



UNIVERSITEIT • STELLENBOSCH • UNIVERSITY
jou kennisvenoot • your knowledge partner

Orthogonal Frequency Division Multiplexing
(OFDM) implementation as part of a Software
Defined Radio (SDR) environment

By

Christoph Sonntag



*Thesis presented in partial fulfilment
Of the requirements for the degree of*

Master of Electronic Engineering

Department of Electrical and Electronic Engineering
University of Stellenbosch
Private Bag X1, Matieland, 7602, South Africa

Supervisor: J. Lourens.

December 2005

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and has not previously in its entirety or in part been submitted at any university for a degree.

Signature:

Date:

Christoph Sonntag

Abstract

Orthogonal Frequency Division Multiplexing (OFDM) has gained considerable attention the past couple of years. In our modern world the need for faster data transmission is never-ending. OFDM modulation provides us with a way of more densely packing modulated carriers in the frequency domain than other existing Frequency Multiplexing schemes, thus achieving higher data rates through communications channels.

Software Defined Radio (SDR) creates a very good entry point for designing any communications system. SDR is an architecture that aims to minimise hardware components in electronic communications circuits by doing all possible processing in the software domain. Such systems have many advantages over existing hardware implementations and can be executed on various platforms and embedded systems, given that the appropriate analogue front ends are attached to the system.

The purpose of this thesis is a study into Orthogonal Frequency Division Multiplexing and its implementation as it is described in the IEEE 802.11a specifications. The implementation is done in C++, and the software is written in a modular way, for easy porting to the Software Defined Radio libraries and other platforms.

Afterwards, the OFDM software is tested, by decoding simulated as well as real-time OFDM signals. All the results are captured and critically compared against theoretical values and other existing systems. Finally the result of this thesis is a set of tested C++ functions together with real-time and simulated performance results and a detailed thesis explaining all the major issues involved.

Opsomming

Ortogonale Frekwensie Deel Multipleksering (OFDM) is 'n digitale modulasie tegniek wat vir die afgelope paar jaar groeiende aandag ontvang het. In ons moderne wêreld is die aanvraag vir vinniger data kommunikasiestelsels nimmer-eindigend. OFDM modulasie tegnieke bied 'n manier om gemoduleerde draer-seine meer gekonsentreerd as ander bestaande Frekwensie Multiplekserende tegnieke in die frekwensie spektrum in te pas. Dus kan OFDM gebaseerde sisteme, hoër data koerse deur bestaande kommunikasie kanale handhaaf.

Sagteware Gedefinieerde Radio (SDR) bied 'n goeie ingangspunt vir die ontwerp van enige kommunikasie sisteem. SDR is 'n argitektuur wat poog om die aantal hardeware komponente in elektroniese kommunikasie stroombane te minimeer deur alle moontlike prosessering in sagteware te doen. Sulke sisteme het baie voordele bo bestaande hardeware implementasies en kan op verskillende platforms en toegewyde sisteme uitgevoer word, indien die regte analoog koppelvlak voorsien word.

Die doel van die tesis is die studie van Ortogonale Frekwensie Deel Multipleksering en die implementasie soos dit in die IEEE 802.11a spesifikasies beskryf word. Die implementasie word gedoen in C++. Die sagteware moet in 'n modulêre manier geskryf word, vir maklike oordrag na die Sagteware Gedefinieerde Radio biblioteke en ander platforms.

Uiteindelik sal die OFDM sagteware getoets word deur gesimuleerde en reële-tyd OFDM seine te dekodeer. Alle toetsresultate word krities vergelyk teen verwagte waardes en reeds bestaande sisteme. Die resultaat van die tesis is a stel getoetsde C++ programme, tesame met hulle simulاسie en toets resultate, asook 'n gedetailleerde tesis wat alle belangrike kwessies volledig bespreek.

Acknowledgements

I would like to thank the following people for their contributions to my thesis:

- My mom and dad, Alett and Ernst Sonntag, for their continuing love and support; and for believing in me.
- My project supervisor, Prof. Johan Lourens, for his help and insight.
- The University of Stellenbosch for usages of the laboratory facilities.

