie to see the differences. The test movie, clock.avi, is the face of a clock with a rotating hand. The movie has 12 frames and is played at 1 frame per second. As the frames are played, the hand rotates around stopping at each number on the face. The test is set so that as the frames are played, the SNR was reduced to highlight the impact that a noisy environment would have on a movie or television broadcast.

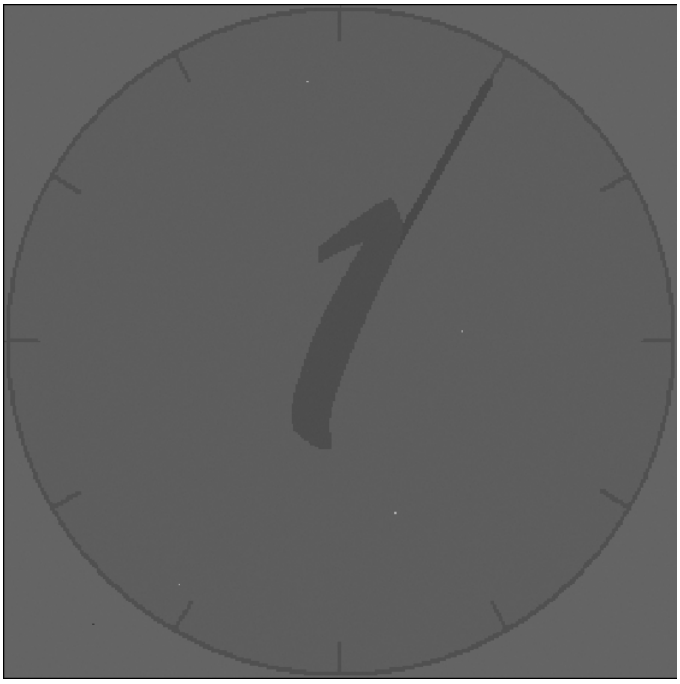

Figure 6.14: Video test output with E_b/N_o @ 35 dB, rxclocka.avi

At frame 1, the E_b/N_o was set at 35 dB. This should produce a quality picture. The frames were sent through the system with a reduction of the E_b/N_o at 3 dB per frame. This left the range from 2 dB to 35 dB. At approximately 30 dB, the picture started to show grainy spots. At 17 dB, the picture was non-existent due to the noise. A matrix of the BER, bitout, corresponds to the errors each frame has. From a user's subjective viewpoint, the BER amount can be found for a movie broadcast. A suitable amount of errors can be found before it impacts on the movie quality.

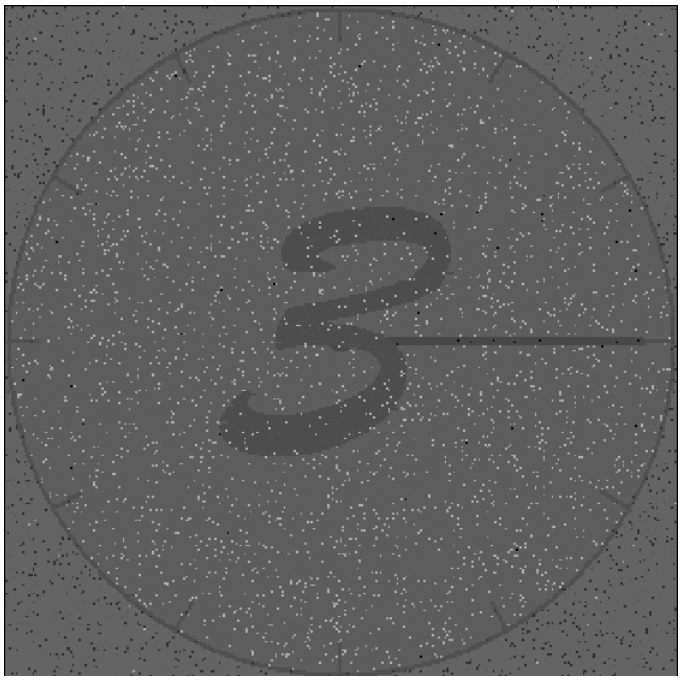

Figure 6.15: Video test output with E_b/N_o @ 29 dB, rxclocka.avi

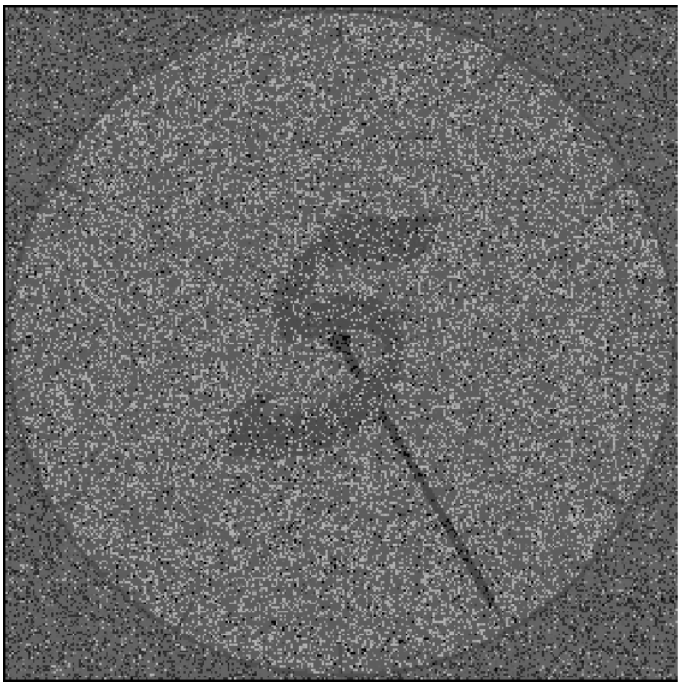

Figure 6.16: Video test output with E_b/N_o @ 23 dB, rxclocka.avi

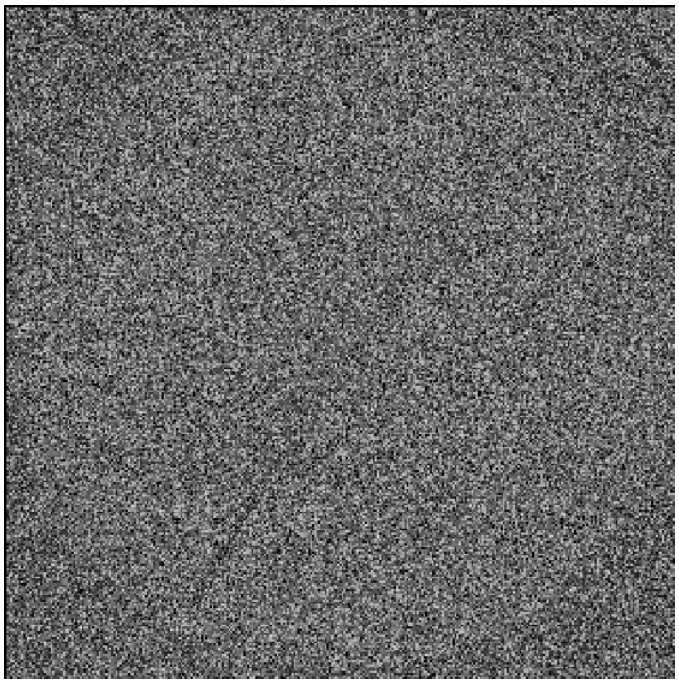

Figure 6.17: Video test output with E_b/N_o @ 17 dB, rxclocka.avi

6.2 Discussion

6.2.1 Research

The research for this project has shown the possibilities of this technology in the coming future. There are already many different types of applications in use clearly showing the flexibility the modulation technique has to offer. As technology progresses, the advent of this technique to ensure continual connectivity is becoming more of a realistic goal. Within the last century, many communication techniques have been introduced that have made it possible to consider the Earth a much smaller place than ever. People have been able to contact other people without a large effort of getting together. All of these techniques relied on serial transmission. With OFDM, this technique of sending information in a parallel data transfer has broken the mould of the previous century. Given the possibilities of OFDM transmission, many groups have researched this approach and have developed many new standards for commercial use. The time between the development of these standards to the wide spread use of the technology

has been very short. Such is the case of the 802.11g standard known as "WIFI". It was ratified in late 20003 and many businesses were upgrading their 802.11b equipment less than 2 years later. The demand for faster and faster connections is pushing the technology ahead as can be seen with the popularity of digital televisions. The project has given me a thorough insight into the possibilities of the future and the communication approaches that may be used to achieve them. I have learnt that a thorough understanding of the critical aspects of an idea can lead to possible new methods to improve the current environment.

6.2.2 Guide Vs Simulink

For the simulator, Guide, Matlab's GUI program was used. Another alternative to Guide is Simulink. This is supported by the Matlab people. Simulink is purposely produced to simulate, model and analyse systems. Without a complete understanding of Simulink, it looks a lot easier than writing each functional block in Guide and its associated m files. The blocks are preset and only the parameters need to be filled prior to the operation of the simulation. This would save a lot of time coding and debugging.

6.2.3 Goal Achievement

The use of a simulator that mirrors theory is the ultimate result of any simulation. It proves that the research, application of the techniques used and the investigation of the environment that impacts on the technology have been understood. Through the research into OFDM transmission, a wide variety of applications have been found to operate. This allows for a broad use for the simulator to imitate. As with all wireless communications, the channel will impact on the signal being transmitted through it. To offset these changes, is to show that a thorough understanding of a radio channel was achieved. The design of the simulator can be verified with the knowledge of theoretical outputs of each functional block. The output of each block was checked to ensure the theory matched prior to advancing to the next block. This practise ensured a quality simulator could be achieved. Being able to alter the systems parameters to suit many different environments allows the flexibility to find any short comings the modulation

technique would have.

6.3 Chapter Summary

From the results of the simulation, a few rules could be gleaned. They are:

1. A modulation scheme must be used for all types of communications.
2. The use of different codes did not have a large impact on improving the signal and its robustness towards noise.
3. The size of the sub-carriers was best met by the smallest size, 64. As the sub-carrier size increased, the response of the system decreased.

For the maximum efficiency of the system, a sub-carrier size of 64, a modulation scheme using 64-QAM and the three-quarter code rate.

Chapter 7

Conclusions and Further Work

7.1 Conclusions

The results of the simulator show a relationship between the theory and the simulator. With all simulators, being able to verify that the results mirror the experiences in the outside world can be a difficult objective to achieve. With the graphical results, it is clear that as the SNR increases, the BER decreases. This point shows that as the signal has more power, it is more resistant to the impact of noise. With the different code rates, their use clearly shows an improvement in the SNR response of the system. At a BER of 0.001, the SNR is a minimum of 3 dB better than the non-coded system. This allows for a saving in costs as half the power is needed to achieve the same result. As with all scenarios, once the SNR goes below a defined level, the use of different code rates do not help the transmission. That level seems to be approximately 12 dB. For the use of different modulation schemes, a huge improvement of upwards to 20 dB can be seen using different modulation schemes. Again, once the SNR goes below approximately 10 dB, the use of any scheme does not make any improvement to the BER. The use of a larger sub-carrier size does not improve the system's response to the noisy channel. The lowest sub-carrier size, 64, has an improvement of approximately 20 dB over the 8K sub-carrier size. The most efficient system parameters would be a three-quarter code rate, a 64-QAM modulation scheme and a sub-carrier size of 64. This is very similar to the highest rate system found in the WLAN standard. The video

test shows the impact that BER would have on the quality of a broadcast transmission. Through the matching of a BER value to a picture frame, a subjective outlook can be achieved. This is point for designers to start as they can choose an appropriate BER level and see the impact it would have on the signal.

7.2 Further Work

OFDM modulation is a very flexible transmission technique. It has a broad range of uses of which there are some applications already in commercial use. The dissertation only covers a minute amount of issues that affect the modulation scheme. For the future work, the following areas could be investigated

- The use of Simulink as the simulation environment
- Introduction of narrowband noise and its impact on throughput
- Varying the size of the cyclic extension
- Use of a RF stage to fully complement the baseband stage
- The creation of a model test bed to trial the technique in real scenarios
- Consider the impact of current mobility issues, such as hand-offs, would have on OFDM as a wireless, mobile communication system.
- Research the implications on security and how various levels of security can be utilised.

References

Adams, J. (1998), *Risk*, UCL Press, London.

Allied Telesyn and Bothell (2005), 'DSL White Paper', www.alliedtelesyn.com. viewed 5t̂h July 2006.

Armstrong, J. (2002*a*), OFDM - Orthogonal Frequency Division Multiplexing, Technical report, IEEE Signal Processing Society - Victorian Chapter, Melbourne.

Armstrong, J. (2002*b*), 'OFDM Fundamentals and Emerging Applications', Monash University.

Bahai, A. R., Saltzberg, B. R. & Ergen, M. (2004), *Multi-Carrier Digital Communications: Theory and Applications of OFDM*, second edn, Springer Science + Business Media Inc., New York.

Chuang, J., Sollenberger, N. & Labs-Research, A. &. T. (2000), Beyond 3G - Wideband Wireless Data Access Based on OFDM and Dynamic Packet Assignment, Technical report, IEEE, New Jersey.

DAB (2006), *Radio Broadcasting Systems: Digital Audio Broadcasting to Mobile, Portable and Fixed Receivers (2006-06)*, www.etsi.org. viewed 7t̂h July 2006.

DVB (2004*a*), *Digital Video Broadcasting (DVB): Framing structure, channel coding and modulation for digital terrestrial television (2004-11)*, www.etsi.org. viewed 7t̂h July 2006.

DVB (2004*b*), *Digital Video Broadcasting (DVB): Transmission System for Handheld Terminals (DVB-H)(2004-11)*, www.etsi.org. viewed 7t̂h July 2006.

Engels, M. (2003), *Wireless OFDM Systems : How to make them work?*, Kluwer Academic Publishers, Norwell, Massachusetts.

Fla (2006), *OFDM for Mobile Data Communications*, www.iec.org/online/tutorials/ofdm. viewed 12t̂h July 2006.

Hanzo, L., Münster, M., Choi, B. & Keller, T. (2003), *OFDM and MC-CDMA for Broadband Multi-User Communications, WLANs and Broadcasting*, John Wiley & Sons Ltd, West Sussex, England.

Hanzo, L., Ng, S., Keller, T. & Webb, W. (2004), *Quadrature Amplitude Modulation From Basics to Adaptive Trellis-Coded, Turbo-Equalised and Space-Time Coded OFDM, CDMA and MC-CDMA Systems*, second edn, John Wiley & Sons Ltd, West Sussex, England.

Hanzo, L., Wong, C. & YEE, M. (2002), *Adaptive Wireless Transceivers : Turbo-Coded, Turbo-Equalized and Space-Time Coded TDMA, CDMA and OFDM Systems*, John Wiley & Sons Ltd, West Sussex, England.

Heiskala, J. & Terry, J. (2002), *OFDM Wireless LANs : A Theoretical and Practical Guide*, Sams Publishing, U.S.A.

Intini, A. (2000), 'Orthogonal Frequency Division Multiplexing for Wireless Networks', University of California, California.

Johnston, S., Gostelow, P. & Jones, E. (1999), *Engineering and Society an Australian perspective*, second edn, Addison-Wesley Longman Australia Pty Limited, South Melbourne.

Kraus, J. D. & Fleisch, D. A. (1999), *Electromagnetics with Applications*, fifth edn, McGraw-Hill International, Singapore.

Miller, G. M. (1993), *Modern Electronic Communication*, fourth edn, Regents/Prentice Hall International, Englewood Cliffs, New Jersey.

Prasad, R. (2004), *OFDM for Wireless Communications Systems*, Artech House, U.S.A.

WLA (2003*a*), *Part 11: Wireless LAN Medium Access Control (MAC) and Physical (PHY) specifications. Amendment 4: Further Higher Data Rate Extension in the 2.4 GHz Band*, www.ieee.org. viewed 7t̂h March 2006.

WLA (2003*b*), *Part 11: Wireless LAN Medium Access Control (MAC) and Physical (PHY) specifications. High-speed Physical Layer in the 5 GHz Band*, www.ieee.org. viewed 7ĥ March 2006.

Young, P. H. (1985), *Electronic Communication Techniques*, second edn, Merrill Publishing, Columbus, Ohio.

Appendix A

Project Specification

University of Southern Queensland

FACULTY OF ENGINEERING AND SURVEYING

ENG 4111/4112 Research Project

PROJECT SPECIFICATION

For:	**Barry DUNBAR**
TOPIC:	Software Simulator Development for Orthogonal Frequency Division Multiplexing (OFDM) Modulation
SUPERVISOR:	Wei Xiang
ENROLMENT:	ENG 4111 - S1, X, 2006 ENG 4122 - S2, X, 2006
PROJECT AIM:	This project aims to develop a software simulator to determine the robustness of an Orthogonal Frequency Division Multiplexing (OFDM) modulation transceiver whilst subjected to varying system parameters.
PROGRAMME:	**ISSUE A, 27th March 2006**

1. Research the methodology of Orthogonal Frequency Division Multiplexing (OFDM) modulation.
2. Examine a radio channel and discuss the impact on communication signals and systems.
3. Design an Orthogonal Frequency Division Multiplexing (OFDM) modulation simulator using MATLAB.
4. Verify the simulation results matching the theoretical results of an Orthogonal Frequency Division Multiplexing (OFDM) transceiver system.
5. Analyse the simulation results relative to different scenarios and rationalise these results with regard to network integrity.

As time permits:

1. Consider the possible hand-over problems, which become apparent to mobile network users, and solutions to these problems.
2. Develop the simulator so that it incorporates various levels of security.

AGREED:_____ (Student) _____ (Supervisor)

(Dated)_____/_____/_____

Appendix B

Program Code

B.1 OFDM_Simulator

```
%% TITLE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%                Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Setup a Guide screen
function varargout = OFDM_simulator(varargin)


%Program Settings and Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% OFDM_SIMULATOR M-file for OFDM_simulator.figTITLE
%
%      OFDM_SIMULATOR, by itself, creates a new OFDM_SIMULATOR or
%      raises the existing
%      singleton*.
%
%      H = OFDM_SIMULATOR returns the handle to a new OFDM_SIMULATOR
%      or the handle to
%      the existing singleton*.
%
%      OFDM_SIMULATOR('CALLBACK',hObject,eventData,handles,...)
%      calls the local
%      function named CALLBACK in OFDM_SIMULATOR.M with the given
%      input arguments.
%
%      OFDM_SIMULATOR('Property','Value',...) creates a new
%      OFDM_SIMULATOR or raises the
```

```
%      existing singleton*.  Starting from the left, property
%      value pairs are
%      applied to the GUI before OFDM_simulator_OpeningFunction
%      gets called.  An
%      unrecognized property name or invalid value makes property
%      application
%      stop.  All inputs are passed to OFDM_simulator_OpeningFcn
%      via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
%      only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help OFDM_simulator

% Last Modified by GUIDE v2.5 25-Oct-2006 00:33:12

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Begin initialization code - DO NOT EDIT
%
gui_Singleton = 1; gui_State = struct('gui_Name',       mfilename,
...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @OFDM_simulator_OpeningFcn, ...
                   'gui_OutputFcn',  @OFDM_simulator_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% --- Executes just before OFDM_simulator is made visible.
%
function OFDM_simulator_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to OFDM_simulator (see VARARGIN)

%Initialising Variables
    handles.noSubCar      = 52;
    handles.noDataCar     = 48;
    handles.noPilotCar    = 4;
    handles.noPadCar      = 12;
    handles.noCeCar       = 16;
    hsubnono = handles.noSubCar + handles.noPadCar + handles.noCeCar;
    handles.totSubChan    = hsubnono;
    handles.noTxAnt       = 1;
    handles.noRxAnt       = 1;
    handles.inpTypeVal    = 1;
    handles.codTypeVal    = 1;
    handles.modTypeVal    = 1;
    handles.modkval       = 1;
    handles.codextrabits  = 0;
    handles.intextrabits  = 0;
```

```
    handles.modextrabits  = 0;
    handles.intperm       = 0;
    handles.subCarSizVal  = 1;
    handles.ifftextrabits = 0;
    handles.inpVidcou     = 0;
    handles.pakNumVal     = 10000;
    handles.bandNumVal    = 20000000;
    handles.ebminNumVal   = 5;
    handles.ebstpNumVal   = 2;
    handles.ebmaxNumVal   = 26;
    handles.fadTypeVal    = 1;
    handles.outputbutnum  = 1;


guidata(hObject,handles);
% Choose default command line output for OFDM_simulator
    handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes OFDM_simulator wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = OFDM_simulator_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Get default command line output from handles structure
varargout{1} = handles.output;

%OFDM_simulator is now visible
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Various Additions to screen layout

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Input Type Pop-up Menu
% --- Executes on selection change in InputType.
function InputType_Callback(hObject, eventdata, handles)
% hObject    handle to InputType (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns InputType
% contents as cell array
% contents{get(hObject,'Value')} returns selected item from InputType

val = get(hObject,'Value'); handles.inpTypeVal = val;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function InputType_CreateFcn(hObject, eventdata, handles)
% hObject    handle to InputType (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
```

```
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
%End of Input Type Pop-up Menu
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Coding Type Pop-Up menu
% --- Executes on selection change in CodeType.
function CodeType_Callback(hObject, eventdata, handles)
% hObject    handle to CodeType (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns CodeType
% contents as cell array
% contents{get(hObject,'Value')} returns selected item from CodeType

val = get(hObject,'Value'); handles.codTypeVal = val;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function CodeType_CreateFcn(hObject, eventdata, handles)
% hObject    handle to CodeType (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
%End of Coding Type Pop-Up Menu
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Modulation Type Pop-up Menu
% --- Executes on selection change in ModulationType.
function ModulationType_Callback(hObject, eventdata, handles)
% hObject    handle to ModulationType (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns ModulationType
% contents as cell array
% contents{get(hObject,'Value')} returns selected item from ModulationType

val = get(hObject,'Value'); handles.modTypeVal = val;
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function ModulationType_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ModulationType (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
%End of Modulation Type Pop-up menu
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Press Button "RUN" to start simulation
% --- Executes on button press in RunSimulation.
```

```
function RunSimulation_Callback(hObject, eventdata, handles)
% hObject    handle to RunSimulation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Gather all variable values
handles;


        [kp,mstrp,cstrp,sc,dc,pc,zc,gc,tc] = ofdm_para(handles);
        handles.modkval = kp;
        handles.modstr = mstrp;
        handles.codstr = cstrp;
        handles.noSubCar      = sc;
        handles.noDataCar     = dc;
        handles.noPilotCar    = pc;
        handles.noPadCar      = zc;
        handles.noCeCar       = gc;
        handles.totSubChan    = tc;
        guidata(hObject,handles);

        handles
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Receiver output Analysis
% BER Computation
% Compare x and z to obtain the number of errors and
% the bit error rate.
%handles;

       bnum = handles.outputbutnum;
       btyp = handles.inpTypeVal;
```

```
        if  btyp ==3;

        [dout, mout] = ofdm_sysvid(handles);

else

switch  bnum

    case 1  %Single test

        [dout, mout] = ofdm_systest(handles);

    case 2  %Code Rate Comparsion test

        [dout, mout] = ofdm_syscod(handles);

    case 3  %Modulation Type Comparison test

        [dout, mout] = ofdm_sysmod(handles);

    case 4  %Carrier Size Comparison test

        [dout, mout] = ofdm_syscar(handles);
end end

%% Press Button "Close" to end simulation and close application
%
% --- Executes on button press in CloseSimulation.
function CloseSimulation_Callback(hObject, eventdata, handles)
% hObject    handle to CloseSimulation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
        close all;

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
% str2double(get(hObject,'String')) returns contents of edit1 as a double

pval = get(hObject,'String'); pakval = str2double(pval);
    handles.pakNumVal = pakval;
    guidata(hObject,handles);
    handles;
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1


% --- Executes on selection change in SubCarrierSize.
function SubCarrierSize_Callback(hObject, eventdata, handles)
% hObject    handle to SubCarrierSize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns SubCarrierSize
% contents as cell array
% contents{get(hObject,'Value')} returns selected item from
% SubCarrierSize

    val = get(hObject,'Value');
    handles.subCarSizVal = val;
    guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function SubCarrierSize_CreateFcn(hObject, eventdata, handles)
% hObject    handle to SubCarrierSize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
% str2double(get(hObject,'String')) returns contents of edit2 as a
% double

bval = get(hObject,'String'); banval = str2double(bval);

    handles.bandNumVal = banval;
    guidata(hObject,handles);
    handles;

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end

% --- Executes on selection change in popupmenu6.
function popupmenu6_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu6
% contents as cell array
% contents{get(hObject,'Value')} returns selected item from popupmenu6

    val = get(hObject,'Value');
    handles.fadTypeVal = val;
    guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function popupmenu6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
% str2double(get(hObject,'String')) returns contents of edit3 as a double

    val = get(hObject,'String');
    ebminval = str2double(val);
    handles.ebminNumVal = ebminval;
    guidata(hObject,handles);
    handles;

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
```

```
% str2double(get(hObject,'String')) returns contents of edit4 as a
%double

    val = get(hObject,'String');
    ebstpval = str2double(val);
    handles.ebstpNumVal = ebstpval;
    guidata(hObject,handles);
    handles;

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
% str2double(get(hObject,'String')) returns contents of edit5 as a
% double

val = get(hObject,'String'); ebmaxval = str2double(val);
```

```
    handles.ebmaxNumVal = ebmaxval;
    guidata(hObject, handles);
    handles;

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function uipanel12_SelectionChangeFcn(hObject,eventdata,handles)
%function uibuttongroup1_SelectionChangeFcn(hObject,eventdata,handles)
% hObject    handle to uipanel12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Get Tag of selected object

switch get(hObject,'Tag')

    case 'radiobutton4'

        val = get(hObject,'Tag');
        handles.outputbutval = val;
```

```
        handles.outputbutnum = 1;
        guidata(hObject,handles);
        handles;


    case 'radiobutton2'

        val = get(hObject,'Tag');
        handles.outputbutval = val;
        handles.outputbutnum = 2;
        guidata(hObject,handles);
        handles;


    case 'radiobutton1'

        val = get(hObject,'Tag');
        handles.outputbutval = val;
        handles.outputbutnum = 3;
        guidata(hObject,handles);
        handles;

    case 'radiobutton5'

        val = get(hObject,'Tag');
        handles.outputbutval = val;
        handles.outputbutnum = 4;
        guidata(hObject,handles);
        handles;
end

% --------------------------------------------------------------------
function Untitled_1_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to Untitled_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

B.2 OFDM_para

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Simulator Parameters

function [kl,mstrl,cstrl,hsc,hdc,hpc,hzc,hgc,htc] =
    ofdm_para(handles);


handles;

%Assigning number of bits to Modulation value and string
    if handles.modTypeVal == 1
    k = 1;
    mstr = 'NONE';
else
    if handles.modTypeVal == 2
        k = 1;
        mstr = 'BPSK';
    else
        if handles.modTypeVal == 3
            k = 2;
            mstr = 'QPSK';
```

```
        else
            if handles.modTypeVal == 4
                k = 4;
                mstr = '16QAM';
            else
                k = 6;
                mstr = '64QAM';
            end
        end
    end
end

%Assigning code string to code value
    kl = k; mstrl = mstr;

if handles.codTypeVal == 1
    cstr = 'NONE';
else
    if handles.codTypeVal == 2
        cstr = '1/2';
    else
        if handles.codTypeVal == 3
            cstr = '2/3';
        else
            cstr = '3/4';
        end
    end
end

cstrl = cstr;

%Assigning the Sub-carrier sizes
    popupsubsiz_sel_id = handles.subCarSizVal;
```

```
switch  popupsubsiz_sel_id

    %64
    case 1

        hsc = 52;
        hdc = 48;
        hpc = 4;
        hzc = 12;
        hgc = 16;
        htc = 80;


    %256
    case 2

        hsc = 208;
        hdc = 192;
        hpc = 16;
        hzc = 48;
        hgc = 64;
        htc = 320;


    %512
    case 3

        hsc = 416;
        hdc = 384;
        hpc = 32;
        hzc = 96;
        hgc = 128;
```

```
        htc = 640;


    %1024
    case 4

        hsc = 832;
        hdc = 768;
        hpc = 64;
        hzc = 192;
        hgc = 256;
        htc = 1280;


    %2048
    case 5

        hsc = 1664;
        hdc = 1536;
        hpc = 128;
        hzc = 384;
        hgc = 512;
        htc = 2560;


    %4096
    case 6

        hsc = 3328;
        hdc = 3072;
        hpc = 256;
        hzc = 768;
        hgc = 1024;
```

```
        htc = 5120;


    %8192
    case 7

        hsc = 6656;
        hdc = 6144;
        hpc = 512;
        hzc = 1536;
        hgc = 2048;
        htc = 10240;

end

%-------------------------------------------------------------------------
\section{ofdm${\_}$systest}
%-------------------------------------------------------------------------

\begin{lstlisting}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% System Simulator Analysis

% BER Computation
% Compare input and output to obtain the number of errors and
% the bit error rate.
```

```
function [osimdat,osimout] = ofdm_systest(handles);

        handles; ek = handles.modkval;
        ebmin = handles.ebminNumVal;
        ebstep = handles.ebstpNumVal;
        ebmax = handles.ebmaxNumVal;

        Ebi = [ebmin:ebstep:ebmax];
        snri = Ebi + 3 + 10*log10(ek);
        ja = ((ebmax - ebmin)/ebstep)+1;
        ebi = 5;
        for jj = 1:ja
%% Transmitter

        handles;

        [ofdmtxout,dbitin,dal,cdatabinlen ,cdatalen ,colmapl,...
            coe,ine,inpe,moe,kvl,ifx, pt] = ofdm_tx(handles);
        handles.inpFileSize = dal;
        handles.codextrabits = coe;
        handles.intextrabits = ine;
        handles.intperm = inpe;
        handles.modextrabits = moe;
        handles.ifftextrabits = ifx;
        handles.txpreamblevals = pt;
        handles;

%% Channel

        tic;
        [simchanout] = ofdm_chann(ofdmtxout,kvl,ebi,handles);
        toc;
%% Receiver
```

```
        [ofdmrxout] = ofdm_rx(simchanout,handles);


%% Analysis
% BER Computation
% Compare x and z to obtain the number of errors and
% the bit error rate.
        handles;
        datain = dbitin;
        output = ofdmrxout;
        [number_of_errors,bit_error_rate] = biterr(datain,output);
        bitout(:,jj) = bit_error_rate;
        ebi = ebi + 3;
end

%% Output

        osimdat = dbitin;
        osimout = ofdmrxout;
        handles;
        bitout
        ek;
        bitouta(1,:) = bitout(1,:);

%Plot  results.
        figure(1);
        pi = 1;
        semilogy(snri,bitout(pi,:),'r');
        xlabel('SNR (dB)');
        ylabel('Bit Error Rate (BER)');
        grid on;
        drawnow;
```

```
        title(['OFDM Transmission System Test result']);
```

B.3 ofdm_syscod

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%                Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% System Simulator Analysis

% BER Computation
% Compare input and output to obtain the number of errors and
% the bit error rate.

function [osimdat,osimout] = ofdm_syscod(handles);

        handles;
        ek = handles.modkval;
        ebmin = handles.ebminNumVal;
        ebstep = handles.ebstpNumVal;
        ebmax = handles.ebmaxNumVal;
        Ebi = [ebmin:ebstep:ebmax];
        snri = Ebi + 3 + 10*log10(ek);
        ja = ((ebmax - ebmin)/ebstep)+1;

        handles.codTypeVal = 1;
for ii = 1:4

        handles.codTypeVal;
        handles;
        ebi = 5;
```

```
for jj = 1:ja
%% Transmitter

        handles;

        [ofdmtxout,dbitin,dal,cdatabinlen ,cdatalen ,colmapl,...
            coe,ine,inpe,moe,kvl,ifx, pt] = ofdm_tx(handles);
        handles.inpFileSize = dal;
        handles.codextrabits = coe;
        handles.intextrabits = ine;
        handles.intperm = inpe;
        handles.modextrabits = moe;
        handles.ifftextrabits = ifx;
        handles.txpreamblevals = pt;
        handles;

%% Channel

        tic;
        [simchanout] = ofdm_chann(ofdmtxout,kvl,ebi,handles);
        toc;
%% Receiver

        [ofdmrxout] = ofdm_rx(simchanout,handles);

%% Analysis
% BER Computation
% Compare x and z to obtain the number of errors and
% the bit error rate.
        handles;
        datain = dbitin;
        output = ofdmrxout;
        [number_of_errors,bit_error_rate] = biterr(datain,output);
```

```
        bitout(ii,jj) = bit_error_rate;
        ebi = ebi + 3;
end

        handles.codTypeVal = handles.codTypeVal + 1;
end
%% Output

        osimdat = dbitin;
        osimout = ofdmrxout;
        handles;
        bitout
        ek;
        bitouta(1,:) = bitout(1,:);

% Plot results.

        figure(1);
        pi = 1;
        semilogy(snri,bitout(pi,:),'-r+');
        xlabel('SNR (dB)');
        ylabel('Bit Error Rate (BER)');
        grid on;
        drawnow;
        hold on;
        semilogy(snri,bitout((pi+1),:),'-g*');
        semilogy(snri,bitout((pi+2),:),'-bx');
        semilogy(snri,bitout((pi+3),:),'-cp');
        legend('NONE','1/2','2/3','3/4');
        title(['OFDM Transmission, "',num2str(handles.modstr),...
            '" Modulation Rate, Varying Code Rates ']);

hold off;
```

B.4 ofdm_sysmod

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%                Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% System Simulator Analysis

% BER Computation
% Compare input and output to obtain the number of errors and
% the bit error rate.

function [osimdat,osimout] = ofdm_sysmod(handles);

        handles;
        ebmin = handles.ebminNumVal;
        ebstep = handles.ebstpNumVal;
        ebmax = handles.ebmaxNumVal;
        Ebi = [ebmin:ebstep:ebmax];
        ja = ((ebmax - ebmin)/ebstep)+1;
        handles.modTypeVal = 1;
        [kpm] = ofdm_para(handles);
        ek = kpm;
for ii = 1:5

        handles;
        ebi = 5;

for jj = 1:ja
%% Transmitter
```

```
        handles;
        [ofdmtxout,dbitin,dal,cdatabinlen ,cdatalen ,...
            colmapl ,coe,ine,inpe,moe,kvl,ifx,...
            pt] = ofdm_tx(handles);

        handles.inpFileSize = dal;
        handles.codextrabits = coe;
        handles.intextrabits = ine;
        handles.intperm = inpe;
        handles.modextrabits = moe;
        handles.ifftextrabits = ifx;
        handles.txpreamblevals = pt;
        ifx;
        handles;

%% Channel

        tic;
        [simchanout] = ofdm_chann(ofdmtxout,kvl,ebi,handles);
        toc;
%% Receiver

        [ofdmrxout] = ofdm_rx(simchanout,handles);


%% Analysis
% BER Computation
% Compare x and z to obtain the number of errors and
% the bit error rate.
        handles;
        datain = dbitin;
        output = ofdmrxout;
```

```
        [number_of_errors,bit_error_rate] = biterr(datain,output);
        bitout(ii,jj) = bit_error_rate;
        snrout(ii,jj) = ebi + 3 + 10*log10(kvl);
        ebi = ebi + 3;
end

        handles.modTypeVal = handles.modTypeVal + 1;

end
%% Output

        osimdat = dbitin;
        osimout = ofdmrxout;
        handles;
        bitout
        ek;
        bitouta(1,:) = bitout(1,:);
        snrout;
% Plot results.

        figure(1);
        pi = 1;
        semilogy(snrout(pi,:),bitout(pi,:),'-r+');
        xlabel('SNR (dB)');
        ylabel('Bit Error Rate (BER)');
        grid on;
        drawnow;
        hold on;
        semilogy(snrout((pi+1),:),bitout((pi+1),:),'-g*');
        semilogy(snrout((pi+2),:),bitout((pi+2),:),'-bx');
        semilogy(snrout((pi+3),:),bitout((pi+3),:),'-cp');
        semilogy(snrout((pi+4),:),bitout((pi+4),:),'-md');
        legend('NONE','BPSK','QPSK','16QAM','64QAM');
```

```
        title(['OFDM Transmission, "',num2str(handles.codstr),...
            '" Code Rate, Varying Modulation Rates ']);
        hold off;
```

B.5 ofdm_syscar

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% System Simulator Analysis

% BER Computation
% Compare input and output to obtain the number of errors and
% the bit error rate.

function [osimdat,osimout] = ofdm_syscar(handles);

        handles;
        ebmin = handles.ebminNumVal;
        ebstep = handles.ebstpNumVal;
        ebmax = handles.ebmaxNumVal;
        Ebi = [ebmin:ebstep:ebmax];
        ja = ((ebmax - ebmin)/ebstep)+1;
        handles.subCarSizVal = 1;

for ii = 1:7

        handles;
        [kpm,m1trl,c1trl,h1c,h2c,h3c,h4c,h5c,...
```

```
        h6c] = ofdm_para(handles);
    ek = kpm;
    msr = m1trl;
    cst = c1trl;
    handles.noSubCar   = h1c;
    handles.noDataCar  = h2c;
    handles.noPilotCar = h3c;
    handles.noPadCar   = h4c;
    handles.noCeCar    = h5c;
    handles.totSubChan = h6c;
    handles;
    ebi = 5;

for jj = 1:ja
%% Transmitter


    handles;
    [ofdmtxout,dbitin,dal,cdatabinlen ,cdatalen ,...
        colmapl ,coe,ine,inpe,moe,kvl,ifx,...
        pt] = ofdm_tx(handles);

    handles.inpFileSize = dal;
    handles.codextrabits = coe;
    handles.intextrabits = ine;
    handles.intperm = inpe;
    handles.modextrabits = moe;
    handles.ifftextrabits = ifx;
    handles.txpreamblevals = pt;
    ifx;
    handles;

%% Channel
```

```
        tic;
        [simchanout] = ofdm_chann(ofdmtxout,kvl,ebi,handles);
        toc;
%% Receiver

        [ofdmrxout] = ofdm_rx(simchanout,handles);


%% Analysis
% BER Computation
% Compare x and z to obtain the number of errors and
% the bit error rate.

        handles;
        datain = dbitin;
        output = ofdmrxout;
        [number_of_errors,bit_error_rate] = biterr(datain,output);
        bitout(ii,jj) = bit_error_rate;
        snrout(ii,jj) = ebi + 3 + 10*log10(kvl);
        ebi = ebi + 3;
end

        handles.subCarSizVal = handles.subCarSizVal + 1;

end
%% Output

        osimdat = dbitin;
        osimout = ofdmrxout;
        handles;
        bitout
        ek;
```

```
        bitouta(1,:) = bitout(1,:);
        snrout;
% Plot  results.