Glossary

ADC	Analogue-to-Digital Converter
ADSL	Asymmetric Digital Subscriber Line
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPSK	Binary Phase-shift Keying
C++	C-Plus-Plus, programming language
DAB	Digital Audio Broadcasting
DAC	Digital-to-Analogue Converter
DFT	Discrete Fourier Transform
DSL	Digital Subscriber Line
DVB	Digital Video Broadcasting
FEC	Forward Error Correction
FFT	Fast Fourier Transform
GHz	Gigahertz (1000 MHz)
F_s	Sampling Frequency
Hz	Hertz (Cycles per second)
i, j	The complex number
IDFT	Inverse Discrete Fourier Transform
IEEE	Institute of Electrical and Electronics Engineers
IFFT	Inverse Fast Fourier Transform
kHz	Kilohertz (1000 Hz)
LAN	Local Area Network
Mbps	Megabits per second (1048576 bits per second)
MHz	Megahertz (1000 kHz)
OFDM	Orthogonal Frequency Division Multiplexing
PAPR	Peak-to-Average Power Ratio
PDF	Probability Density Function
PPM	Parts Per Million
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase-shift Keying
RMS	Root Mean Square

Glossary

SDR Software Defined Radio

SER Symbol Error Rate

Contents

Declaration	2
Abstract	3
Opsomming	4
Acknowledgements	5
Glossary	6
Contents	8
List of Figures	15
List of Tables	18
1. Introduction	20
1.1 Introduction	20
1.2 Orthogonal Frequency Division Multiplexing (OFDM)	20
1.3 Software Defined Radio (SDR)	21
1.4 Thesis Objectives	22
1.5 Thesis Overview	22
1.6 Synopsis on Thesis Results	24
1.7 Conclusions	25
2. An Overview of Digital Modulation Techniques and Frequency	
Division Multiplexing	26
2.1 Introduction	26
2.2 An Overview of Digital Modulation Techniques	27
2.2.1 Amplitude Shift-Keying	27
2.2.2 Phase Shift-Keying	28
2.3 The Influence of Noise on Digital Modulation	30
2.3.1 Quantifying Additive White Gaussian Noise	31
2.3.2 Signal-To-Noise Ratio	33
2.3.3 Probability of Error In QPSK Modulation	34
2.4 An Overview of Frequency Division Multiplexing	43
2.5 Conclusions	44

Contents

3. OFDM Mathematics	45
3.1 Introduction	45
3.2 A Mathematical Approach to OFDM Symbols	45
3.2.1 Time Domain Analysis of OFDM Symbols	45
3.2.1 Frequency Domain Analysis of OFDM Symbols	47
3.3 The Issue of Orthogonality	49
3.4 Conclusions	51
4. Encoding and Decoding OFDM data symbols	52
4.1 Introduction	52
4.2 Analogue and Digital Domains	52
4.3 Encoding OFDM Symbols	55
4.3.1 Encoding Single OFDM Symbols	56
4.3.2 Encoding Multiple OFDM Symbols	57
4.4 Decoding OFDM Symbols	59
4.4.1 Decoding Single OFDM Symbols	59
4.4.2 Decoding Multiple OFDM Symbols	60
4.5 Conclusions	61
5. Identifying and Overcoming OFDM system performance influences ..	62
5.1 Introduction	62
5.2 Basic System Performance Influences	62
5.2.1 Noise	62
5.2.2 OFDM Symbol Synchronisation Issues	65
5.2.2.1 Initial OFDM Symbol Synchronisation	65
5.2.2.2 Continues re-synchronisation of OFDM data symbols ..	69
5.2.3 Multipath Effects on OFDM symbols	73
5.2.4 Peak-to-Average Power Ratio	75
5.2.4.1 OFDM Dynamic Range Issues	76
5.2.4.2 Transmitted Amplifier Issues	76

Contents

5.2.5 Communications Channel Estimation.	77
5.2.6 System Performance Influences Conclusions	80
5.3 Advanced System Performance Improvements	80
5.3.1 Data Interleaving.	80
5.3.2 Sub-carrier Adaptive Bit Loading.	81
5.3.3 Spatial Multiplexing Techniques.	81
5.4 Conclusions	81
6. IEEE 802.11a OFDM standard	82
6.1 Introduction.	82
6.2 The IEEE 802.11a Standard.	82
6.2.1 The IEEE 802.11a Standard Overview	82
6.2.2 The IEEE 802.11a Short Preamble Symbol	86
6.2.3 The IEEE 802.11a Long Preamble Symbol.	87
6.2.4 The IEEE 802.11a Signal Symbol.	89
6.2.5 The IEEE 802.11a Data Symbol	90
6.2.5.1 Convolutional Encoding Of The Digital Data.	91
6.2.5.2 Mapping Digital Data Bits To Complex Coefficients.	91
6.2.5.3 Mapping Of The Complex Coefficients To The Frequency Spectrum Locations	91
6.2.5.4 Addition Of The Reference Carriers.	91
6.2.5.5 Converting The Data Symbol To Time Domain Signal	92
6.2.5.6 Addition Of The Guard Interval	92
6.2.6 Transceiving The IEEE 802.11a Packets	92
6.3 Conclusions	93
7. SDR implementation of an IEEE 802.11a OFDM system.	95
7.1 Introduction	95
7.2 Implementation Overview	96
7.3 Hardware Implementation	98

Contents

7.4 Software Implementation.	101
7.4.1 The Graphical User Interface	101
7.4.2 The Software Structures	105
7.4.2.1 The OFDM Specification Structure	105
7.4.2.2 The OFDM Receiver Structure.	106
7.4.2.3 The OFDM Receiver Structure.	107
7.4.2.4 The OFDM Comparer Structure	108
7.4.2.5 The OFDM Audio Structure	108
7.4.3 The Software Libraries	109
7.4.3.1 The Main OFDM Library.	110
7.4.3.2 The OFDM Mathematics Library.	110
7.4.3.3 The OFDM Transmitter Library.	111
7.4.3.4 The OFDM Receiver Library	112
7.4.3.5 The OFDM Comparer Library	113
7.4.3.6 The OFDM Audio Library	114
7.4.4 Conclusions on the Software Implementation.	114
7.5 The Complete IEEE 802.11a Transceiver System.	115
7.5.1 The IEEE 802.11a Transmitter	115
7.5.1.1 The Encoding/Transmitting Process Overview	116
7.5.1.2 Initiating The OFDM Audio System	116
7.5.1.3 Initiating The Main OFDM Specifications.	117
7.5.1.4 Selecting an Appropriate Sound Device For Output	118
7.5.1.5 Initiating The OFDM Transmitter.	118
7.5.1.6 Loading The Appropriate Test Data.	120
7.5.1.7 Encoding The Test Data	122
7.5.1.8 Adding AWGN To The Transmission	124
7.5.1.9 Transmitting The IEEE 802.11a OFDM Packet.	124
7.5.1.10 Conclusions On The Encoding/Transmitting Process	125
7.5.2 The IEEE 802.11a Receiver	126
7.5.2.1 The Receiving/Decoding Process Overview.	127
7.5.2.2 Initiating The OFDM Audio System	127

Contents

7.5.2.3 Initiating The Main OFDM Specifications.	127
7.5.2.4 Selecting an Appropriate Sound Device For Input.	127
7.5.2.5 Initiating The OFDM Receiver.	128
7.5.2.6 Decoding IEEE 802.11a Signals.	129
7.5.2.7 Loading a Compare File	133
7.5.2.8 Comparing Received Data	133
7.5.2.9 Saving The Transmission to File	134
7.5.2.10 Conclusions On The Receiving/Decoding Process.	134
7.5.3 Conclusions On The IEEE 802.11a Transceiver System	135
7.6 Conclusions	135
8. OFDM system performance tests and results	136
8.1 Introduction	136
8.2 Data Performance Tests and Results	136
8.2.1 The Test Data Set	137
8.2.2 Simulation Tests	138
8.2.2.1 Simulation Test 1	138
8.2.2.2 Simulation Test 2	140
8.2.2.3 Simulation Test 3	142
8.2.2.4 Simulation Test 4	144
8.2.2.5 Simulation Test 5	146
8.2.2.6 Simulation Test 6	148
8.2.2.7 Simulation Test 7.	149
8.2.2.8 Simulation Test Conclusions	151
8.2.3 Initial Sound Device Transmission Tests	151
8.2.3.1 Communications Channel Estimation.	152
8.2.3.2 Determining Sampling Frequency Drift.	153
8.2.3.3 AWGN Performance Tests.	154
8.2.3.4 Initial Sound Device Transmission Test Conclusions	156
8.2.4 Real-time Buffered Transmission Tests.	157
8.2.4.1 The Test Data Set	157