        figure(1);
        pi = 1;
        semilogy(snrout(pi,:),bitout(pi,:),'-r+');
        xlabel('SNR (dB)');
        ylabel('Bit Error Rate (BER)');
        grid on;
        drawnow;
        hold on;
        semilogy(snrout((pi+1),:),bitout((pi+1),:),'-g*');
        semilogy(snrout((pi+2),:),bitout((pi+2),:),'-bx');
        semilogy(snrout((pi+3),:),bitout((pi+3),:),'-cp');
        semilogy(snrout((pi+4),:),bitout((pi+4),:),'-md');
        semilogy(snrout((pi+5),:),bitout((pi+5),:),'-ys');
        semilogy(snrout((pi+6),:),bitout((pi+6),:),'-kh');
        legend('64','256','512','1024','2048','4096','8192');
        title(['OFDM Transmission, Varying Sub Carrier Size ']);
        hold off;
```

B.6 ofdm_sysvid

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% System Simulator Analysis
```

```
% BER Computation
% Compare input and output to obtain the number of errors and
% the bit error rate.

function [osimdat,osimout] = ofdm_sysvid(handles);

        handles;
        ebmin = handles.ebminNumVal;
        ebstep = handles.ebstpNumVal;
        ebmax = handles.ebmaxNumVal;
        Ebi = [ebmin:ebstep:ebmax];

        %must be = or < number of frames
        ja = ((ebmax - ebmin)/ebstep)+1;
        ebi = 35;


for jj = 1:12

        handles.inpVidcou = jj;
        handles;
%% Transmitter

        handles;
        [ofdmtxout,dbitin,dal,vbl,vcl,vml,coe,ine,...
            inpe,moe,kvl,ifx, pt] = ofdm_tx(handles);

        handles.inpFileSize = dal;
        handles.inpVidbinLen = vbl;
        handles.inpVidcdLen = vcl;
        handles.inpVidcMLen = vml;
        handles.codextrabits = coe;
        handles.intextrabits = ine;
```

```
        handles.intperm = inpe;
        handles.modextrabits = moe;
        handles.ifftextrabits = ifx;
        handles.txpreamblevals = pt;
        handles;

%% Channel

        tic;
        [simchanout] = ofdm_chann(ofdmtxout,kvl,ebi,handles);
        toc;
%% Receiver

        [ofdmrxout] = ofdm_rx(simchanout,handles);

%% Analysis
% BER Computation
% Compare x and z to obtain the number of errors and
% the bit error rate.

        handles;
        datain = dbitin;
        output = ofdmrxout;
        rxmovbin = ofdmrxout;
        rxml = vbl;
        xmovd = rxmovbin(1:rxml);
        rmbinl = length(rxmovbin);
        movMapd = rxmovbin((rxml+1):rmbinl);
        xmdmin = min(xmovd);
        xmdmax = max(xmovd);
        xmdran = xmdmax - xmdmin;
        xmdsc = xmovd/xmdran;
        xmdme = mean(xmdsc);
```

```
        xmdscf = 0.5 - xmdme;
        xmdsca = xmdsc + xmdscf;
        xmdrnd = round(xmdsca);
        xmove = reshape(xmdrnd,[],8);
        xmovf = bi2de(xmove);
        xl = vcl;
        xw = (vbl/8)/vcl;
        xmovg = reshape(xmovf,xl,xw);

        mMdmin = min(movMapd);
        mMdmax = max(movMapd);
        mMdran = mMdmax - mMdmin;
        mMdsc = movMapd/mMdran;
        mMdme = mean(mMdsc);
        mMdscf = 0.5 - mMdme;
        mMdsca = mMdsc + mMdscf;
        mMdrnd = round(mMdsca);
        movMape = reshape(mMdrnd,[],8);
        movMapf = bi2de(movMape);
        movMapff = double(movMapf);
        movMapfff = movMapff / 255;
        ml = vml;
        mw = ((rmbinl - vbl)/8)/vml;
        movMapg = reshape(movMapfff,ml,mw);
        rxfra(jj) = im2frame(xmovg+1,movMapg);
        [number_of_errors,bit_error_rate] = biterr(datain,output);
        bitout(:,jj) = bit_error_rate;
        snrout(:,jj) = ebi + 3 + 10*log10(kvl);
        ebi = ebi - 3;
end

        rxmov = rxfra;
```

```
%% Output

        osimdat = dbitin;
        osimout = ofdmrxout;
        handles;
        bitout
        bitouta(1,:) = bitout(1,:);
        snrout;
        movie2avi(rxmov,'rxclocka.avi','compression',...
            'RLE','colormap',movMapg,'fps',1);
        rxfileinfo = aviinfo('rxclocka.avi');
```

B.7 ofdm_tx

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Transmitter

function [txout,databitin,dl,cdbl,cdl,cml,ce,ie,...
    ip,me,kv,ife, ptx ] = ofdm_tx(handles);

handles;

%% Input function call

        [datain,datalen,cdatbinl,...
            cdatl,colmapl] = ofdm_inputSelect(handles);
```

```
        [dp,dq] = size(datain);

%% Coding function call

        [codemessout,codeextbit] = ofdm_coder(datain,...
            handles.codTypeVal);

        interlvdin = codemessout;

%% Interleaver function call

        [interlvdmessout,interlvdextbit,...
            interlvdperm] = ofdm_interleaver(interlvdin,...
            handles.noDataCar,handles.modTypeVal);

        modin = interlvdmessout;

%% Modulator function call

        [modsymoutput, modextbits,...
            kval] = ofdm_modulator(modin,handles.modTypeVal);

        modsymoutput;

%% IFFT function call

        [ifftoutput, iffext] = ofdm_ifft(modsymoutput,handles);
        ifftoutput;

%% Preamble function call

        preamble = ofdm_pream();
```

```
%% Add preamble and data together

        frameout = [preamble ifftoutput];

%% Output

        txout = frameout;

        %for BER tests
        databitin = datain;
        dl = datalen;
        cdbl = cdatbinl;
        cdl = cdatl;
        cml = colmapl;
        ce = codeextbit;
        ie = interlvdextbit;
        ip = interlvdperm;
        me = modextbits;
        kv = kval;
        ife = iffext;
        ptx = preamble;
```

B.8 ofdm_inputSelect

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%                Project - OFDM Simulator
%
%Barry Dunbar   0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% InputSelect
function [inpout,inputLength,xmovcl,xmovl,...
```

```
    movMapl] = ofdm_inputSelect(handles)

        %Assigning variables
        popupmodselid = handles.inpTypeVal;
        s = handles.pakNumVal;

switch  popupmodselid

    case 1  %All ones

        %Generate all '1's bit stream
        dataorig = ones(s,1);

        %Output variables
        inpout = dataorig;
        inputLength = length(dataorig);
        xmovcl = 0;
        xmovl = 0;
        movMapl = 0;

    case 2  %Random ones or zeros

        %Generate random bit stream
        dataorig = randsrc(s,1,[0 1]);

        %Output variables
        inpout = dataorig;
        inputLength = length(dataorig);
        xmovcl = 0;
        xmovl = 0;
        movMapl = 0;

    case 3  %Video file read in
```

```
%Read in movie file
mov = aviread('clock.avi');

fileinfo = aviinfo('clock.avi');
movfra = fileinfo.NumFrames;

%CDATA matrix
imov = handles.inpVidcou;

%convert frame to image
[xmov,movMap] = frame2im(mov(imov));

[xl xw]= size(xmov);
xa = xl * xw;

%1 column * many rows
xmova = reshape(xmov,xa,1);

%convert to type double
xmovab = double(xmova);

%convert to binary stream
xmovb = de2bi(xmovab,8);

[xbl xbw]= size(xmovb);
xba = xbl * xbw;

%1 row * many colums
xmovc = reshape(xmovb,1,xba);

%COLOURMAP matrix
[ml mw]= size(movMap);
```

```
        ma = ml * mw;

        %1 column * many rows
        movMapa = reshape(movMap,ma,1);

        %convert to whole integers
        movMapaa = movMapa * 255;

        %convert to binary stream
        movMapb = de2bi(movMapaa,8);

        [mbl mbw]= size(movMapb);
        mba = mbl * mbw;

        %1 row * many columns
        movMapc = reshape(movMapb,1,mba);

        %append CDATA and COLOURMAP into one bit stream
        movbin = [xmovc movMapc];

        movbina = double(movbin);
        movbinaa = movbina';

        %video file output variables
        inpout = movbinaa;
        inputLength = length(movbinaa);
        xmovcl = length(xmovc);
        xmovl = xl;
        movMapl = ml;
end
```

B.9 ofdm_coder

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%                Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Coder
function [cod_out,cod_ext] = ofdm_coder(cod_in, codType)

        %Assigning variables
        codeIn = cod_in;
        popupcod_sel_id = codType;


switch  popupcod_sel_id


    case 1  %No coding

        %Out equals In
        cod_out = codeIn;

        %Output variable
        cod_ext = 0;

    case 2  %1/2 Rate

        %Trellis code
        t = poly2trellis(7,[133 171]);

        %Encode
        tcode = convenc(codeIn,t);
```

```
        %Length is (2*length)
        codeLengthOrig = length(tcode);

        %Output variables
        cod_out = tcode;
        cod_ext = 0;

    case 3  %2/3 Rate

        %Trellis code
        t = poly2trellis(7,[133 171]);

        %Encode
        tcode = convenc(codeIn,t);

        %Length is (2*length)* 3/4 rounded up
        codeLengthOrig = length(tcode);

        %Puncture code for 2/3 rate
        punct23code = tcode;
        punct23code(4:4:end)=[];

        codeLength23 = length(punct23code);
        codeLength23Orig = 4/3*codeLength23;
        punct23extbit = codeLength23Orig - codeLengthOrig;

        %Output variables
        cod_out = punct23code;
        cod_ext = punct23extbit;

    case 4  %3/4 Rate
```

```
        %Trellis code
        t = poly2trellis(7,[133 171]);

        %Encode
        tcode = convenc(codeIn,t);

        %Length is (2*length)*3/4 rounded up
        codeLengthOrig = length(tcode);

        %Puncture code for 3/4 rate
        punct34code = tcode;
        punct34code(4:6:end)=[];
        punct34code(4:5:end)=[];

        codeLength34 = length(punct34code);
        codeLength34Orig = 3/2*codeLength34;
        punct34extbit = codeLength34Orig - codeLengthOrig;

        %Output variables
        cod_out = punct34code;
        cod_ext = punct34extbit;

end
```

B.10 ofdm_interleaver

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%                Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% Interleaver
function [interlvd_out,interlvd_ext,...
    inter_perm] = ofdm_interleaver(interlvd_in,NoSub,MapType)

        %Assigning variables
        %Input
        interleavIn = interlvd_in;

        %Length of Input
        interleavLen = length(interleavIn);

        %No. of Sub-Carriers
        NoSubc = NoSub;

        %Modulation Type
        ModType = MapType;

        %Assigning number of bits per symbol

        switch ModType

            case 1

                NoBits = 1;

            case 2

                NoBits = 1;

            case 3
                NoBits = 2;

            case 4
```

```
        NoBits = 4;

    case 5

        NoBits = 6;
end


%divide input by no. of bits per symbol
%and no. of Sub carriers
leav_sym_len = interleavLen/(NoSubc*NoBits);

%round up to whole number
leav_len_rnd  = ceil(leav_sym_len);

%new lenghth * bits * subcarriers
inputLength = leav_len_rnd * (NoSubc*NoBits);

%extra bits amount
levExtBitsLen = inputLength - interleavLen;

%make extra bits all ones
levExtBits = ones(1,levExtBitsLen);

%Add extra bits to input
lev_inputresh = interleavIn.';
newLev = [lev_inputresh levExtBits];
newLevLen = length(newLev);

%1 row * many columns.
newLev = newLev.';
```

```
%Reshape into No. of Subcarriers * numerous columns
levMess = reshape(newLev,(NoSubc*NoBits),[]);

%Set up permutation
q = randperm((NoSub*NoBits)).';

%Interleave data
newintrlvd = intrlv(levMess,q);

%make serial
newinterlved = reshape(newintrlvd,1,[]);

%output variables
interlvd_out = newinterlved;
interlvd_ext = levExtBitsLen;
inter_perm   = q;
```

B.11 ofdm_modulator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Modulator
function [mod_out, extbits, kbit] = ofdm_modulator(mod_in, modType)

%Assigning variables
mod_input = mod_in;
popupmod_sel_id = modType;
inModLengthOrig = length(mod_input);
```

```
switch  popupmod_sel_id

    case 1  %No Modulation

        mod_out = mod_input;
        extbits = 0;
        kbit = 1;

    case 2  %BPSK Modulation

        %Modulation parrameter
        M=2;

        %Number of bits per symbol eg. 1 bits per symbol
        k = log2(M);

        %Padding data length with extra '1's for multiple
        %of symbol size
        input_sym_len = inModLengthOrig/k;
        sym_len_rnd  = ceil(input_sym_len);
        inputLength = sym_len_rnd * k;
        msgExtBitsLen = inputLength - inModLengthOrig;
        msgExtBits = ones(1,msgExtBitsLen);
        newMess = [mod_input msgExtBits];
        newMessLen = length(newMess);
        newMess = newMess';

        %symbol length
        inputSymbolLength = newMessLen/k ;

        %Reshape for matrix of 1 rows * numerous columns
        msg_sym_reshape = reshape(newMess,k,inputSymbolLength);
```

```
        %Decimalisation
        msg_sym = bi2de(msg_sym_reshape.','left-msb');

        %Modulation
        msg_tx = pskmod(msg_sym,M);

        %Output variables
        mod_out = msg_tx;
        extbits = msgExtBitsLen;
        kbit = 1;

        %Scatterplot
        %h1 = scatterplot(msg_tx);
        %axis([-3 3 -3 3]); % Set axis ranges.


    case 3  %QPSK Modulation

        %Modulation parrameter
        M=4;

        %Number of bits per symbol eg. 2 bits per symbol
        k = log2(M);

        %Padding data length with extra '1's for multiple
        %of symbol size
        input_sym_len = inModLengthOrig/k;
        sym_len_rnd  = ceil(input_sym_len);
        inputLength = sym_len_rnd * k;
        msgExtBitsLen = inputLength - inModLengthOrig;
        msgExtBits = ones(1,msgExtBitsLen);
        newMess = [mod_input msgExtBits];
```

```
        newMessLen = length(newMess);
        newMess = newMess';

        %symbol length
        inputSymbolLength = newMessLen/k;

        %Reshape for matrix of 2 rows * numerous columns
        msg_sym_reshape = reshape(newMess,k,inputSymbolLength);

        %Decimalisation
        msg_sym = bi2de(msg_sym_reshape.','left-msb');

        %Modulation
        msg_tx = pskmod(msg_sym,M);

        %Output variables
        mod_out = msg_tx;
        extbits = msgExtBitsLen;
        kbit = 2;

        %Scatterplot
        %h1 = scatterplot(msg_tx);
        %axis([-3 3 -3 3]); % Set axis ranges.

    case 4  %16QAM Modulation

        %Modulation parrameter
        M=16;

        %Number of bits per symbol eg. 4 bits per symbol
        k = log2(M);

        %Padding data length with extra '1's for multiple
```

```
        %of symbol size
        input_sym_len = inModLengthOrig/k;
        sym_len_rnd  = ceil(input_sym_len);
        inputLength = sym_len_rnd * k;
        msgExtBitsLen = inputLength - inModLengthOrig;
        msgExtBits = ones(1,msgExtBitsLen);
        newMess = [mod_input msgExtBits];
        newMessLen = length(newMess);
        newMess = newMess';

        %symbol length
        inputSymbolLength = newMessLen/k;

        %Reshape for matrix of 4 rows * numerous columns
        msg_sym_reshape = reshape(newMess,k,inputSymbolLength);

        %Decimalisation
        msg_sym = bi2de(msg_sym_reshape.','left-msb');

        %Modulation
        msg_tx = qammod(msg_sym,M);

        %Output variables
        mod_out = msg_tx;
        extbits = msgExtBitsLen;
        kbit = 4;

        %Scatterplot
        %h1 = scatterplot(msg_tx);
        %axis([-5 5 -5 5]); % Set axis ranges.

    case 5  %64QAM Modulation
```

```
%Modulation parrameter
M=64;

%Number of bits per symbol eg. 6 bits per symbol
k = log2(M);

%Padding data length with extra '1's for multiple
%of symbol size
input_sym_len = inModLengthOrig/k;
sym_len_rnd  = ceil(input_sym_len);
inputLength = sym_len_rnd * k;
msgExtBitsLen = inputLength - inModLengthOrig;
msgExtBits = ones(1,msgExtBitsLen);
newMess = [mod_input msgExtBits];
newMessLen = length(newMess);
newMess = newMess';

%symbol length
inputSymbolLength = newMessLen/k;

%Reshape for matrix of 6 rows * numerous columns
msg_sym_reshape = reshape(newMess,k,inputSymbolLength);

%Decimalisation
msg_sym = bi2de(msg_sym_reshape.','left-msb');

%Modulation
msg_tx = qammod(msg_sym,M);

%Output variables
mod_out = msg_tx;
extbits = msgExtBitsLen;
kbit = 6;
```

```
        %Scatterplot
        %h1 = scatterplot(msg_tx);
        %axis([-9 9 -9 9]); % Set axis ranges.

end
```

B.12 ofdm_ifft

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ifft_out, ifftextbits] = ofdm_ifft(ifft_in,handles)

        %Assigning variables
        ifft_input = ifft_in;
        ifftLenOrig = length(ifft_input);
        noSubCar = handles.noSubCar;
        noDatCar = handles.noDataCar;
        noPilCar = handles.noPilotCar;
        noPadCar = handles.noPadCar;
        noCeCar  = handles.noCeCar;
        noTotCar = handles.totSubChan;
        scsv     = handles.subCarSizVal;


        %Calling required IFFT function
switch  scsv
```

```
case 1  %IFFT64

    %IFFT function call
    [ifft64output,...
        iff64ext] = ofdm_ifft64(ifft_input,handles);

    %Output variables
    ifft_out = ifft64output;
    ifftextbits = iff64ext;

case 2  %IFFT256

    %IFFT function call
    [ifft256output,...
        iff256ext] = ofdm_ifft256(ifft_input,handles);

    %Output variables
    ifft_out = ifft256output;
    ifftextbits = iff256ext;

case 3  %IFFT512

    %IFFT function call
    [ifft512output,...
        iff512ext] = ofdm_ifft512(ifft_input,handles);

    %Output variables
    ifft_out = ifft512output;
    ifftextbits = iff512ext;

case 4  %IFFT1024

    %IFFT function call
```

```
        [ifft1024output,...
            iff1024ext] = ofdm_ifft1024(ifft_input,handles);

        %Output variables
        ifft_out = ifft1024output;
        ifftextbits = iff1024ext;

    case 5  %IFFT2048

        %IFFT function call
        [ifft2048output,...
            iff2048ext] = ofdm_ifft2048(ifft_input,handles);

        %Output variables
        ifft_out = ifft2048output;
        ifftextbits = iff2048ext;

    case 6  %IFFT4096

        %IFFT function call
        [ifft4096output,...
            iff4096ext] = ofdm_ifft4096(ifft_input,handles);

        %Output variables
        ifft_out = ifft4096output;
        ifftextbits = iff4096ext;

    case 7  %IFFT8192

        %IFFT function call
        [ifft8192output,...
            iff8192ext] = ofdm_ifft8192(ifft_input,handles);
```

```
        %Output variables
        ifft_out = ifft8192output;
        ifftextbits = iff8192ext;

end
```

B.13 ofdm_ifft64

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% IFFT64
function [ifft64out, ifft64extbits] = ofdm_ifft64(ifft64in,handles)

        %Assigning variables
        ifft_input = ifft64in;
        ifftLenOrig = length(ifft_input);
        noSubCar = handles.noSubCar;
        noDatCar = handles.noDataCar;
        noPilCar = handles.noPilotCar;
        noPadCar = handles.noPadCar;
        noCeCar  = handles.noCeCar;
        noTotCar = handles.totSubChan;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add pilots, DC zero and zero pad the Symbols

        %Padding length with '1's for mulitple of data sub-carriers
        sym_len_col = (length(ifft_input)/noDatCar);
```

```
sym_len_rnd  = ceil(sym_len_col);
inpifftLen = sym_len_rnd * noDatCar;
ifftExtBitsLen = inpifftLen - ifftLenOrig;
ifftExtBits = ones(1,ifftExtBitsLen);
ifft_input = ifft_input';
newifftMess = [ifft_input ifftExtBits];
newifftLen = length(newifftMess);
newifftMess = newifftMess';

%Reshape to Number of data sub-carriers
ifftSym = reshape(newifftMess,noDatCar,sym_len_rnd);

%Assigning data to data sub-carriers
ifftSymPil([7:11,13:25,27:32,34:39,41:53,...
    55:59],:) = ifftSym([1:5,6:18,19:24,25:30,31:43,44:48],:);

%Assigning zero pad to padding sub-carriers
%(Nc = 1 to 6, 60 to 64)
%Assigning zero pad to DC sub-carrier
%(Nc = 33)
%Assigning pilots to pilot sub-carriers
%(Nc = 12, 26, 40, 54)
ifftSymPil([1:6],:)   =  0;
ifftSymPil([12],:)    =  1;
ifftSymPil([26],:)    =  1;
ifftSymPil([33],:)    =  0;
ifftSymPil([40],:)    =  1;
ifftSymPil([54],:)    = -1;
ifftSymPil([60:64],:) =  0;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% IFFT OFDM symbols
```

```
        %IFFT function
        ifftSymOut = ifft(ifftSymPil);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add Cyclic prefix

        %Add cyclic extension
        ifftSymPrefix([1:16,...
            17:80],:) = ifftSymOut([49:64,1:64],:);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Output

        %Parallel OFDM symbols
        ifft_outa = ifftSymPrefix;

        %Serial OFDM symbols
        ifft_outSer = reshape(ifft_outa,1,[]);

        %Output variables
        ifft64out = ifft_outSer;
        ifft64extbits = ifftExtBitsLen;
```

B.14 ofdm_ifft256

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
```

```
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% IFFT256
function [ifft_out, ifftextbits] = ofdm_ifft256(ifft_in,handles)

        %Assigning variables
        ifft_input = ifft_in;
        ifftLenOrig = length(ifft_input);
        noSubCar = handles.noSubCar;
        noDatCar = handles.noDataCar;
        noPilCar = handles.noPilotCar;
        noPadCar = handles.noPadCar;
        noCeCar  = handles.noCeCar;
        noTotCar = handles.totSubChan;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add pilots, DC zero and zero pad the Symbols

        %Padding length with '1's for mulitple of data sub-carriers
        sym_len_col = (length(ifft_input)/noDatCar);
        sym_len_rnd  = ceil(sym_len_col);
        inpifftLen = sym_len_rnd * noDatCar;
        ifftExtBitsLen = inpifftLen - ifftLenOrig;
        ifftExtBits = ones(1,ifftExtBitsLen);
        ifft_input = ifft_input';
        newifftMess = [ifft_input ifftExtBits];
        newifftLen = length(newifftMess);
        newifftMess = newifftMess';

        %Reshape to Number of data sub-carriers
        ifftSym = reshape(newifftMess,noDatCar,sym_len_rnd);

        %Assigning data to data sub-carriers
```

```
ifftSymPil([25:30,32:43,45:56,58:69,71:82,84:95,97:108,...
    110:121,123:128,130:135,137:148,150:161,163:174,...
    176:187,189:200,202:213,215:226,228:233],:)...
    = ifftSym([1:6,7:18,19:30,31:42,43:54,55:66,...
    67:78,79:90,91:96,97:102,103:114,115:126,127:138,...
    139:150,151:162,163:174,175:186,187:192],:);

%Assigning zero pad to padding sub-carriers
%Assigning zero pad to DC sub-carrier
%Assigning pilots to pilot sub-carriers

ifftSymPil([1:24],:)    =  0;
ifftSymPil([31],:)      =  1;
ifftSymPil([44],:)      =  1;
ifftSymPil([57],:)      =  1;
ifftSymPil([70],:)      =  1;
ifftSymPil([83],:)      =  1;
ifftSymPil([96],:)      =  1;
ifftSymPil([109],:)     =  1;
ifftSymPil([122],:)     =  1;
ifftSymPil([129],:)     =  0;
ifftSymPil([136],:)     =  1;
ifftSymPil([149],:)     =  1;
ifftSymPil([162],:)     =  1;
ifftSymPil([175],:)     =  1;
ifftSymPil([188],:)     =  1;
ifftSymPil([201],:)     =  1;
ifftSymPil([214],:)     =  1;
ifftSymPil([227],:)     = -1;
ifftSymPil([234:256],:) =  0;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% IFFT OFDM symbols

        %IFFT function
        ifftSymOut = ifft(ifftSymPil);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add Cyclic prefix

        %Add cyclic extension
        ifftSymPrefix([1:64,...
            65:320],:) = ifftSymOut([193:256,1:256],:);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Output

        %Parallel OFDM symbols
        ifft_outa = ifftSymPrefix;

        %Serial OFDM symbols
        ifft_outSer = reshape(ifft_outa,1,[]);

        %Output variables
        ifft_out = ifft_outSer;
        ifftextbits = ifftExtBitsLen;
```

B.15 ofdm_ifft512

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%                  Project - OFDM Simulator
```

```
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% IFFT512
function [ifft_out, ifftextbits] = ofdm_ifft512(ifft_in,handles)

        %Assigning variables
        ifft_input = ifft_in;
        ifftLenOrig = length(ifft_input);
        noSubCar = handles.noSubCar;
        noDatCar = handles.noDataCar;
        noPilCar = handles.noPilotCar;
        noPadCar = handles.noPadCar;
        noCeCar  = handles.noCeCar;
        noTotCar = handles.totSubChan;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add pilots, DC zero and zero pad the Symbols

        %Padding length with '1's for mulitple of data sub-carriers
        sym_len_col = (length(ifft_input)/noDatCar);
        sym_len_rnd  = ceil(sym_len_col);
        inpifftLen = sym_len_rnd * noDatCar;
        ifftExtBitsLen = inpifftLen - ifftLenOrig;
        ifftExtBits = ones(1,ifftExtBitsLen);
        ifft_input = ifft_input';
        newifftMess = [ifft_input ifftExtBits];
        newifftLen = length(newifftMess);
        newifftMess = newifftMess';

        %Reshape to Number of data sub-carriers
        ifftSym = reshape(newifftMess,noDatCar,sym_len_rnd);
```

```
%Assigning data to data sub-carriers
ifftSymPil([49:54,56:67,69:80,82:93,95:106,108:119,121:132,...
    134:145,147:158,160:171,173:184,186:197,199:210,212:223,...
    225:236,238:249,251:256,258:263,265:276,278:289,291:302,...
    304:315,317:328,330:341,343:354,356:367,369:380,382:393,...
    395:406,408:419,421:432,434:445,447:458,460:465],:)...
    = ifftSym([1:6,7:18,19:30,31:42,43:54,55:66,67:78,79:90,...
    91:102,103:114,115:126,127:138,139:150,151:162,163:174,...
    175:186,187:192,193:198,199:210,211:222,223:234,235:246,...
    247:258,259:270,271:282,283:294,295:306,307:318,319:330,...
    331:342,343:354,355:366,367:378,379:384],:);

%Assigning zero pad to padding sub-carriers
%Assigning zero pad to DC sub-carrier
%Assigning pilots to pilot sub-carriers

ifftSymPil([1:48],:)    =  0;

ifftSymPil([55,68,81,94,107,...
    120,133,146,159,172,185,198,211,224,237,250,264,277,...
    290,303,316,329,342,355,368,381,394,407,420,433,...
    446],:)     =  1;
ifftSymPil([257],:)     =  0;
ifftSymPil([459],:)     = -1;
ifftSymPil([466:512],:) =  0;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% IFFT OFDM symbols

%IFFT function
ifftSymOut = ifft(ifftSymPil);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add Cyclic prefix

        %Add cyclic extension
        ifftSymPrefix([1:128,...
            129:640],:) = ifftSymOut([385:512,1:512],:);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Output

        %Parallel OFDM symbols
        ifft_outa = ifftSymPrefix;

        %Serial OFDM symbols
        ifft_outSer = reshape(ifft_outa,1,[]);

        %Output variables
        ifft_out = ifft_outSer;
        ifftextbits = ifftExtBitsLen;
```

B.16 ofdm_ifft1024

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% IFFT1024
function [ifft_out, ifftextbits] = ofdm_ifft1024(ifft_in,handles)
```

```
%Assigning variables
ifft_input = ifft_in;
ifftLenOrig = length(ifft_input);
noSubCar = handles.noSubCar;
noDatCar = handles.noDataCar;
noPilCar = handles.noPilotCar;
noPadCar = handles.noPadCar;
noCeCar  = handles.noCeCar;
noTotCar = handles.totSubChan;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add pilots, DC zero and zero pad the Symbols

%Padding length with '1's for mulitple of data sub-carriers
sym_len_col = (length(ifft_input)/noDatCar);
sym_len_rnd  = ceil(sym_len_col);
inpifftLen = sym_len_rnd * noDatCar;
ifftExtBitsLen = inpifftLen - ifftLenOrig;
ifftExtBits = ones(1,ifftExtBitsLen);
ifft_input = ifft_input';
newifftMess = [ifft_input ifftExtBits];
newifftLen = length(newifftMess);
newifftMess = newifftMess';

%Reshape to Number of data sub-carriers
ifftSym = reshape(newifftMess,noDatCar,sym_len_rnd);

%Assigning data to data sub-carriers
ifftSymPil([97:102,104:115,117:128,130:141,143:154,156:167,...
    169:180,182:193,195:206,208:219,221:232,234:245,247:258,...
    260:271,273:284,286:297,299:310,312:323,325:336,338:349,...
    351:362,364:375,377:388,390:401,403:414,416:427,429:440,...
```

```
    442:453,455:466,468:479,481:492,494:505,507:512,514:519,...
    521:532,534:545,547:558,560:571,573:584,586:597,599:610,...
    612:623,625:636,638:649,651:662,664:675,677:688,690:701,...
    703:714,716:727,729:740,742:753,755:766,768:779,781:792,...
    794:805,807:818,820:831,833:844,846:857,859:870,872:883,...
    885:896,898:909,911:922,924:929],:)...
   = ifftSym([1:6,7:18,19:30,31:42,43:54,55:66,67:78,79:90,...
    91:102,103:114,115:126,127:138,139:150,151:162,163:174,...
    175:186,187:198,199:210,211:222,223:234,235:246,247:258,...
    259:270,271:282,283:294,295:306,307:318,319:330,331:342,...
    343:354,355:366,367:378,379:384,385:390,391:402,403:414,...
    415:426,427:438,439:450,451:462,463:474,475:486,487:498,...
    499:510,511:522,523:534,535:546,547:558,559:570,571:582,...
    583:594,595:606,607:618,619:630,631:642,643:654,655:666,...
    667:678,679:690,691:702,703:714,715:726,727:738,739:750,...
    751:762,763:768],:);



%Assigning zero pad to padding sub-carriers
%Assigning zero pad to DC sub-carrier
%Assigning pilots to pilot sub-carriers

ifftSymPil([1:96],:)      =  0;
ifftSymPil([103,116,129,142,155,168,181,194,207,220,233,...
    246,259,272,285,298,311,324,337,350,363,376,389,402,...
    415,428,441,454,467,480,493,506,520,533,546,559,572,...
    585,598,611,624,637,650,663,676,689,702,715,728,741,...
    754,767,780,793,806,819,832,845,858,871,884,...
    897,910,],:)      =  1;

ifftSymPil([513],:)       =  0;
ifftSymPil([923],:)       = -1;
```

```
        ifftSymPil([930:1024],:) =  0;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% IFFT OFDM symbols

        %IFFT function
        ifftSymOut = ifft(ifftSymPil);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add Cyclic prefix

        %Add cyclic extension
        ifftSymPrefix([1:256,...
            257:1280],:) = ifftSymOut([769:1024,1:1024],:);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Output

        %Parallel OFDM symbols
        ifft_outa = ifftSymPrefix;

        %Serial OFDM symbols
        ifft_outSer = reshape(ifft_outa,1,[]);

        %Output variables
        ifft_out = ifft_outSer;
        ifftextbits = ifftExtBitsLen;
```

B.17 ofdm_ifft2048

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% IFFT2048
function [ifft_out, ifftextbits] = ofdm_ifft2048(ifft_in,handles)

        %Assigning variables
        ifft_input = ifft_in;
        ifftLenOrig = length(ifft_input);
        noSubCar = handles.noSubCar;
        noDatCar = handles.noDataCar;
        noPilCar = handles.noPilotCar;
        noPadCar = handles.noPadCar;
        noCeCar  = handles.noCeCar;
        noTotCar = handles.totSubChan;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add pilots, DC zero and zero pad the Symbols

        %Padding length with '1's for mulitple of data sub-carriers
        sym_len_col = (length(ifft_input)/noDatCar);
        sym_len_rnd  = ceil(sym_len_col);
        inpifftLen = sym_len_rnd * noDatCar;
        ifftExtBitsLen = inpifftLen - ifftLenOrig;
        ifftExtBits = ones(1,ifftExtBitsLen);
        ifft_input = ifft_input';
        newifftMess = [ifft_input ifftExtBits];
        newifftLen = length(newifftMess);
        newifftMess = newifftMess';
```

```
%Reshape to Number of data sub-carriers
ifftSym = reshape(newifftMess,noDatCar,sym_len_rnd);


%Assigning data to data sub-carriers
ifftSymPil([193:198,200:211,213:224,226:237,239:250,252:263,...
    265:276,278:289,291:302,304:315,317:328,330:341,343:354,...
    356:367,369:380,382:393,395:406,408:419,421:432,434:445,...
    447:458,460:471,473:484,486:497,499:510,512:523,525:536,...
    538:549,551:562,564:575,577:588,590:601,603:614,616:627,...
    629:640,642:653,655:666,668:679,681:692,694:705,707:718,...
    720:731,733:744,746:757,759:770,772:783,785:796,798:809,...
    811:822,824:835,837:848,850:861,863:874,876:887,889:900,...
    902:913,915:926,928:939,941:952,954:965,967:978,980:991,...
    993:1004,1006:1017,1019:1024,1026:1031,1033:1044,...
    1046:1057,1059:1070,1072:1083,1085:1096,1098:1109,...
    1111:1122,1124:1135,1137:1148,1150:1161,1163:1174,...
    1176:1187,1189:1200,1202:1213,1215:1226,1228:1239,...
    1241:1252,1254:1265,1267:1278,1280:1291,1293:1304,...
    1306:1317,1319:1330,1332:1343,1345:1356,1358:1369,...
    1371:1382,1384:1395,1397:1408,1410:1421,1423:1434,...
    1436:1447,1449:1460,1462:1473,1475:1486,1488:1499,...
    1501:1512,1514:1525,1527:1538,1540:1551,1553:1564,...
    1566:1577,1579:1590,1592:1603,1605:1616,1618:1629,...
    1631:1642,1644:1655,1657:1668,1670:1681,1683:1694,...
    1696:1707,1709:1720,1722:1733,1735:1746,1748:1759,...
    1761:1772,1774:1785,1787:1798,1800:1811,1813:1824,...
    1826:1837,1839:1850,1852:1857],:)...
   = ifftSym([1:6,7:18,19:30,31:42,43:54,55:66,67:78,79:90,...
    91:102,103:114,115:126,127:138,139:150,151:162,163:174,...
    175:186,187:198,199:210,211:222,223:234,235:246,247:258,...
    259:270,271:282,283:294,295:306,307:318,319:330,331:342,...
```

```
    343:354,355:366,367:378,379:384,385:390,391:402,403:414,...
    415:426,427:438,439:450,451:462,463:474,475:486,487:498,...
    499:510,511:522,523:534,535:546,547:558,559:570,571:582,...
    583:594,595:606,607:618,619:630,631:642,643:654,655:666,...
    667:678,679:690,691:702,703:714,715:726,727:738,739:750,...
    751:762,763:768,769:774,775:786,787:798,799:810,811:822,...
    823:834,835:846,847:858,859:870,871:882,883:894,895:906,...
    907:918,919:930,931:942,943:954,955:966,967:978,979:990,...
    991:1002,1003:1014,1015:1026,1027:1038,1039:1050,...
    1051:1062,1063:1074,1075:1086,1087:1098,1099:1110,...
    1111:1122,1123:1134,1135:1146,1147:1158,1159:1170,...
    1171:1182,1183:1194,1195:1206,1207:1218,1219:1230,...
    1231:1242,1243:1254,1255:1266,1267:1278,1279:1290,...
    1291:1302,1303:1314,1315:1326,1327:1338,1339:1350,...
    1351:1362,1363:1374,1375:1386,1387:1398,1399:1410,...
    1411:1422,1423:1434,1435:1446,1447:1458,1459:1470,...
    1471:1482,1483:1494,1495:1506,1507:1518,1519:1530,...
    1531:1536],:);


%Assigning zero pad to padding sub-carriers
%Assigning zero pad to DC sub-carrier
%Assigning pilots to pilot sub-carriers

ifftSymPil([1:192],:)    =  0;
ifftSymPil([1025],:)     =  0;
ifftSymPil([1858:2048],:) =  0;


ifftSymPil([199,212,225,238,251,264,277,290,303,316,329,...
    342,355,368,381,394,407,420,433,446,459,472,485,498,...
    511,524,537,550,563,576,589,602,615,628,641,654,667,...
    680,693,706,719,732,745,758,771,784,797,810,823,836,...
```

```
    849,862,875,888,901,914,927,940,953,966,979,992,1005,...
    1018,1032,1045,1058,1071,1084,1097,1110,1123,1136,...
    1149,1162,1175,1188,1201,1214,1227,1240,1253,1266,...
    1279,1292,1305,1318,1331,1344,1357,1370,1383,1396,...
    1409,1422,1435,1448,1461,1474,1487,1500,1513,1526,...
    1539,1552,1565,1578,1591,1604,1617,1630,1643,1656,...
    1669,1682,1695,1708,1721,1734,1747,1760,1773,1786,...
    1799,1812,1825,1838],:)     =  1;

ifftSymPil([1851],:)       = -1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% IFFT OFDM symbols

    %IFFT function
    ifftSymOut = ifft(ifftSymPil);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add Cyclic prefix

    %Add cyclic extension
    ifftSymPrefix([1:512,...
        513:2560],:) = ifftSymOut([1537:2048,1:2048],:);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Output

    %Parallel OFDM symbols
    ifft_outa = ifftSymPrefix;

    %Serial OFDM symbols
```

```
ifft_outSer = reshape(ifft_outa,1,[]);