Contents

8.2.4.2 AWGN Performance Tests.	157
8.2.4.3 Final IEEE 802.11a Decoder Performance Graph	159
8.2.4.4 Real Time Buffered Transmission Test Conclusions.	160
8.3 Speed Performance Tests and Results	160
8.3.1 Encoding Speed Tests	160
8.3.2 Decoding Speed Tests.	161
8.3.3 Conclusions on Speed Performance Tests.	162
8.4 Conclusions	164
9. Conclusions	165
9.1 Introduction	165
9.2 The IEEE 802.11a Standard and OFDM Modulation	166
9.3 Software Defined Radio.	168
9.4 Notes on the c++ Software.	168
9.5 SDR Performance Tests	169
9.6 Future Work and Research.	170
9.7 Comparing Against Other Existing OFDM Systems.	170
9.8 Conclusions	172
9.9 Thanks	172
Appendix A: Useful Mathematical Proofs	173
A.1 Useful Trigonometry Functions.	173
A.2 Finding the Zero-points of a Sync Waveform.	173
A.3 The RMS of a Sinus wave.	175
Appendix B: Digital Modulation Schemes.	177
B.1 BPSK constellation bit encoding.	177
B.2 QPSK constellation bit encoding.	178
B.3 16-QAM constellation bit encoding.	178
B.4 64-QAM constellation bit encoding.	179

Contents

Appendix C: SDR OFDM program details	181
C.1 The OFDM Specifications Structure Details	181
C.2 The OFDM Transmitter Structure Details	185
C.3 The OFDM Receiver Structure Details	187
C.4 The OFDM Comparer Structure Details	193
C.5 The OFDM Audio Structure Details	194
C.6 The Main OFDM Library Functions	195
C.7 The OFDM Math Library Functions	195
C.8 The OFDM Transmitter Library Functions	197
C.9 The OFDM Receiver Library Functions	198
C.10 The OFDM Comparer Library Functions	200
C.11 The OFDM Audio Library Functions	201
Appendix D: SDR OFDM Included CD	202
D.1 SDR OFDM Included CD	202
Bibliography	203

List of Figures

Figure 2.1: An ASK digitally modulated bit stream.	27
Figure 2.2: A BPSK digitally modulated bit stream.	29
Figure 2.3: An example of a Gaussian noise signal	32
Figure 2.4: The Gaussian probability distribution function	32
Figure 2.5: A constellation diagram of a QPSK encoded signal with AWGN. . .	35
Figure 2.6: BER and SER vs. SNR of a QPSK modulated signal	42
Figure 2.7: Example of FDMA modulated signals, and their guard-bands	43
Figure 3.1: A real-time domain OFDM signal.. . . .	46
Figure 3.2: The Fourier Transform of a rectangular window.	47
Figure 3.3: A magnitude spectrum OFDM signal.	48
Figure 4.1: The conversion of a continuous-time signal to a discrete-time signal	53
Figure 4.2: A graphical example of the single OFDM symbol encoding process.	56
Figure 4.3: A graphical example of the multiple OFDM symbol encoding processes	58
Figure 4.4: A graphical example of the single OFDM symbol decoding process.	59
Figure 4.5: An OFDM decoder incorrectly decodes an OFDM symbol due to lack of OFDM symbol synchronisation	61
Figure 5.1: A graphical example of an OFDM decoder with initial OFDM symbol synchronisation.	66
Figure 5.2: A graphical example of initial OFDM symbol synchronisation results using cross-correlation.	68
Figure 5.3: OFDM symbol de-synchronisation due to sampling frequency drift	70
Figure 5.4: Multipath effects causes delayed signals to reach the receiver	74
Figure 5.5: The effects of delay spread and the use of guard intervals.	75
Figure 6.1: An IEEE 802.11a standard packet format	83
Figure 6.2: An IEEE 802.11a IDFT/IFFT function block.	85