%Output variables
ifft_out = ifft_outSer;
ifftextbits = ifftExtBitsLen;
```

B.18 ofdm_ifft4096

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% IFFT4096
function [ifft_out, ifftextbits] = ofdm_ifft4096(ifft_in,handles)

%Assigning variables
ifft_input = ifft_in;
ifftLenOrig = length(ifft_input);
noSubCar = handles.noSubCar;
noDatCar = handles.noDataCar;
noPilCar = handles.noPilotCar;
noPadCar = handles.noPadCar;
noCeCar  = handles.noCeCar;
noTotCar = handles.totSubChan;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add pilots, DC zero and zero pad the Symbols

%Padding length with '1's for mulitple of data sub-carriers
sym_len_col = (length(ifft_input)/noDatCar);
```

```
sym_len_rnd  = ceil(sym_len_col);
inpifftLen = sym_len_rnd * noDatCar;
ifftExtBitsLen = inpifftLen - ifftLenOrig;
ifftExtBits = ones(1,ifftExtBitsLen);
ifft_input = ifft_input';
newifftMess = [ifft_input ifftExtBits];
newifftLen = length(newifftMess);
newifftMess = newifftMess';

%Reshape to Number of data sub-carriers
ifftSym = reshape(newifftMess,noDatCar,sym_len_rnd);

%Assigning data to data sub-carriers
ifftSymPil([385:390,392:403,405:416,418:429,431:442,444:455,...
    457:468,470:481,483:494,496:507,509:520,522:533,535:546,...
    548:559,561:572,574:585,587:598,600:611,613:624,626:637,...
    639:650,652:663,665:676,678:689,691:702,704:715,717:728,...
    730:741,743:754,756:767,769:780,782:793,795:806,808:819,...
    821:832,834:845,847:858,860:871,873:884,886:897,899:910,...
    912:923,925:936,938:949,951:962,964:975,977:988,990:1001,...
    1003:1014,1016:1027,1029:1040,1042:1053,1055:1066,...
    1068:1079,1081:1092,1094:1105,1107:1118,1120:1131,...
    1133:1144,1146:1157,1159:1170,1172:1183,1185:1196,...
    1198:1209,1211:1222,1224:1235,1237:1248,1250:1261,...
    1263:1274,1276:1287,1289:1300,1302:1313,1315:1326,...
    1328:1339,1341:1352,1354:1365,1367:1378,1380:1391,...
    1393:1404,1406:1417,1419:1430,1432:1443,1445:1456,...
    1458:1469,1471:1482,1484:1495,1497:1508,1510:1521,...
    1523:1534,1536:1547,1549:1560,1562:1573,1575:1586,...
    1588:1599,1601:1612,1614:1625,1627:1638,1640:1651,...
    1653:1664,1666:1677,1679:1690,1692:1703,1705:1716,...
    1718:1729,1731:1742,1744:1755,1757:1768,1770:1781,...
    1783:1794,1796:1807,1809:1820,1822:1833,1835:1846,...
```

```
    1848:1859,1861:1872,1874:1885,1887:1898,1900:1911,...
    1913:1924,1926:1937,1939:1950,1952:1963,1965:1976,...
    1978:1989,1991:2002,2004:2015,2017:2028,2030:2041,...
    2043:2048,2050:2055,2057:2068,2070:2081,2083:2094,...
    2096:2107,2109:2120,2122:2133,2135:2146,2148:2159,...
    2161:2172,2174:2185,2187:2198,2200:2211,2213:2224,...
    2226:2237,2239:2250,2252:2263,2265:2276,2278:2289,...
    2291:2302,2304:2315,2317:2328,2330:2341,2343:2354,...
    2356:2367,2369:2380,2382:2393,2395:2406,2408:2419,...
    2421:2432,2434:2445,2447:2458,2460:2471,2473:2484,...
    2486:2497,2499:2510,2512:2523,2525:2536,2538:2549,...
    2551:2562,2564:2575,2577:2588,2590:2601,2603:2614,...
    2616:2627,2629:2640,2642:2653,2655:2666,2668:2679,...
    2681:2692,2694:2705,2707:2718,2720:2731,2733:2744,...
    2746:2757,2759:2770,2772:2783,2785:2796,2798:2809,...
    2811:2822,2824:2835,2837:2848,2850:2861,2863:2874,...
    2876:2887,2889:2900,2902:2913,2915:2926,2928:2939,...
    2941:2952,2954:2965,2967:2978,2980:2991,2993:3004,...
    3006:3017,3019:3030,3032:3043,3045:3056,3058:3069,...
    3071:3082,3084:3095,3097:3108,3110:3121,3123:3134,...
    3136:3147,3149:3160,3162:3173,3175:3186,3188:3199,...
    3201:3212,3214:3225,3227:3238,3240:3251,3253:3264,...
    3266:3277,3279:3290,3292:3303,3305:3316,3318:3329,...
    3331:3342,3344:3355,3357:3368,3370:3381,3383:3394,...
    3396:3407,3409:3420,3422:3433,3435:3446,3448:3459,...
    3461:3472,3474:3485,3487:3498,3500:3511,3513:3524,...
    3526:3537,3539:3550,3552:3563,3565:3576,3578:3589,...
    3591:3602,3604:3615,3617:3628,3630:3641,3643:3654,...
    3656:3667,3669:3680,3682:3693,3695:3706,3708:3713],:)...
  = ifftSym([1:6,7:18,19:30,31:42,43:54,55:66,67:78,79:90,...
  91:102,103:114,115:126,127:138,139:150,151:162,163:174,...
  175:186,187:198,199:210,211:222,223:234,235:246,247:258,...
  259:270,271:282,283:294,295:306,307:318,319:330,331:342,...
```

```
343:354,355:366,367:378,379:390,391:402,403:414,415:426,...
427:438,439:450,451:462,463:474,475:486,487:498,499:510,...
511:522,523:534,535:546,547:558,559:570,571:582,583:594,...
595:606,607:618,619:630,631:642,643:654,655:666,667:678,...
679:690,691:702,703:714,715:726,727:738,739:750,751:762,...
763:774,775:786,787:798,799:810,811:822,823:834,835:846,...
847:858,859:870,871:882,883:894,895:906,907:918,919:930,...
931:942,943:954,955:966,967:978,979:990,991:1002,1003:1014,...
1015:1026,1027:1038,1039:1050,1051:1062,1063:1074,1075:1086,...
1087:1098,1099:1110,1111:1122,1123:1134,1135:1146,1147:1158,...
1159:1170,1171:1182,1183:1194,1195:1206,1207:1218,1219:1230,...
1231:1242,1243:1254,1255:1266,1267:1278,1279:1290,1291:1302,...
1303:1314,1315:1326,1327:1338,1339:1350,1351:1362,1363:1374,...
1375:1386,1387:1398,1399:1410,1411:1422,1423:1434,1435:1446,...
1447:1458,1459:1470,1471:1482,1483:1494,1495:1506,1507:1518,...
1519:1530,1531:1536,1537:1542,1543:1554,1555:1566,1567:1578,...
1579:1590,1591:1602,1603:1614,1615:1626,1627:1638,1639:1650,...
1651:1662,1663:1674,1675:1686,1687:1698,1699:1710,1711:1722,...
1723:1734,1735:1746,1747:1758,1759:1770,1771:1782,1783:1794,...
1795:1806,1807:1818,1819:1830,1831:1842,1843:1854,1855:1866,...
1867:1878,1879:1890,1891:1902,1903:1914,1915:1926,1927:1938,...
1939:1950,1951:1962,1963:1974,1975:1986,1987:1998,1999:2010,...
2011:2022,2023:2034,2035:2046,2047:2058,2059:2070,2071:2082,...
2083:2094,2095:2106,2107:2118,2119:2130,2131:2142,2143:2154,...
2155:2166,2167:2178,2179:2190,2191:2202,2203:2214,2215:2226,...
2227:2238,2239:2250,2251:2262,2263:2274,2275:2286,2287:2298,...
2299:2310,2311:2322,2323:2334,2335:2346,2347:2358,2359:2370,...
2371:2382,2383:2394,2395:2406,2407:2418,2419:2430,2431:2442,...
2443:2454,2455:2466,2467:2478,2479:2490,2491:2502,2503:2514,...
2515:2526,2527:2538,2539:2550,2551:2562,2563:2574,2575:2586,...
2587:2598,2599:2610,2611:2622,2623:2634,2635:2646,2647:2658,...
2659:2670,2671:2682,2683:2694,2695:2706,2707:2718,2719:2730,...
2731:2742,2743:2754,2755:2766,2767:2778,2779:2790,2791:2802,...
```

```
2803:2814,2815:2826,2827:2838,2839:2850,2851:2862,2863:2874,...
2875:2886,2887:2898,2899:2910,2911:2922,2923:2934,2935:2946,...
2947:2958,2959:2970,2971:2982,2983:2994,2995:3006,3007:3018,...
3019:3030,3031:3042,3043:3054,3055:3066,3067:3072],:);


%Assigning zero pad to padding sub-carriers
%Assigning zero pad to DC sub-carrier
%Assigning pilots to pilot sub-carriers

ifftSymPil([1:384],:)    =  0;
ifftSymPil([2049],:)     =  0;
ifftSymPil([3714:4096],:) =  0;


ifftSymPil([391,404,417,430,443,456,469,482,495,508,521,534,...
    547,560,573,586,599,612,625,638,651,664,677,690,703,716,...
    729,742,755,768,781,794,807,820,833,846,859,872,885,898,...
    911,924,937,950,963,976,989,1002,1015,1028,1041,1054,1067,...
    1080,1093,1106,1119,1132,1145,1158,1171,1184,1197,1210,...
    1223,1236,1249,1262,1275,1288,1301,1314,1327,1340,1353,...
    1366,1379,1392,1405,1418,1431,1444,1457,1470,1483,1496,...
    1509,1522,1535,1548,1561,1574,1587,1600,1613,1626,1639,...
    1652,1665,1678,1691,1704,1717,1730,1743,1756,1769,1782,...
    1795,1808,1821,1834,1847,1860,1873,1886,1899,1912,1925,...
    1938,1951,1964,1977,1990,2003,2016,2029,2042,2056,2069,...
    2082,2095,2108,2121,2134,2147,2160,2173,2186,2199,2212,...
    2225,2238,2251,2264,2277,2290,2303,2316,2329,2342,2355,...
    2368,2381,2394,2407,2420,2433,2446,2459,2472,2485,2498,...
    2511,2524,2537,2550,2563,2576,2589,2602,2615,2628,2641,...
    2654,2667,2680,2693,2706,2719,2732,2745,2758,2771,2784,...
    2797,2810,2823,2836,2849,2862,2875,2888,2901,2914,2927,...
    2940,2953,2966,2979,2992,3005,3018,3031,3044,3057,3070,...
```

```
        3083,3096,3109,3122,3135,3148,3161,3174,3187,3200,3213,...
        3226,3239,3252,3265,3278,3291,3304,3317,3330,3343,3356,...
        3369,3382,3395,3408,3421,3434,3447,3460,3473,3486,3499,...
        3512,3525,3538,3551,3564,3577,3590,3603,3616,3629,3642,...
        3655,3668,3681,3694,3707],:)      =  1;

    ifftSymPil([3707],:)     = -1;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% IFFT OFDM symbols

    %IFFT function
    ifftSymOut = ifft(ifftSymPil);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add Cyclic prefix

    %Add cyclic extension
    ifftSymPrefix([1:1024,...
        1025:5120],:) = ifftSymOut([3073:4096,1:4096],:);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Output

    %Parallel OFDM symbols
    ifft_outa = ifftSymPrefix;

    %Serial OFDM symbols
    ifft_outSer = reshape(ifft_outa,1,[]);
```

```
%Output variables
ifft_out = ifft_outSer;
ifftextbits = ifftExtBitsLen;
```

B.19 ofdm_ifft8192

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% IFFT8192
function [ifft_out, ifftextbits] = ofdm_ifft8192(ifft_in,handles)

        %Assigning variables
        ifft_input = ifft_in;
        ifftLenOrig = length(ifft_input);
        noSubCar = handles.noSubCar;
        noDatCar = handles.noDataCar;
        noPilCar = handles.noPilotCar;
        noPadCar = handles.noPadCar;
        noCeCar  = handles.noCeCar;
        noTotCar = handles.totSubChan;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add pilots, DC zero and zero pad the Symbols

        %Padding length with '1's for mulitple of data sub-carriers
        sym_len_col = (length(ifft_input)/noDatCar);
        sym_len_rnd  = ceil(sym_len_col);
```

```
inpifftLen = sym_len_rnd * noDatCar;
ifftExtBitsLen = inpifftLen - ifftLenOrig;
ifftExtBits = ones(1,ifftExtBitsLen);
ifft_input = ifft_input';
newifftMess = [ifft_input ifftExtBits];
newifftLen = length(newifftMess);
newifftMess = newifftMess';

%Reshape to Number of data sub-carriers
ifftSym = reshape(newifftMess,noDatCar,sym_len_rnd);

%Assigning data to data sub-carriers
ifftSymPil([769:774,776:787,789:800,802:813,815:826,828:839,...
    841:852,854:865,867:878,880:891,893:904,906:917,919:930,...
    932:943,945:956,958:969,971:982,984:995,997:1008,...
    1010:1021,1023:1034,1036:1047,1049:1060,1062:1073,...
    1075:1086,1088:1099,1101:1112,1114:1125,1127:1138,...
    1140:1151,1153:1164,1166:1177,1179:1190,1192:1203,...
    1205:1216,1218:1229,1231:1242,1244:1255,1257:1268,...
    1270:1281,1283:1294,1296:1307,1309:1320,1322:1333,...
    1335:1346,1348:1359,1361:1372,1374:1385,1387:1398,...
    1400:1411,1413:1424,1426:1437,1439:1450,1452:1463,...
    1465:1476,1478:1489,1491:1502,1504:1515,1517:1528,...
    1530:1541,1543:1554,1556:1567,1569:1580,1582:1593,...
    1595:1606,1608:1619,1621:1632,1634:1645,1647:1658,...
    1660:1671,1673:1684,1686:1697,1699:1710,1712:1723,...
    1725:1736,1738:1749,1751:1762,1764:1775,1777:1788,...
    1790:1801,1803:1814,1816:1827,1829:1840,1842:1853,...
    1855:1866,1868:1879,1881:1892,1894:1905,1907:1918,...
    1920:1931,1933:1944,1946:1957,1959:1970,1972:1983,...
    1985:1996,1998:2009,2011:2022,2024:2035,2037:2048,...
    2050:2061,2063:2074,2076:2087,2089:2100,2102:2113,...
    2115:2126,2128:2139,2141:2152,2154:2165,2167:2178,...
```

```
2180:2191,2193:2204,2206:2217,2219:2230,2232:2243,...
2245:2256,2258:2269,2271:2282,2284:2295,2297:2308,...
2310:2321,2323:2334,2336:2347,2349:2360,2362:2373,...
2375:2386,2388:2399,2401:2412,2414:2425,2427:2438,...
2440:2451,2453:2464,2466:2477,2479:2490,2492:2503,...
2505:2516,2518:2529,2531:2542,2544:2555,2557:2568,...
2570:2581,2583:2594,2596:2607,2609:2620,2622:2633,...
2635:2646,2648:2659,2661:2672,2674:2685,2687:2698,...
2700:2711,2713:2724,2726:2737,2739:2750,2752:2763,...
2765:2776,2778:2789,2791:2802,2804:2815,2817:2828,...
2830:2841,2843:2854,2856:2867,2869:2880,2882:2893,...
2895:2906,2908:2919,2921:2932,2934:2945,2947:2958,...
2960:2971,2973:2984,2986:2997,2999:3010,3012:3023,...
3025:3036,3038:3049,3051:3062,3064:3075,3077:3088,...
3090:3101,3103:3114,3116:3127,3129:3140,3142:3153,...
3155:3166,3168:3179,3181:3192,3194:3205,3207:3218,...
3220:3231,3233:3244,3246:3257,3259:3270,3272:3283,...
3285:3296,3298:3309,3311:3322,3324:3335,3337:3348,...
3350:3361,3363:3374,3376:3387,3389:3400,3402:3413,...
3415:3426,3428:3439,3441:3452,3454:3465,3467:3478,...
3480:3491,3493:3504,3506:3517,3519:3530,3532:3543,...
3545:3556,3558:3569,3571:3582,3584:3595,3597:3608,...
3610:3621,3623:3634,3636:3647,3649:3660,3662:3673,...
3675:3686,3688:3699,3701:3712,3714:3725,3727:3738,...
3740:3751,3753:3764,3766:3777,3779:3790,3792:3803,...
3805:3816,3818:3829,3831:3842,3844:3855,3857:3868,...
3870:3881,3883:3894,3896:3907,3909:3920,3922:3933,...
3935:3946,3948:3959,3961:3972,3974:3985,3987:3998,...
4000:4011,4013:4024,4026:4037,4039:4050,4052:4063,...
4065:4076,4078:4089,4091:4096,4098:4103,4105:4116,...
4118:4129,4131:4142,4144:4155,4157:4168,4170:4181,...
4183:4194,4196:4207,4209:4220,4222:4233,4235:4246,...
4248:4259,4261:4272,4274:4285,4287:4298,4300:4311,...
```

```
4313:4324,4326:4337,4339:4350,4352:4363,4365:4376,...
4378:4389,4391:4402,4404:4415,4417:4428,4430:4441,...
4443:4454,4456:4467,4469:4480,4482:4493,4495:4506,...
4508:4519,4521:4532,4534:4545,4547:4558,4560:4571,...
4573:4584,4586:4597,4599:4610,4612:4623,4625:4636,...
4638:4649,4651:4662,4664:4675,4677:4688,4690:4701,...
4703:4714,4716:4727,4729:4740,4742:4753,4755:4766,...
4768:4779,4781:4792,4794:4805,4807:4818,4820:4831,...
4833:4844,4846:4857,4859:4870,4872:4883,4885:4896,...
4898:4909,4911:4922,4924:4935,4937:4948,4950:4961,...
4963:4974,4976:4987,4989:5000,5002:5013,5015:5026,...
5028:5039,5041:5052,5054:5065,5067:5078,5080:5091,...
5093:5104,5106:5117,5119:5130,5132:5143,5145:5156,...
5158:5169,5171:5182,5184:5195,5197:5208,5210:5221,...
5223:5234,5236:5247,5249:5260,5262:5273,5275:5286,...
5288:5299,5301:5312,5314:5325,5327:5338,5340:5351,...
5353:5364,5366:5377,5379:5390,5392:5403,5405:5416,...
5418:5429,5431:5442,5444:5455,5457:5468,5470:5481,...
5483:5494,5496:5507,5509:5520,5522:5533,5535:5546,...
5548:5559,5561:5572,5574:5585,5587:5598,5600:5611,...
5613:5624,5626:5637,5639:5650,5652:5663,5665:5676,...
5678:5689,5691:5702,5704:5715,5717:5728,5730:5741,...
5743:5754,5756:5767,5769:5780,5782:5793,5795:5806,...
5808:5819,5821:5832,5834:5845,5847:5858,5860:5871,...
5873:5884,5886:5897,5899:5910,5912:5923,5925:5936,...
5938:5949,5951:5962,5964:5975,5977:5988,5990:6001,...
6003:6014,6016:6027,6029:6040,6042:6053,6055:6066,...
6068:6079,6081:6092,6094:6105,6107:6118,6120:6131,...
6133:6144,6146:6157,6159:6170,6172:6183,6185:6196,...
6198:6209,6211:6222,6224:6235,6237:6248,6250:6261,...
6263:6274,6276:6287,6289:6300,6302:6313,6315:6326,...
6328:6339,6341:6352,6354:6365,6367:6378,6380:6391,...
6393:6404,6406:6417,6419:6430,6432:6443,6445:6456,...
```

```
        6458:6469,6471:6482,6484:6495,6497:6508,6510:6521,...
        6523:6534,6536:6547,6549:6560,6562:6573,6575:6586,...
        6588:6599,6601:6612,6614:6625,6627:6638,6640:6651,...
        6653:6664,6666:6677,6679:6690,6692:6703,6705:6716,...
        6718:6729,6731:6742,6744:6755,6757:6768,6770:6781,...
        6783:6794,6796:6807,6809:6820,6822:6833,6835:6846,...
        6848:6859,6861:6872,6874:6885,6887:6898,6900:6911,...
        6913:6924,6926:6937,6939:6950,6952:6963,6965:6976,...
        6978:6989,6991:7002,7004:7015,7017:7028,7030:7041,...
        7043:7054,7056:7067,7069:7080,7082:7093,7095:7106,...
        7108:7119,7121:7132,7134:7145,7147:7158,7160:7171,...
        7173:7184,7186:7197,7199:7210,7212:7223,7225:7236,...
        7238:7249,7251:7262,7264:7275,7277:7288,7290:7301,...
        7303:7314,7316:7327,7329:7340,7342:7353,7355:7366,...
        7368:7379,7381:7392,7394:7405,7407:7418,7420:7425],:)...
    = ifftSym([1:6,7:18,19:30,31:42,43:54,55:66,67:78,79:90,...
    91:102,103:114,115:126,127:138,139:150,151:162,163:174,...
    175:186,187:198,199:210,211:222,223:234,235:246,247:258,...
    259:270,271:282,283:294,295:306,307:318,319:330,331:342,...
    343:354,355:366,367:378,379:390,391:402,403:414,415:426,...
    427:438,439:450,451:462,463:474,475:486,487:498,499:510,...
    511:522,523:534,535:546,547:558,559:570,571:582,583:594,...
    595:606,607:618,619:630,631:642,643:654,655:666,667:678,...
    679:690,691:702,703:714,715:726,727:738,739:750,751:762,...
    763:774,775:786,787:798,799:810,811:822,823:834,835:846,...
    847:858,859:870,871:882,883:894,895:906,907:918,919:930,...
    931:942,943:954,955:966,967:978,979:990,991:1002,1003:1014,...
    1015:1026,1027:1038,1039:1050,1051:1062,1063:1074,1075:1086,...
    1087:1098,1099:1110,1111:1122,1123:1134,1135:1146,1147:1158,...
    1159:1170,1171:1182,1183:1194,1195:1206,1207:1218,1219:1230,...
    1231:1242,1243:1254,1255:1266,1267:1278,1279:1290,1291:1302,...
    1303:1314,1315:1326,1327:1338,1339:1350,1351:1362,1363:1374,...
    1375:1386,1387:1398,1399:1410,1411:1422,1423:1434,1435:1446,...
```