List of Figures

Figure 6.3: The IEEE 802.11a short preamble train.	87
Figure 6.4: The IEEE 802.11a long preamble train	89
Figure 6.5: The IEEE 802.11a signal symbol bit assignment.	89
Figure 7.1: The intended IEEE 802.11a SDR transceiver system	99
Figure 7.2: The implemented IEEE 802.11a based system.	100
Figure 7.3: The OFDM encoding/decoding demo GUI	103
Figure 7.4: The implemented IEEE 802.11a OFDM transmitter flowchart	115
Figure 7.5: The OFDM audio initiation flowchart	116
Figure 7.6: The main OFDM specifications initiation flowchart	117
Figure 7.7: The selection of an appropriate sound device for output flowchart.	118
Figure 7.8: The OFDM transmitter initiation flowchart.	119
Figure 7.9: The loading of a transmission file flowchart	120
Figure 7.10: The encoding of a IEEE 802.11a OFDM packet flowchart	123
Figure 7.11: Adding AWGN to the transmission signal flowchart.	124
Figure 7.12: Transmitting the IEEE 802.11a OFDM packet flowchart	125
Figure 7.13: The implemented IEEE 802.11a OFDM receiver flowchart	126
Figure 7.14: The OFDM receiver initiation flowchart	128
Figure 7.15: Decoding an IEEE 802.11a packet from audio device flowchart.	129
Figure 7.16: Decoding an IEEE 802.11a packet from a local file flowchart	130
Figure 7.17: The IEEE 802.11a decoder state machine flowchart	130
Figure 7.18: The IEEE 802.11a decoder state 4, data decoder flowchart.	131
Figure 7.19: The IEEE 802.11a decoder state 3, statistics extraction flowchart.	131
Figure 7.20: The IEEE 802.11a decoder state 2, signal threshold detector and state 1, initial OFDM symbol synchronisation flowchart	132
Figure 7.21: The IEEE 802.11a decoder sub-sample offset estimator flowchart.	132
Figure 7.22: Loading a compare file flowchart.	133
Figure 7.23: Comparing received data flowchart	134
Figure 7.24: Saving the received transmission to file flowchart.	134
Figure 8.1: Test Image 1.	137

List of Figures

Figure 8.2: Test Image 2.	137
Figure 8.3: Test Image 3.	138
Figure 8.4: Test Image 4.	138
Figure 8.5: Test Image 5.	138
Figure 8.6: The results from Simulation Test 2	142
Figure 8.7: Simulation BER vs. SNR of the OFDM signal (with and without the reference carrier phase compensation enabled) and the theoretically expected QPSK modulated signal.	147
Figure 8.8: Simulation BER vs. SNR of the OFDM signal with reference carrier phase compensation enabled, influenced by AWGN and sampling frequency drift.	150
Figure 8.9: The impulse response of the communications channel and sound device	152
Figure 8.10: The transfer function of the communications channel and sound device.	153
Figure 8.11: The calculated sub-sample offset and resulting sample frequency drift	154
Figure 8.12: Initial sound device transmission AWGN test, BER vs. SNR. . . .	156
Figure 8.13: Real-time buffered transmission AWGN test, BER vs. SNR. . . .	158
Figure 8.14: Final real-time buffered transmission AWGN test, BER vs. complete OFDM signal SNR	159
Figure 9.1: Implemented IEEE 802.11a transceiver comparison graph.	171
Figure B.1: BPSK constellation bit encoding	177
Figure B.2: QPSK constellation bit encoding	178
Figure B.3: 16-QAM constellation bit encoding.	178
Figure B.4: 64-QAM constellation bit encoding.	180

List of Tables

Table 2.1: BER and SER vs. SNR for a QPSK modulated signal.	42
Table 4.1: The OFDM symbol encoding process	57
Table 4.2: The OFDM symbol decoding process	60
Table 6.1: The IEEE 802.11a timing parameters.	84
Table 6.2: Data rate dependant parameters of an IEEE 802.11a system.	85
Table 6.3: Contents of the IEEE 802.11a signal field	90
Table 6.4: IEEE 802.11a data symbol generation process.	90
Table 7.1: Differences between the intended IEEE 802.11a system and the implemented system.	97
Table 7.2: Guide to the functions of the OFDM Encoding/Decoding Demo GUI.	104
Table 7.3: Test image parameters	121
Table 8.1: The three most important data performance tests.	137
Table 8.2: Simulation test 1 parameters.	139
Table 8.3: Simulation test 1 results	139
Table 8.4: Simulation test 2 parameters.	141
Table 8.5: Simulation test 3 parameters.	143
Table 8.6: Simulation test 3 results	143
Table 8.7: Simulation test 4 parameters.	145
Table 8.8: Simulation test 4 results	145
Table 8.9: Simulation test 5 parameters.	146
Table 8.10: Simulation test 6 parameters.	148
Table 8.11: Simulation test 6 results	149
Table 8.12: Simulation test 7 parameters.	149
Table 8.13: Initial sound device transmission test parameters.	155
Table 8.14: Real-time buffered transmission test parameters	158
Table 8.15: Speed test 1 parameters.	160
Table 8.16: Speed test 1 results	161
Table 8.17: Speed test 2 parameters.	161
Table 8.18: Speed test 2 results	162