```
1447:1458,1459:1470,1471:1482,1483:1494,1495:1506,1507:1518,...
1519:1530,1531:1542,1543:1554,1555:1566,1567:1578,1579:1590,...
1591:1602,1603:1614,1615:1626,1627:1638,1639:1650,1651:1662,...
1663:1674,1675:1686,1687:1698,1699:1710,1711:1722,1723:1734,...
1735:1746,1747:1758,1759:1770,1771:1782,1783:1794,1795:1806,...
1807:1818,1819:1830,1831:1842,1843:1854,1855:1866,1867:1878,...
1879:1890,1891:1902,1903:1914,1915:1926,1927:1938,1939:1950,...
1951:1962,1963:1974,1975:1986,1987:1998,1999:2010,2011:2022,...
2023:2034,2035:2046,2047:2058,2059:2070,2071:2082,2083:2094,...
2095:2106,2107:2118,2119:2130,2131:2142,2143:2154,2155:2166,...
2167:2178,2179:2190,2191:2202,2203:2214,2215:2226,2227:2238,...
2239:2250,2251:2262,2263:2274,2275:2286,2287:2298,2299:2310,...
2311:2322,2323:2334,2335:2346,2347:2358,2359:2370,2371:2382,...
2383:2394,2395:2406,2407:2418,2419:2430,2431:2442,2443:2454,...
2455:2466,2467:2478,2479:2490,2491:2502,2503:2514,2515:2526,...
2527:2538,2539:2550,2551:2562,2563:2574,2575:2586,2587:2598,...
2599:2610,2611:2622,2623:2634,2635:2646,2647:2658,2659:2670,...
2671:2682,2683:2694,2695:2706,2707:2718,2719:2730,2731:2742,...
2743:2754,2755:2766,2767:2778,2779:2790,2791:2802,2803:2814,...
2815:2826,2827:2838,2839:2850,2851:2862,2863:2874,2875:2886,...
2887:2898,2899:2910,2911:2922,2923:2934,2935:2946,2947:2958,...
2959:2970,2971:2982,2983:2994,2995:3006,3007:3018,3019:3030,...
3031:3042,3043:3054,3055:3066,3067:3072,3073:3078,3079:3090,...
3091:3102,3103:3114,3115:3126,3127:3138,3139:3150,3151:3162,...
3163:3174,3175:3186,3187:3198,3199:3210,3211:3222,3223:3234,...
3235:3246,3247:3258,3259:3270,3271:3282,3283:3294,3295:3306,...
3307:3318,3319:3330,3331:3342,3343:3354,3355:3366,3367:3378,...
3379:3390,3391:3402,3403:3414,3415:3426,3427:3438,3439:3450,...
3451:3462,3463:3474,3475:3486,3487:3498,3499:3510,3511:3522,...
3523:3534,3535:3546,3547:3558,3559:3570,3571:3582,3583:3594,...
3595:3606,3607:3618,3619:3630,3631:3642,3643:3654,3655:3666,...
3667:3678,3679:3690,3691:3702,3703:3714,3715:3726,3727:3738,...
3739:3750,3751:3762,3763:3774,3775:3786,3787:3798,3799:3810,...
```

```
3811:3822,3823:3834,3835:3846,3847:3858,3859:3870,3871:3882,...
3883:3894,3895:3906,3907:3918,3919:3930,3931:3942,3943:3954,...
3955:3966,3967:3978,3979:3990,3991:4002,4003:4014,4015:4026,...
4027:4038,4039:4050,4051:4062,4063:4074,4075:4086,4087:4098,...
4099:4110,4111:4122,4123:4134,4135:4146,4147:4158,4159:4170,...
4171:4182,4183:4194,4195:4206,4207:4218,4219:4230,4231:4242,...
4243:4254,4255:4266,4267:4278,4279:4290,4291:4302,4303:4314,...
4315:4326,4327:4338,4339:4350,4351:4362,4363:4374,4375:4386,...
4387:4398,4399:4410,4411:4422,4423:4434,4435:4446,4447:4458,...
4459:4470,4471:4482,4483:4494,4495:4506,4507:4518,4519:4530,...
4531:4542,4543:4554,4555:4566,4567:4578,4579:4590,4591:4602,...
4603:4614,4615:4626,4627:4638,4639:4650,4651:4662,4663:4674,...
4675:4686,4687:4698,4699:4710,4711:4722,4723:4734,4735:4746,...
4747:4758,4759:4770,4771:4782,4783:4794,4795:4806,4807:4818,...
4819:4830,4831:4842,4843:4854,4855:4866,4867:4878,4879:4890,...
4891:4902,4903:4914,4915:4926,4927:4938,4939:4950,4951:4962,...
4963:4974,4975:4986,4987:4998,4999:5010,5011:5022,5023:5034,...
5035:5046,5047:5058,5059:5070,5071:5082,5083:5094,5095:5106,...
5107:5118,5119:5130,5131:5142,5143:5154,5155:5166,5167:5178,...
5179:5190,5191:5202,5203:5214,5215:5226,5227:5238,5239:5250,...
5251:5262,5263:5274,5275:5286,5287:5298,5299:5310,5311:5322,...
5323:5334,5335:5346,5347:5358,5359:5370,5371:5382,5383:5394,...
5395:5406,5407:5418,5419:5430,5431:5442,5443:5454,5455:5466,...
5467:5478,5479:5490,5491:5502,5503:5514,5515:5526,5527:5538,...
5539:5550,5551:5562,5563:5574,5575:5586,5587:5598,5599:5610,...
5611:5622,5623:5634,5635:5646,5647:5658,5659:5670,5671:5682,...
5683:5694,5695:5706,5707:5718,5719:5730,5731:5742,5743:5754,...
5755:5766,5767:5778,5779:5790,5791:5802,5803:5814,5815:5826,...
5827:5838,5839:5850,5851:5862,5863:5874,5875:5886,5887:5898,...
5899:5910,5911:5922,5923:5934,5935:5946,5947:5958,5959:5970,...
5971:5982,5983:5994,5995:6006,6007:6018,6019:6030,6031:6042,...
6043:6054,6055:6066,6067:6078,6079:6090,6091:6102,6103:6114,...
6115:6126,6127:6138,6139:6144],:);
```

```
%Assigning zero pad to padding sub-carriers
%Assigning zero pad to DC sub-carrier
%Assigning pilots to pilot sub-carriers

ifftSymPil([1:768],:)    =  0;
ifftSymPil([4097],:)     =  0;
ifftSymPil([7426:8192],:) =  0;

ifftSymPil([775,788,801,814,827,840,853,866,879,892,905,918,...
    931,944,957,970,983,996,1009,1022,1035,1048,1061,1074,...
    1087,1100,1113,1126,1139,1152,1165,1178,1191,1204,1217,...
    1230,1243,1256,1269,1282,1295,1308,1321,1334,1347,1360,...
    1373,1386,1399,1412,1425,1438,1451,1464,1477,1490,1503,...
    1516,1529,1542,1555,1568,1581,1594,1607,1620,1633,1646,...
    1659,1672,1685,1698,1711,1724,1737,1750,1763,1776,1789,...
    1802,1815,1828,1841,1854,1867,1880,1893,1906,1919,1932,...
    1945,1958,1971,1984,1997,2010,2023,2036,2049,2062,2075,...
    2088,2101,2114,2127,2140,2153,2166,2179,2192,2205,2218,...
    2231,2244,2257,2270,2283,2296,2309,2322,2335,2348,2361,...
    2374,2387,2400,2413,2426,2439,2452,2465,2478,2491,2504,...
    2517,2530,2543,2556,2569,2582,2595,2608,2621,2634,2647,...
    2660,2673,2686,2699,2712,2725,2738,2751,2764,2777,2790,...
    2803,2816,2829,2842,2855,2868,2881,2894,2907,2920,2933,...
    2946,2959,2972,2985,2998,3011,3024,3037,3050,3063,3076,...
    3089,3102,3115,3128,3141,3154,3167,3180,3193,3206,3219,...
    3232,3245,3258,3271,3284,3297,3310,3323,3336,3349,3362,...
    3375,3388,3401,3414,3427,3440,3453,3466,3479,3492,3505,...
    3518,3531,3544,3557,3570,3583,3596,3609,3622,3635,3648,...
    3661,3674,3687,3700,3713,3726,3739,3752,3765,3778,3791,...
    3804,3817,3830,3843,3856,3869,3882,3895,3908,3921,3934,...
    3947,3960,3973,3986,3999,4012,4025,4038,4051,4064,4077,...
```

```
    4090,4104,4117,4130,4143,4156,4169,4182,4195,4208,4221,...
    4234,4247,4260,4273,4286,4299,4312,4325,4338,4351,4364,...
    4377,4390,4403,4416,4429,4442,4455,4468,4481,4494,4507,...
    4520,4533,4546,4559,4572,4585,4598,4611,4624,4637,4650,...
    4663,4676,4689,4702,4715,4728,4741,4754,4767,4780,4793,...
    4806,4819,4832,4845,4858,4871,4884,4897,4910,4923,4936,...
    4949,4962,4975,4988,5001,5014,5027,5040,5053,5066,5079,...
    5092,5105,5118,5131,5144,5157,5170,5183,5196,5209,5222,...
    5235,5248,5261,5274,5287,5300,5313,5326,5339,5352,5365,...
    5378,5391,5404,5417,5430,5443,5456,5469,5482,5495,5508,...
    5521,5534,5547,5560,5573,5586,5599,5612,5625,5638,5651,...
    5664,5677,5690,5703,5716,5729,5742,5755,5768,5781,5794,...
    5807,5820,5833,5846,5859,5872,5885,5898,5911,5924,5937,...
    5950,5963,5976,5989,6002,6015,6028,6041,6054,6067,6080,...
    6093,6106,6119,6132,6145,6158,6171,6184,6197,6210,6223,...
    6236,6249,6262,6275,6288,6301,6314,6327,6340,6353,6366,...
    6379,6392,6405,6418,6431,6444,6457,6470,6483,6496,6509,...
    6522,6535,6548,6561,6574,6587,6600,6613,6626,6639,6652,...
    6665,6678,6691,6704,6717,6730,6743,6756,6769,6782,6795,...
    6808,6821,6834,6847,6860,6873,6886,6899,6912,6925,6938,...
    6951,6964,6977,6990,7003,7016,7029,7042,7055,7068,7081,...
    7094,7107,7120,7133,7146,7159,7172,7185,7198,7211,7224,...
    7237,7250,7263,7276,7289,7302,7315,7328,7341,7354,7367,...
    7380,7393,7406],:)      =  1;

iffSymPil([7419],:)     = -1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% IFFT OFDM symbols

    %IFFT function
```

```
        ifftSymOut = ifft(ifftSymPil);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add Cyclic prefix

        %Add cyclic extension
        ifftSymPrefix([1:2048,...
            2049:10240],:) = ifftSymOut([6145:8192,1:8192],:);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Output

        %Parallel OFDM symbols
        ifft_outa = ifftSymPrefix;

        %Serial OFDM symbols
        ifft_outSer = reshape(ifft_outa,1,[]);

        %Output variables
        ifft_out = ifft_outSer;
        ifftextbits = ifftExtBitsLen;
```

B.20 ofdm_pream

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%              Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% Preamble Generator
function [preamble_out] = ofdm_pream()


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Short Preamble

        %Allocating the 12 sub-carriers
        pshort = (sqrt(13/6))*[0,0,0,0,0,0,0,0,1+j,0,0,0,-1-j,0,0, ...
            0,1+j,0,0,0,-1-j,0,0,0,-1-j,0,0,0,1+j,0,0,0,0,0,0,0, ...
            -1-j,0,0,0,-1-j,0,0,0,1+j,0,0,0,1+j,0,0,0,1+j,0,0,0, ...
            1+j,0,0,0,0,0,0,0];
        pshorta = pshort.';

        %IFFT short preamble
        pshortifft = ifft(pshorta);

        %Taking the first 16 symbols
        p16([1:16],:) =  pshortifft([1:16],:);

        %Making 10 training symbols
        peramsh = [p16 p16 p16 p16 p16 p16 p16 p16 p16 p16];

        %Serialise
        peramblesh = reshape(peramsh,160,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Long Preamble

        %Allocating the 53 sub-carriers
        plong = [0,0,0,0,0,0,1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,1,1,-1, ...
            -1,1,1,-1,1,-1,1,1,1,1,0,1,-1,-1,1,1,-1,1,-1,1,-1,-1, ...
            -1,-1,-1,1,1,-1,-1,1,-1,1,-1,1,1,1,1,0,0,0,0,0];
```

```
        plonga = plong.';

        %IFFT short preamble
        plongifft = ifft(plonga);

        %Making cyclic extensions of the training symbols
        plongifftPrefix([1:16,17:80],:) = plongifft([49:64,1:64],:);

        %Adding the cyclic extensions
        peramblelo([1:16,17:32,33:96,97:160],:) = plongifft([33:48, ...
            49:64,1:64,1:64],:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Preamble Out

        %Combining the short and long symbols to make the preamble
        preamblefull = [peramblesh peramblelo];

        %Output variable
        preamble_out = reshape(preamblefull,1,320);
```

B.21 ofdm_chann

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Channel

function [channout] = ofdm_chann(channin,kin,Ebin,handles)
```

```
        %Assigning variables
        kc = kin;
        EbNo = Ebin;
        popupmod_sel_id = handles.fadTypeVal;

        %Calculating SNR
        snr = EbNo + 3 + 10*log10(kc);

switch  popupmod_sel_id

    case 1  %None

        %No fading channel parameters
        fadedSig = channin;

    case 2  %Rayleigh

        %Rayleigh fading channel parameters
        chan = rayleighchan;

        %Effect of fading channel
        fadedSig = filter(chan, channin);

    case 3  %Rician

        %Rician fading channel parameters
        chan = ricianchan;
        chan.KFactor = 3;

        %Effect of fading channel
        fadedSig = filter(chan, channin);
```

```
end

        %Add Gaussian noise and output
        channout = awgn(fadedSig,snr);
```

B.22 ofdm_rx

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Receiver

function [rxout] = ofdm_rx(ofin,handles);

        handles;
        ofdmrxin = ofin;
        present = handles.txpreamblevals;

%% Remove Preamble

        [preout,datout] = ofdm_rempream(ofdmrxin, handles);
%

%% FFT function call

        fftoutput = ofdm_fft(datout,handles);

        fftoutput;
        [fp,fq] = size(fftoutput);
```

```
%% Demodulator function call

        demodin = fftoutput;
        demodoutput = ofdm_demodulator(demodin,...
            handles.modTypeVal,handles.modextrabits);

        n = length(demodoutput);
        demodoutput;

%% Deinterleaver function call

        deinterlvdin = demodoutput;
        [deinterlvdout] = ofdm_deinterleaver(deinterlvdin,...
            handles.noDataCar,handles.modTypeVal,...
            handles.intextrabits,handles.intperm);

        decodemessin = deinterlvdout;

%% Decoder function call

        [decodemessout] = ofdm_decoder(decodemessin,...
            handles.codTypeVal,handles.codextrabits);

%% Output

        rxout = decodemessout;
```

B.23 ofdm_rempream

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Receiver Preamble Removal
function [preout, datout] = ofdm_rempream(pre_in,handles)

        %Assigning variables
        rxIQ = pre_in;
        preamble_tx = handles.txpreamblevals;

        %Finding the size and removing preamble
        rxIQs = reshape(rxIQ,1,[]);
        [rxIQsp,rxIQsq] = size(rxIQs);
        rxIQpre(:,[1:320]) = rxIQs(:,[1:320]);
        [rxIQpp,rxIQpq] = size(rxIQpre);
        rxIQdat(:,[1:(rxIQsq-rxIQpq)]) = ...
            rxIQs(:,[(rxIQpq+1):rxIQsq]);

        [rxIQdp,rxIQdq] = size(rxIQdat);

        %Received preamble values
        rxIpream = real(rxIQpre);
        rxQpream = imag(rxIQpre);
        [rxTHETA,rxRHO] = cart2pol(rxIpream,rxQpream);

        %Known preamble values
        txIpream = real(preamble_tx);
        txQpream = imag(preamble_tx);
        [txTHETA,txRHO] = cart2pol(txIpream,txQpream);

        %Finds channel estimate through preamble comparison
        preamresrho = txRHO ./ rxRHO;
```

```
preamresthet = txTHETA - rxTHETA;

%Adjusts for channel estimate
preamresrhoa = preamresrho .* rxRHO;
preamrestheta = preamresthet + rxTHETA;

[rempreresX,rempreresY] = pol2cart(preamrestheta,...
    preamresrhoa);
rempreamcom = complex(rempreresX,rempreresY);

%Output variables
preout = rxIQpre;
datout = rxIQdat;
```

B.24 ofdm_fft

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%                Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT
function [fft_out] = ofdm_fft(fft_in,handles)


%Assigning variables
fft_input = fft_in;
fftLenOrig = length(fft_input);
noSubCar = handles.noSubCar;
noDatCar = handles.noDataCar;
noPilCar = handles.noPilotCar;
```

```
        noPadCar = handles.noPadCar;
        noCeCar  = handles.noCeCar;
        noTotCar = handles.totSubChan;
        fft_ModTy = handles.modTypeVal;
        noExtbits = handles.ifftextrabits;
        scsv = handles.subCarSizVal;

        %Calling required FFT function
switch  scsv

    case 1  %FFT64

        %FFT function call
        [fft64output] = ofdm_fft64(fft_input,handles);

        %Output variables
        fft_out = fft64output;

    case 2  %FFT256

        %FFT function call
        [fft256output] = ofdm_fft256(fft_input,handles);

        %Output variables
        fft_out = fft256output;

    case 3  %FFT512

        %FFT function call
        [fft512output] = ofdm_fft512(fft_input,handles);

        %Output variables
        fft_out = fft512output;
```

```
case 4  %FFT1024

    %FFT function call
    [fft1024output] = ofdm_fft1024(fft_input,handles);

    %Output variables
    fft_out = fft1024output;

case 5  %FFT2048

    %FFT function call
    [fft2048output] = ofdm_fft2048(fft_input,handles);

    %Output variables
    fft_out = fft2048output;

case 6  %FFT4096

    %FFT function call
    [fft4096output] = ofdm_fft4096(fft_input,handles);

    %Output variables
    fft_out = fft4096output;

case 7  %FFT8192

    %FFT function call
    [fft8192output] = ofdm_fft8192(fft_input,handles);

    %Output variables
    fft_out = fft8192output;
```

end

B.25 ofdm_fft64

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%                    Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT64
function [fft_out] = ofdm_fft64(fft_in ,handles)


        %Assigning variables
        fft_input = fft_in;
        fftLenOrig = length(fft_input);
        noSubCar = handles.noSubCar;
        noDatCar = handles.noDataCar;
        noPilCar = handles.noPilotCar;
        noPadCar = handles.noPadCar;
        noCeCar  = handles.noCeCar;
        noTotCar = handles.totSubChan;
        fft_ModTy = handles.modTypeVal;
        noExtbits = handles.ifftextrabits;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove Cyclic prefix

        %Reshape to make parallel, ensures Total Sub-carrier deep
        [fftInpLen,fftInpWid] = size(fft_input);
        fft_Par = reshape(fft_input,noTotCar,[]);
```

```
        %Remove cyclic extension
        fftSym([1:64],:) = fft_Par([17:noTotCar],:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT OFDM symbols

        %FFT function
        fftSymOut = fft(fftSym);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove the pilots, DC zero and zero padding of the Symbols

        %Remove Pilots, DC and zero pads
        fftSymPil([1:4],:)     = fftSymOut([12,26,40,54],:);
        fftSymZeros([1:11],:) = fftSymOut([1:6,60:64],:);
        fftSymDCzero([1],:)    = fftSymOut([33],:);

        %Remove data sub-carriers
        fftSymDat([1:48],:)    = fftSymOut([7:11,...
             13:25,27:32,34:39,...
             41:53,55:59],:);

        [fftSymLen,fftSymWid] = size(fftSymDat);
        fftDatSer = reshape(fftSymDat,(fftSymLen*fftSymWid),1);
        fftSerLen = length(fftDatSer);
        fftMessLen = fftSerLen - noExtbits;
        fftMess([1:fftMessLen],:) = fftDatSer([1:fftMessLen],:);

        %fftMessLen;
        %fftMess;
        fftMessa = reshape(fftMess,1,fftMessLen);
```

```
        if fft_ModTy == 1
        fftMessa = real(fftMessa);

        %makes negative zeros (-0) a zero (0) for biterr
        fftMessa = fftMessa.*fftMessa;
        fftMessa = round(fftMessa);

        end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Output

        %Output varaiables
        fft_out = fftMessa;
```

B.26 ofdm_fft256

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT256
function [fft_out] = ofdm_fft256(fft_in,handles)


        %Assigning variables
        fft_input = fft_in;
```

```
        fftLenOrig = length(fft_input);
        noSubCar = handles.noSubCar;
        noDatCar = handles.noDataCar;
        noPilCar = handles.noPilotCar;
        noPadCar = handles.noPadCar;
        noCeCar  = handles.noCeCar;
        noTotCar = handles.totSubChan;
        fft_ModTy = handles.modTypeVal;
        noExtbits = handles.ifftextrabits;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove Cyclic prefix

        %Reshape to make parallel, ensures Total Sub-carrier deep
        [fftInpLen,fftInpWid] = size(fft_input);
        fft_Par = reshape(fft_input,320,[]);

        %Remove cyclic extension
        fftSym([1:256],:) = fft_Par([65:320],:);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT OFDM symbols

        %FFT function
        fftSymOut = fft(fftSym);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove the pilots, DC zero and zero padding of the Symbols

        %Remove Pilots, DC and zero pads
        fftSymPil([1:16],:)     =  fftSymOut([31,44,57,70,83,96,109,...
            122,136,149,162,175,188,201,214,227],:);
        fftSymZeros([1:47],:) =  fftSymOut([1:24,234:256],:);
```

```
    fftSymDCzero([1],:)    =  fftSymOut([129],:);

    %Remove data sub-carriers
    fftSymDat([1:192],:)    =  fftSymOut([25:30,32:43,...
        45:56,58:69,71:82,84:95,97:108,110:121,123:128,...
        130:135,137:148,150:161,163:174,176:187,189:200,...
        202:213,215:226,228:233],:);

    [fftSymLen,fftSymWid] = size(fftSymDat);
    fftDatSer = reshape(fftSymDat,(fftSymLen*fftSymWid),1);
    fftSerLen = length(fftDatSer);
    %noExtbits;
    fftMessLen = fftSerLen - noExtbits;
    fftMess([1:fftMessLen],:) = fftDatSer([1:fftMessLen],:);

    %fftMessLen;
    %fftMess;
    fftMessa = reshape(fftMess,1,fftMessLen);




    if fft_ModTy == 1
    fftMessa = real(fftMessa);

    %makes negative zeros (-0) a zero (0) for biterr
    fftMessa = fftMessa.*fftMessa;
    fftMessa = round(fftMessa);

    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Output
```

```
%Output varaiables
fft_out = fftMessa;
```

B.27 ofdm_fft512

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT512
function [fft_out] = ofdm_fft512(fft_in,handles)


        %Assigning variables
        fft_input = fft_in;
        fftLenOrig = length(fft_input);
        noSubCar = handles.noSubCar;
        noDatCar = handles.noDataCar;
        noPilCar = handles.noPilotCar;
        noPadCar = handles.noPadCar;
        noCeCar  = handles.noCeCar;
        noTotCar = handles.totSubChan;
        fft_ModTy = handles.modTypeVal;
        noExtbits = handles.ifftextrabits;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove Cyclic prefix

        %Reshape to make parallel, ensures Total Sub-carrier deep
        [fftInpLen,fftInpWid] = size(fft_input);
```

```
        fft_Par = reshape(fft_input,640,[]);

        %Remove cyclic extension
        fftSym([1:512],:) = fft_Par([129:640],:);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT OFDM symbols

        %FFT function
        fftSymOut = fft(fftSym);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove the pilots, DC zero and zero padding of the Symbols

        %Remove Pilots, DC and zero pads
        fftSymPil([1:32],:)     = fftSymOut([55,68,81,94,107,...
            120,133,146,159,172,185,198,211,224,237,250,264,277,...
            290,303,316,329,342,355,368,381,394,407,420,433,446,459],:);
        fftSymZeros([1:95],:) = fftSymOut([1:48,466:512],:);
        fftSymDCzero([1],:)   = fftSymOut([257],:);

        %Remove data sub-carriers
        fftSymDat([1:384],:)   = fftSymOut([49:54,56:67,69:80,82:93,...
            95:106,108:119,121:132,134:145,147:158,160:171,173:184,...
            186:197,199:210,212:223,225:236,238:249,251:256,258:263,...
            265:276,278:289,291:302,304:315,317:328,330:341,343:354,...
            356:367,369:380,382:393,395:406,408:419,421:432,434:445,...
            447:458,460:465],:);
        [fftSymLen,fftSymWid] = size(fftSymDat);
        fftDatSer = reshape(fftSymDat,(fftSymLen*fftSymWid),1);
        fftSerLen = length(fftDatSer);
        fftMessLen = fftSerLen - noExtbits;
        fftMess([1:fftMessLen],:) = fftDatSer([1:fftMessLen],:);
```

```
    %fftMessLen;
    %fftMess;
    fftMessa = reshape(fftMess,1,fftMessLen);




    if fft_ModTy == 1
    fftMessa = real(fftMessa);

    %makes negative zeros (-0) a zero (0) for biterr
    fftMessa = fftMessa.*fftMessa;
    fftMessa = round(fftMessa);

    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Output

    %Output varaiables
    fft_out = fftMessa;
```

B.28 ofdm_fft1024

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar   0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT1024
function [fft_out] = ofdm_fft1024(fft_in,handles)
```

```
        %Assigning variables
        fft_input = fft_in;
        fftLenOrig = length(fft_input);
        noSubCar = handles.noSubCar;
        noDatCar = handles.noDataCar;
        noPilCar = handles.noPilotCar;
        noPadCar = handles.noPadCar;
        noCeCar  = handles.noCeCar;
        noTotCar = handles.totSubChan;
        fft_ModTy = handles.modTypeVal;
        noExtbits = handles.ifftextrabits;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove Cyclic prefix

        %Reshape to make parallel, ensures Total Sub-carrier deep
        [fftInpLen,fftInpWid] = size(fft_input);
        fft_Par = reshape(fft_input,1280,[]);

        %Remove cyclic extension
        fftSym([1:1024],:) = fft_Par([257:1280],:);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT OFDM symbols

        %FFT function
        fftSymOut = fft(fftSym);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove the pilots, DC zero and zero padding of the Symbols
```

```
%Remove Pilots, DC and zero pads
fftSymPil([1:64],:)     =  fftSymOut([103,116,129,142,155,...
    168,181,194,207,220,233,246,259,272,285,298,311,324,...
    337,350,363,376,389,402,415,428,441,454,467,480,493,...
    506,520,533,546,559,572,585,598,611,624,637,650,663,...
    676,689,702,715,728,741,754,767,780,793,806,819,832,...
    845,858,871,884,897,910,923],:);
fftSymZeros([1:191],:) =  fftSymOut([1:96,930:1024],:);
fftSymDCzero([1],:)    =  fftSymOut([513],:);

%Remove data sub-carriers
fftSymDat([1:768],:)    =  fftSymOut([97:102,104:115,117:128,...
    130:141,143:154,156:167,169:180,182:193,195:206,208:219,...
    221:232,234:245,247:258,260:271,273:284,286:297,299:310,...
    312:323,325:336,338:349,351:362,364:375,377:388,390:401,...
    403:414,416:427,429:440,442:453,455:466,468:479,481:492,...
    494:505,507:512,514:519,521:532,534:545,547:558,560:571,...
    573:584,586:597,599:610,612:623,625:636,638:649,651:662,...
    664:675,677:688,690:701,703:714,716:727,729:740,742:753,...
    755:766,768:779,781:792,794:805,807:818,820:831,833:844,...
    846:857,859:870,872:883,885:896,898:909,911:922,924:929],:);


[fftSymLen,fftSymWid] = size(fftSymDat);
fftDatSer = reshape(fftSymDat,(fftSymLen*fftSymWid),1);
fftSerLen = length(fftDatSer);
fftMessLen = fftSerLen - noExtbits;
fftMess([1:fftMessLen],:) = fftDatSer([1:fftMessLen],:);

%fftMessLen;
%fftMess;
fftMessa = reshape(fftMess,1,fftMessLen);
```

```
        if fft_ModTy == 1
        fftMessa = real(fftMessa);

        %makes negative zeros (-0) a zero (0) for biterr
        fftMessa = fftMessa.*fftMessa;
        fftMessa = round(fftMessa);

        end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Output

        %Output varaiables
        fft_out = fftMessa;
```

B.29 ofdm_fft2048

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT2048
function [fft_out] = ofdm_fft2048(fft_in,handles)


        %Assigning variables
        fft_input = fft_in;
        fftLenOrig = length(fft_input);
```

```
noSubCar = handles.noSubCar;
noDatCar = handles.noDataCar;
noPilCar = handles.noPilotCar;
noPadCar = handles.noPadCar;
noCeCar  = handles.noCeCar;
noTotCar = handles.totSubChan;
fft_ModTy = handles.modTypeVal;
noExtbits = handles.ifftextrabits;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove Cyclic prefix

%Reshape to make parallel, ensures Total Sub-carrier deep
[fftInpLen,fftInpWid] = size(fft_input);
fft_Par = reshape(fft_input,2560,[]);

%Remove cyclic extension
fftSym([1:2048],:) = fft_Par([513:2560],:);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT OFDM symbols

%FFT function
fftSymOut = fft(fftSym);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove the pilots, DC zero and zero padding of the Symbols

%Remove Pilots, DC and zero pads
fftSymPil([1:128],:)     =  ...
fftSymOut([199,212,225,238,251,264,277,290,303,316,329,...
    342,355,368,381,394,407,420,433,446,459,472,485,498,...
    511,524,537,550,563,576,589,602,615,628,641,654,667,...
```

```
    680,693,706,719,732,745,758,771,784,797,810,823,836,...
    849,862,875,888,901,914,927,940,953,966,979,992,1005,...
    1018,1032,1045,1058,1071,1084,1097,1110,1123,1136,...
    1149,1162,1175,1188,1201,1214,1227,1240,1253,1266,...
    1279,1292,1305,1318,1331,1344,1357,1370,1383,1396,...
    1409,1422,1435,1448,1461,1474,1487,1500,1513,1526,...
    1539,1552,1565,1578,1591,1604,1617,1630,1643,1656,...
    1669,1682,1695,1708,1721,1734,1747,1760,1773,1786,...
    1799,1812,1825,1838,1851],:);


fftSymZeros([1:383],:) = fftSymOut([1:192,1858:2048],:);
fftSymDCzero([1],:)    = fftSymOut([1025],:);

%Remove data sub-carriers
fftSymDat([1:1536],:)    = ...
fftSymOut([193:198,200:211,213:224,226:237,239:250,252:263,...
    265:276,278:289,291:302,304:315,317:328,330:341,343:354,...
    356:367,369:380,382:393,395:406,408:419,421:432,434:445,...
    447:458,460:471,473:484,486:497,499:510,512:523,525:536,...
    538:549,551:562,564:575,577:588,590:601,603:614,616:627,...
    629:640,642:653,655:666,668:679,681:692,694:705,707:718,...
    720:731,733:744,746:757,759:770,772:783,785:796,798:809,...
    811:822,824:835,837:848,850:861,863:874,876:887,889:900,...
    902:913,915:926,928:939,941:952,954:965,967:978,980:991,...
    993:1004,1006:1017,1019:1024,1026:1031,1033:1044,...
    1046:1057,1059:1070,1072:1083,1085:1096,1098:1109,...
    1111:1122,1124:1135,1137:1148,1150:1161,1163:1174,...
    1176:1187,1189:1200,1202:1213,1215:1226,1228:1239,...
    1241:1252,1254:1265,1267:1278,1280:1291,1293:1304,...
    1306:1317,1319:1330,1332:1343,1345:1356,1358:1369,...
    1371:1382,1384:1395,1397:1408,1410:1421,1423:1434,...
    1436:1447,1449:1460,1462:1473,1475:1486,1488:1499,...
```

```
        1501:1512,1514:1525,1527:1538,1540:1551,1553:1564,...
        1566:1577,1579:1590,1592:1603,1605:1616,1618:1629,...
        1631:1642,1644:1655,1657:1668,1670:1681,1683:1694,...
        1696:1707,1709:1720,1722:1733,1735:1746,1748:1759,...
        1761:1772,1774:1785,1787:1798,1800:1811,1813:1824,...
        1826:1837,1839:1850,1852:1857],:);

    [fftSymLen,fftSymWid] = size(fftSymDat);
    fftDatSer = reshape(fftSymDat,(fftSymLen*fftSymWid),1);
    fftSerLen = length(fftDatSer);
    fftMessLen = fftSerLen - noExtbits;
    fftMess([1:fftMessLen],:) = fftDatSer([1:fftMessLen],:);
    fftMessa = reshape(fftMess,1,fftMessLen);

    if fft_ModTy == 1
    fftMessa = real(fftMessa);

    %makes negative zeros (-0) a zero (0) for biterr
    fftMessa = fftMessa.*fftMessa;
    fftMessa = round(fftMessa);

    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Output

    %Output varaiables
    fft_out = fftMessa;
```

B.30 ofdm_fft4096

%%

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%                Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT4096
function [fft_out] = ofdm_fft4096(fft_in,handles)


        %Assigning variables
        fft_input = fft_in;
        fftLenOrig = length(fft_input);
        noSubCar = handles.noSubCar;
        noDatCar = handles.noDataCar;
        noPilCar = handles.noPilotCar;
        noPadCar = handles.noPadCar;
        noCeCar  = handles.noCeCar;
        noTotCar = handles.totSubChan;
        fft_ModTy = handles.modTypeVal;
        noExtbits = handles.ifftextrabits;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove Cyclic prefix

        %Reshape to make parallel, ensures Total Sub-carrier deep
        [fftInpLen,fftInpWid] = size(fft_input);
        fft_Par = reshape(fft_input,5120,[]);

        %Remove cyclic extension
        fftSym([1:4096],:) = fft_Par([1025:5120],:);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT OFDM symbols
```

```
%FFT function
fftSymOut = fft(fftSym);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove the pilots, DC zero and zero padding of the Symbols

%Remove Pilots, DC and zero pads
fftSymPil([1:256],:)     =  ...
fftSymOut([391,404,417,430,443,456,469,482,495,508,521,534,...
    547,560,573,586,599,612,625,638,651,664,677,690,703,716,...
    729,742,755,768,781,794,807,820,833,846,859,872,885,898,...
    911,924,937,950,963,976,989,1002,1015,1028,1041,1054,1067,...
    1080,1093,1106,1119,1132,1145,1158,1171,1184,1197,1210,...
    1223,1236,1249,1262,1275,1288,1301,1314,1327,1340,1353,...
    1366,1379,1392,1405,1418,1431,1444,1457,1470,1483,1496,...
    1509,1522,1535,1548,1561,1574,1587,1600,1613,1626,1639,...
    1652,1665,1678,1691,1704,1717,1730,1743,1756,1769,1782,...
    1795,1808,1821,1834,1847,1860,1873,1886,1899,1912,1925,...
    1938,1951,1964,1977,1990,2003,2016,2029,2042,2056,2069,...
    2082,2095,2108,2121,2134,2147,2160,2173,2186,2199,2212,...
    2225,2238,2251,2264,2277,2290,2303,2316,2329,2342,2355,...
    2368,2381,2394,2407,2420,2433,2446,2459,2472,2485,2498,...
    2511,2524,2537,2550,2563,2576,2589,2602,2615,2628,2641,...
    2654,2667,2680,2693,2706,2719,2732,2745,2758,2771,2784,...
    2797,2810,2823,2836,2849,2862,2875,2888,2901,2914,2927,...
    2940,2953,2966,2979,2992,3005,3018,3031,3044,3057,3070,...
    3083,3096,3109,3122,3135,3148,3161,3174,3187,3200,3213,...
    3226,3239,3252,3265,3278,3291,3304,3317,3330,3343,3356,...
    3369,3382,3395,3408,3421,3434,3447,3460,3473,3486,3499,...
    3512,3525,3538,3551,3564,3577,3590,3603,3616,3629,3642,...
    3655,3668,3681,3694,3707],:);
```

```
fftSymZeros([1:767],:) = fftSymOut([1:384,3714:4096],:);
fftSymDCzero([1],:)    = fftSymOut([2049],:);

%Remove data sub-carriers
fftSymDat([1:3072],:)  = ...
fftSymOut([385:390,392:403,405:416,418:429,431:442,444:455,...
    457:468,470:481,483:494,496:507,509:520,522:533,535:546,...
    548:559,561:572,574:585,587:598,600:611,613:624,626:637,...
    639:650,652:663,665:676,678:689,691:702,704:715,717:728,...
    730:741,743:754,756:767,769:780,782:793,795:806,808:819,...
    821:832,834:845,847:858,860:871,873:884,886:897,899:910,...
    912:923,925:936,938:949,951:962,964:975,977:988,990:1001,...
    1003:1014,1016:1027,1029:1040,1042:1053,1055:1066,...
    1068:1079,1081:1092,1094:1105,1107:1118,1120:1131,...
    1133:1144,1146:1157,1159:1170,1172:1183,1185:1196,...
    1198:1209,1211:1222,1224:1235,1237:1248,1250:1261,...
    1263:1274,1276:1287,1289:1300,1302:1313,1315:1326,...
    1328:1339,1341:1352,1354:1365,1367:1378,1380:1391,...
    1393:1404,1406:1417,1419:1430,1432:1443,1445:1456,...
    1458:1469,1471:1482,1484:1495,1497:1508,1510:1521,...
    1523:1534,1536:1547,1549:1560,1562:1573,1575:1586,...
    1588:1599,1601:1612,1614:1625,1627:1638,1640:1651,...
    1653:1664,1666:1677,1679:1690,1692:1703,1705:1716,...
    1718:1729,1731:1742,1744:1755,1757:1768,1770:1781,...
    1783:1794,1796:1807,1809:1820,1822:1833,1835:1846,...
    1848:1859,1861:1872,1874:1885,1887:1898,1900:1911,...
    1913:1924,1926:1937,1939:1950,1952:1963,1965:1976,...
    1978:1989,1991:2002,2004:2015,2017:2028,2030:2041,...
    2043:2048,2050:2055,2057:2068,2070:2081,2083:2094,...
    2096:2107,2109:2120,2122:2133,2135:2146,2148:2159,...
    2161:2172,2174:2185,2187:2198,2200:2211,2213:2224,...
```

```
2226:2237,2239:2250,2252:2263,2265:2276,2278:2289,...
2291:2302,2304:2315,2317:2328,2330:2341,2343:2354,...
2356:2367,2369:2380,2382:2393,2395:2406,2408:2419,...
2421:2432,2434:2445,2447:2458,2460:2471,2473:2484,...
2486:2497,2499:2510,2512:2523,2525:2536,2538:2549,...
2551:2562,2564:2575,2577:2588,2590:2601,2603:2614,...
2616:2627,2629:2640,2642:2653,2655:2666,2668:2679,...
2681:2692,2694:2705,2707:2718,2720:2731,2733:2744,...
2746:2757,2759:2770,2772:2783,2785:2796,2798:2809,...
2811:2822,2824:2835,2837:2848,2850:2861,2863:2874,...
2876:2887,2889:2900,2902:2913,2915:2926,2928:2939,...
2941:2952,2954:2965,2967:2978,2980:2991,2993:3004,...
3006:3017,3019:3030,3032:3043,3045:3056,3058:3069,...
3071:3082,3084:3095,3097:3108,3110:3121,3123:3134,...
3136:3147,3149:3160,3162:3173,3175:3186,3188:3199,...
3201:3212,3214:3225,3227:3238,3240:3251,3253:3264,...
3266:3277,3279:3290,3292:3303,3305:3316,3318:3329,...
3331:3342,3344:3355,3357:3368,3370:3381,3383:3394,...
3396:3407,3409:3420,3422:3433,3435:3446,3448:3459,...
3461:3472,3474:3485,3487:3498,3500:3511,3513:3524,...
3526:3537,3539:3550,3552:3563,3565:3576,3578:3589,...
3591:3602,3604:3615,3617:3628,3630:3641,3643:3654,...
3656:3667,3669:3680,3682:3693,3695:3706,3708:3713],:);


[fftSymLen,fftSymWid] = size(fftSymDat);
fftDatSer = reshape(fftSymDat,(fftSymLen*fftSymWid),1);
fftSerLen = length(fftDatSer);
%noExtbits;
fftMessLen = fftSerLen - noExtbits;
fftMess([1:fftMessLen],:) = fftDatSer([1:fftMessLen],:);

%fftMessLen;
```

```
%fftMess;
fftMessa = reshape(fftMess,1,fftMessLen);



if fft_ModTy == 1
fftMessa = real(fftMessa);

%makes negative zeros (-0) a zero (0) for biterr
fftMessa = fftMessa.*fftMessa;
fftMessa = round(fftMessa);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Output

%Output varaiables
fft_out = fftMessa;
```

B.31 ofdm_fft8192

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT8192
function [fft_out] = ofdm_fft8192(fft_in,handles)
```

```
        %Assigning variables
        fft_input = fft_in;
        fftLenOrig = length(fft_input);
        noSubCar = handles.noSubCar;
        noDatCar = handles.noDataCar;
        noPilCar = handles.noPilotCar;
        noPadCar = handles.noPadCar;
        noCeCar  = handles.noCeCar;
        noTotCar = handles.totSubChan;
        fft_ModTy = handles.modTypeVal;
        noExtbits = handles.ifftextrabits;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove Cyclic prefix

        %Reshape to make parallel, ensures Total Sub-carrier deep
        [fftInpLen,fftInpWid] = size(fft_input);
        fft_Par = reshape(fft_input,10240,[]);

        %Remove cyclic extension
        fftSym([1:8192],:) = fft_Par([2049:10240],:);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% FFT OFDM symbols

        %FFT function
        fftSymOut = fft(fftSym);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove the pilots, DC zero and zero padding of the Symbols

        %Remove Pilots, DC and zero pads
        fftSymPil([1:512],:)     =  ...
```

```
fftSymOut([775,788,801,814,827,840,853,866,879,892,905,918,...
    931,944,957,970,983,996,1009,1022,1035,1048,1061,1074,...
    1087,1100,1113,1126,1139,1152,1165,1178,1191,1204,1217,...
    1230,1243,1256,1269,1282,1295,1308,1321,1334,1347,1360,...
    1373,1386,1399,1412,1425,1438,1451,1464,1477,1490,1503,...
    1516,1529,1542,1555,1568,1581,1594,1607,1620,1633,1646,...
    1659,1672,1685,1698,1711,1724,1737,1750,1763,1776,1789,...
    1802,1815,1828,1841,1854,1867,1880,1893,1906,1919,1932,...
    1945,1958,1971,1984,1997,2010,2023,2036,2049,2062,2075,...
    2088,2101,2114,2127,2140,2153,2166,2179,2192,2205,2218,...
    2231,2244,2257,2270,2283,2296,2309,2322,2335,2348,2361,...
    2374,2387,2400,2413,2426,2439,2452,2465,2478,2491,2504,...
    2517,2530,2543,2556,2569,2582,2595,2608,2621,2634,2647,...
    2660,2673,2686,2699,2712,2725,2738,2751,2764,2777,2790,...
    2803,2816,2829,2842,2855,2868,2881,2894,2907,2920,2933,...
    2946,2959,2972,2985,2998,3011,3024,3037,3050,3063,3076,...
    3089,3102,3115,3128,3141,3154,3167,3180,3193,3206,3219,...
    3232,3245,3258,3271,3284,3297,3310,3323,3336,3349,3362,...
    3375,3388,3401,3414,3427,3440,3453,3466,3479,3492,3505,...
    3518,3531,3544,3557,3570,3583,3596,3609,3622,3635,3648,...
    3661,3674,3687,3700,3713,3726,3739,3752,3765,3778,3791,...
    3804,3817,3830,3843,3856,3869,3882,3895,3908,3921,3934,...
    3947,3960,3973,3986,3999,4012,4025,4038,4051,4064,4077,...
    4090,4104,4117,4130,4143,4156,4169,4182,4195,4208,4221,...
    4234,4247,4260,4273,4286,4299,4312,4325,4338,4351,4364,...
    4377,4390,4403,4416,4429,4442,4455,4468,4481,4494,4507,...
    4520,4533,4546,4559,4572,4585,4598,4611,4624,4637,4650,...
    4663,4676,4689,4702,4715,4728,4741,4754,4767,4780,4793,...
    4806,4819,4832,4845,4858,4871,4884,4897,4910,4923,4936,...
    4949,4962,4975,4988,5001,5014,5027,5040,5053,5066,5079,...
    5092,5105,5118,5131,5144,5157,5170,5183,5196,5209,5222,...
    5235,5248,5261,5274,5287,5300,5313,5326,5339,5352,5365,...
    5378,5391,5404,5417,5430,5443,5456,5469,5482,5495,5508,...
```

```
    5521,5534,5547,5560,5573,5586,5599,5612,5625,5638,5651,...
    5664,5677,5690,5703,5716,5729,5742,5755,5768,5781,5794,...
    5807,5820,5833,5846,5859,5872,5885,5898,5911,5924,5937,...
    5950,5963,5976,5989,6002,6015,6028,6041,6054,6067,6080,...
    6093,6106,6119,6132,6145,6158,6171,6184,6197,6210,6223,...
    6236,6249,6262,6275,6288,6301,6314,6327,6340,6353,6366,...
    6379,6392,6405,6418,6431,6444,6457,6470,6483,6496,6509,...
    6522,6535,6548,6561,6574,6587,6600,6613,6626,6639,6652,...
    6665,6678,6691,6704,6717,6730,6743,6756,6769,6782,6795,...
    6808,6821,6834,6847,6860,6873,6886,6899,6912,6925,6938,...
    6951,6964,6977,6990,7003,7016,7029,7042,7055,7068,7081,...
    7094,7107,7120,7133,7146,7159,7172,7185,7198,7211,7224,...
    7237,7250,7263,7276,7289,7302,7315,7328,7341,7354,7367,...
    7380,7393,7406,7419],:);


fftSymZeros([1:1535],:) = fftSymOut([1:768,7426:8192],:);
fftSymDCzero([1],:)     = fftSymOut([4097],:);

%Remove data sub-carriers
fftSymDat([1:6144],:)    = ...
fftSymOut([769:774,776:787,789:800,802:813,815:826,828:839,...
    841:852,854:865,867:878,880:891,893:904,906:917,919:930,...
    932:943,945:956,958:969,971:982,984:995,997:1008,...
    1010:1021,1023:1034,1036:1047,1049:1060,1062:1073,...
    1075:1086,1088:1099,1101:1112,1114:1125,1127:1138,...
    1140:1151,1153:1164,1166:1177,1179:1190,1192:1203,...
    1205:1216,1218:1229,1231:1242,1244:1255,1257:1268,...
    1270:1281,1283:1294,1296:1307,1309:1320,1322:1333,...
    1335:1346,1348:1359,1361:1372,1374:1385,1387:1398,...
    1400:1411,1413:1424,1426:1437,1439:1450,1452:1463,...
    1465:1476,1478:1489,1491:1502,1504:1515,1517:1528,...
    1530:1541,1543:1554,1556:1567,1569:1580,1582:1593,...
```

```
1595:1606,1608:1619,1621:1632,1634:1645,1647:1658,...
1660:1671,1673:1684,1686:1697,1699:1710,1712:1723,...
1725:1736,1738:1749,1751:1762,1764:1775,1777:1788,...
1790:1801,1803:1814,1816:1827,1829:1840,1842:1853,...
1855:1866,1868:1879,1881:1892,1894:1905,1907:1918,...
1920:1931,1933:1944,1946:1957,1959:1970,1972:1983,...
1985:1996,1998:2009,2011:2022,2024:2035,2037:2048,...
2050:2061,2063:2074,2076:2087,2089:2100,2102:2113,...
2115:2126,2128:2139,2141:2152,2154:2165,2167:2178,...
2180:2191,2193:2204,2206:2217,2219:2230,2232:2243,...
2245:2256,2258:2269,2271:2282,2284:2295,2297:2308,...
2310:2321,2323:2334,2336:2347,2349:2360,2362:2373,...
2375:2386,2388:2399,2401:2412,2414:2425,2427:2438,...
2440:2451,2453:2464,2466:2477,2479:2490,2492:2503,...
2505:2516,2518:2529,2531:2542,2544:2555,2557:2568,...
2570:2581,2583:2594,2596:2607,2609:2620,2622:2633,...
2635:2646,2648:2659,2661:2672,2674:2685,2687:2698,...
2700:2711,2713:2724,2726:2737,2739:2750,2752:2763,...
2765:2776,2778:2789,2791:2802,2804:2815,2817:2828,...
2830:2841,2843:2854,2856:2867,2869:2880,2882:2893,...
2895:2906,2908:2919,2921:2932,2934:2945,2947:2958,...
2960:2971,2973:2984,2986:2997,2999:3010,3012:3023,...
3025:3036,3038:3049,3051:3062,3064:3075,3077:3088,...
3090:3101,3103:3114,3116:3127,3129:3140,3142:3153,...
3155:3166,3168:3179,3181:3192,3194:3205,3207:3218,...
3220:3231,3233:3244,3246:3257,3259:3270,3272:3283,...
3285:3296,3298:3309,3311:3322,3324:3335,3337:3348,...
3350:3361,3363:3374,3376:3387,3389:3400,3402:3413,...
3415:3426,3428:3439,3441:3452,3454:3465,3467:3478,...
3480:3491,3493:3504,3506:3517,3519:3530,3532:3543,...
3545:3556,3558:3569,3571:3582,3584:3595,3597:3608,...
3610:3621,3623:3634,3636:3647,3649:3660,3662:3673,...
3675:3686,3688:3699,3701:3712,3714:3725,3727:3738,...
```

```
3740:3751,3753:3764,3766:3777,3779:3790,3792:3803,...
3805:3816,3818:3829,3831:3842,3844:3855,3857:3868,...
3870:3881,3883:3894,3896:3907,3909:3920,3922:3933,...
3935:3946,3948:3959,3961:3972,3974:3985,3987:3998,...
4000:4011,4013:4024,4026:4037,4039:4050,4052:4063,...
4065:4076,4078:4089,4091:4096,4098:4103,4105:4116,...
4118:4129,4131:4142,4144:4155,4157:4168,4170:4181,...
4183:4194,4196:4207,4209:4220,4222:4233,4235:4246,...
4248:4259,4261:4272,4274:4285,4287:4298,4300:4311,...
4313:4324,4326:4337,4339:4350,4352:4363,4365:4376,...
4378:4389,4391:4402,4404:4415,4417:4428,4430:4441,...
4443:4454,4456:4467,4469:4480,4482:4493,4495:4506,...
4508:4519,4521:4532,4534:4545,4547:4558,4560:4571,...
4573:4584,4586:4597,4599:4610,4612:4623,4625:4636,...
4638:4649,4651:4662,4664:4675,4677:4688,4690:4701,...
4703:4714,4716:4727,4729:4740,4742:4753,4755:4766,...
4768:4779,4781:4792,4794:4805,4807:4818,4820:4831,...
4833:4844,4846:4857,4859:4870,4872:4883,4885:4896,...
4898:4909,4911:4922,4924:4935,4937:4948,4950:4961,...
4963:4974,4976:4987,4989:5000,5002:5013,5015:5026,...
5028:5039,5041:5052,5054:5065,5067:5078,5080:5091,...
5093:5104,5106:5117,5119:5130,5132:5143,5145:5156,...
5158:5169,5171:5182,5184:5195,5197:5208,5210:5221,...
5223:5234,5236:5247,5249:5260,5262:5273,5275:5286,...
5288:5299,5301:5312,5314:5325,5327:5338,5340:5351,...
5353:5364,5366:5377,5379:5390,5392:5403,5405:5416,...
5418:5429,5431:5442,5444:5455,5457:5468,5470:5481,...
5483:5494,5496:5507,5509:5520,5522:5533,5535:5546,...
5548:5559,5561:5572,5574:5585,5587:5598,5600:5611,...
5613:5624,5626:5637,5639:5650,5652:5663,5665:5676,...
5678:5689,5691:5702,5704:5715,5717:5728,5730:5741,...
5743:5754,5756:5767,5769:5780,5782:5793,5795:5806,...
5808:5819,5821:5832,5834:5845,5847:5858,5860:5871,...
```

```
5873:5884,5886:5897,5899:5910,5912:5923,5925:5936,...
5938:5949,5951:5962,5964:5975,5977:5988,5990:6001,...
6003:6014,6016:6027,6029:6040,6042:6053,6055:6066,...
6068:6079,6081:6092,6094:6105,6107:6118,6120:6131,...
6133:6144,6146:6157,6159:6170,6172:6183,6185:6196,...
6198:6209,6211:6222,6224:6235,6237:6248,6250:6261,...
6263:6274,6276:6287,6289:6300,6302:6313,6315:6326,...
6328:6339,6341:6352,6354:6365,6367:6378,6380:6391,...
6393:6404,6406:6417,6419:6430,6432:6443,6445:6456,...
6458:6469,6471:6482,6484:6495,6497:6508,6510:6521,...
6523:6534,6536:6547,6549:6560,6562:6573,6575:6586,...
6588:6599,6601:6612,6614:6625,6627:6638,6640:6651,...
6653:6664,6666:6677,6679:6690,6692:6703,6705:6716,...
6718:6729,6731:6742,6744:6755,6757:6768,6770:6781,...
6783:6794,6796:6807,6809:6820,6822:6833,6835:6846,...
6848:6859,6861:6872,6874:6885,6887:6898,6900:6911,...
6913:6924,6926:6937,6939:6950,6952:6963,6965:6976,...
6978:6989,6991:7002,7004:7015,7017:7028,7030:7041,...
7043:7054,7056:7067,7069:7080,7082:7093,7095:7106,...
7108:7119,7121:7132,7134:7145,7147:7158,7160:7171,...
7173:7184,7186:7197,7199:7210,7212:7223,7225:7236,...
7238:7249,7251:7262,7264:7275,7277:7288,7290:7301,...
7303:7314,7316:7327,7329:7340,7342:7353,7355:7366,...
7368:7379,7381:7392,7394:7405,7407:7418,7420:7425],:);

[fftSymLen,fftSymWid] = size(fftSymDat);
fftDatSer = reshape(fftSymDat,(fftSymLen*fftSymWid),1);
fftSerLen = length(fftDatSer);
%noExtbits;
fftMessLen = fftSerLen - noExtbits;
fftMess([1:fftMessLen],:) = fftDatSer([1:fftMessLen],:);

%fftMessLen;
```

```
%fftMess;
fftMessa = reshape(fftMess,1,fftMessLen);



if fft_ModTy == 1
fftMessa = real(fftMessa);

%makes negative zeros (-0) a zero (0) for biterr
fftMessa = fftMessa.*fftMessa;
fftMessa = round(fftMessa);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Output

%Output varaiables
fft_out = fftMessa;
```

B.32 ofdm_demodulator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%                Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Demodulator
function demod_out = ofdm_demodulator(demod_in, modType,extdebits)

popupdemod_sel_id = modType;
```

```
switch popupdemod_sel_id

    case 1

        demod_out = demod_in;

    case 2  %BPSK Demodulation

        M=2;

        %Number of bits per symbol eg. 1 bits per symbol
        k = log2(M);

        inputSymLength = length(demod_in);
        input_sym_len = inputSymLength*k;

        msg_rx = demod_in;
        msg_dem = pskdemod(msg_rx,M);
        msg_dem_len = length(msg_dem);
        msg_rx_sym = de2bi(msg_dem,'left-msb');
        [m,n] = size(msg_rx_sym);
        msg_rx_size = m*n;
        msg_rx_orig = reshape(msg_rx_sym.',msg_rx_size,1);
        extra_bits = extdebits;
        if extra_bits > 1
        newlen = input_bit_len - extra_bits;
        msg_rx_orig(newlen+1:1:input_bit_len)=[];
        msg_newlen = length(msg_rx_orig);
        end
        demod_out = msg_rx_orig;
```

```
case 3  %QPSK Demodulation

    M=4;
    %Number of bits per symbol eg. 2 bits per symbol
    k = log2(M);

    inputSymLength = length(demod_in);
    input_sym_len = inputSymLength*k;

    msg_rx = demod_in;
    msg_dem = pskdemod(msg_rx,M);
    msg_dem_len = length(msg_dem);
    msg_rx_sym = de2bi(msg_dem,'left-msb');
    [m,n] = size(msg_rx_sym);
    msg_rx_size = m*n;
    msg_rx_orig = reshape(msg_rx_sym.',msg_rx_size,1);
    extra_bits = extdebits;

    if extra_bits > 1
    newlen = input_bit_len - extra_bits;
    msg_rx_orig(newlen+1:1:input_bit_len)=[];
    msg_newlen = length(msg_rx_orig);
    end

    demod_out = msg_rx_orig;


case 4  %16QAM Demodulation

    M=16;
    %Number of bits per symbol eg. 4 bits per symbol
    k = log2(M);
    inputSymLength = length(demod_in);
```

```
input_bit_len = inputSymLength*k;

msg_rx = demod_in;
msg_dem = qamdemod(msg_rx,M);
msg_dem_len = length(msg_dem);
msg_rx_sym = de2bi(msg_dem,'left-msb');
[m,n] = size(msg_rx_sym);
msg_rx_size = m*n;
msg_rx_orig = reshape(msg_rx_sym.',msg_rx_size,1);
extra_bits = extdebits;
if extra_bits > 1
newlen = input_bit_len - extra_bits;
msg_rx_orig(newlen+1:1:input_bit_len)=[];
msg_newlen = length(msg_rx_orig);
end
demod_out = msg_rx_orig;


case 5  %64QAM Demodulation

M=64;
[am,an] = size(demod_in);
%Number of bits per symbol eg. 6 bits per symbol
k = log2(M);

inputSymLength = length(demod_in);
input_bit_len = inputSymLength*k;

msg_rx = demod_in;
msg_dem = qamdemod(msg_rx,M);
msg_dem_len = length(msg_dem);
msg_rx_sym = de2bi(msg_dem,'left-msb');
[m,n] = size(msg_rx_sym);
```

```
        msg_rx_size = m*n;
        msg_rx_orig = reshape(msg_rx_sym.',msg_rx_size,1);
        extra_bits = extdebits;

        if extra_bits > 1
        newlen = input_bit_len - extra_bits;
        msg_rx_orig(newlen+1:1:input_bit_len)=[];
        msg_newlen = length(msg_rx_orig);
        end

        demod_out = msg_rx_orig;

end
```

B.33 ofdm_deinterleaver

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%                Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% De-interleaver
function [deinterlvd_out] = ofdm_deinterleaver(deinterlvd_in,...
    NoSub,MapType,deinterlvd_ext,deinter_perm)

        %Assigning variables
        deinterleavIn = deinterlvd_in;
        deinterleavLen = length(deinterleavIn);
        NoSubc = NoSub;
        ModType = MapType;
        DelevExtBits = deinterlvd_ext;
```

```
qr = deinter_perm;

switch ModType

    case 1

        NoBits = 1;

    case 2

        NoBits = 1;

    case 3
        NoBits = 2;

    case 4

        NoBits = 4;

    case 5

        NoBits = 6;
end

deinterlvinput = reshape(deinterleavIn,(NoSubc*NoBits),[]);
%De-interleave
deintrlvd = deintrlv(deinterlvinput,qr);

newdele = reshape(deintrlvd,1,[]);

if DelevExtBits >= 1

    %Subtract extra bits
```

```
            newlen = deinterleavLen - DelevExtBits;
            newdele(newlen+1:1:deinterleavLen)=[];
            msg_newlen = length(newdele);

        end

        deinterlvd_out = newdele.';
```

B.34 ofdm_decoder

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%          Project - OFDM Simulator
%
%Barry Dunbar  0050022993
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Decoder
function [decod_out] = ofdm_decoder(decod_in, decodType,extdebits)

        %Assigning variables
        deCodeIn = -2*decod_in+1;
        deCodeLengthOrig = length(deCodeIn);
        popupcod_sel_id = decodType;
        deCodebits = extdebits;

switch  popupcod_sel_id


    case 1  %No coding

        %Out equals In after normalising for BER checks
        decod_out = -(deCodeIn/2)+0.5;
```

```
case 2  %1/2 Rate

    %Trellis code
    t = poly2trellis(7,[133 171]);

    % Decode
    decoded = vitdec(deCodeIn,t,96,'trunc','unquant');

    %Output variable
    decod_out = decoded;


case 3  %2/3 Rate

    %Original length of zeros represent inserted data
    noCode = zeros(((4/3)*deCodeLengthOrig - deCodebits),1);

    %Place data in matrix in correct position
    noCode(1:4:end) = deCodeIn(1:3:end);
    noCode(2:4:end) = deCodeIn(2:3:end);
    noCode(3:4:end) = deCodeIn(3:3:end);

    %Trellis code
    t = poly2trellis(7,[133 171]);

    %Decode
    decoded = vitdec(noCode,t,96,'trunc','unquant');

    %Output variable
    decod_out = decoded;
```

```
    case 4  %3/4 Rate

        %Original length of zeros represent inserted data
        noCode = zeros(((3/2)*deCodeLengthOrig - deCodebits),1);

        %Place data in matrix in correct position
        noCode(1:6:end) = deCodeIn(1:4:end);
        noCode(2:6:end) = deCodeIn(2:4:end);
        noCode(3:6:end) = deCodeIn(3:4:end);
        noCode(6:6:end) = deCodeIn(4:4:end);

        %Trellis code
        t = poly2trellis(7,[133 171]);

        %Decode
        decoded = vitdec(noCode,t,96,'trunc','unquant');

        %Output variable
        decod_out = decoded;


end
```