List of Tables

Table 8.19: IEEE 802.11a transmission times for full and low bandwidth signals	162
Table B.1: BPSK encoding table	177
Table B.2: QPSK encoding table.	177
Table B.3: 16-QAM encoding table.	178
Table B.4: 64-QAM encoding table.	179
Table C.1: The OFDM specification structure details.	181
Table C.2: The OFDM transmitter structure details	185
Table C.3: The OFDM receiver structure details.	187
Table C.4: The OFDM comparer structure details.	193
Table C.5: The OFDM audio structure details.	194
Table C.6: The main OFDM library functions.	195
Table C.7: The OFDM math library functions.	195
Table C.8: The OFDM transmitter library functions.	197
Table C.9: The OFDM receiver library functions	198
Table C.10: The OFDM comparer library functions	200
Table C.11: The OFDM audio library functions	201

Chapter 1

Introduction

1.1 Introduction

This thesis basically consists of two major parts. The first part is a study of Orthogonal Frequency Division Multiplexing and its implementation as it is described in the IEEE 802.11a specifications. The second part is a look at Software Defined Radio, an exciting new methodology, which attempts to minimise fixed hardware components in electronic circuits by moving all possible processing to the software domain. The aim of this thesis is a basic IEEE 802.11a OFDM transceiver system written in a modular form in C++ for future implementation in embedded SDR systems.

1.2 Orthogonal Frequency Division Multiplexing (OFDM)

Orthogonal Frequency Division Multiplexing (OFDM) is a frequency multiplexing technique that has gained considerable attention the past few years. This multicarrier transmission technique densely squeezes multiple modulated sub-carriers closely together in the frequency domain, for more efficient bandwidth usage, opposed to other frequency division multiplexing systems. OFDM is already used in various communications systems including Digital Subscriber Line (DSL) systems, Digital Audio and Digital Video Broadcasting (DAB, DVB) [3,12] and wireless LAN systems called WIFI based on the IEEE 802.11a specifications [1,2].

The modulated sub-carriers within an OFDM symbol are orthogonal to each other, which means that they do not interfere with each other. Sub-carrier orthogonality is accomplished by exploiting the properties of the symbol windowing function and by choosing precise sub-carrier frequencies. Sub-carriers are encoded using different digital modulation techniques such as Binary Phase-shift Keying (BPSK), Quadrature Phase-shift Keying (QPSK) and Quadrature Amplitude Modulation (QAM).

OFDM offers many advantages above single carrier modulation schemes, and is a very good candidate for noisy office environments. The narrow-band multi-carrier modulation does not require any channel equalisation [3], and is very good at mitigating the effect of narrowband interference and frequency selective fading [12]. OFDM is also very good at mitigating the effects of time dispersions. The long OFDM symbol periods helps combat inter-symbol interference (ISI)

A digital communications system utilising an OFDM modulation scheme could theoretically use available bandwidth more efficiently than many other schemes. By breaking the bandwidth up into smaller sections, different sub-carrier modulation schemes could be used (sub-carrier bit loading [12]), depending on the quality of each section of bandwidth, which makes OFDM efficient, flexible and adaptable to changing environments.

1.3 Software Defined Radio (SDR)

Software Defined Radio (SDR) is a communications architecture that aims to minimise the use of fixed hardware components in electronic circuits by moving all possible processing to a software domain. Such a system has the advantage of being highly flexible. A generic hardware platform could become any radio communications device, depending upon the software loaded onto the embedded processor, the RF front end, the sampling rate and the processor power available to the software.

The advantages of such systems are numerous and make it an ideal candidate for developing and testing a communications system. SDR systems are easily upgradeable, not needing any hardware changes, only a simple software upload, which in the future, could even be done “over the air” using appropriate software boot-loaders. Reusable software components in different projects has the ability of significantly reducing system design times, which means faster prototyping and faster time to market of final products.

Software Defined Radio is definitely a methodology that will become more popular in the future, and will increase in popularity as embedded processors and digital solid-state memory become more powerful and shrink in size.

1.4 Thesis Objectives

The objective of this thesis is to:

- Introduce the reader to OFDM modulation systems and its advantages and disadvantages over existing FDMA systems.
- Introduce the reader to existing digital modulation techniques and how it is related to OFDM modulation.
- Study the mathematics behind OFDM, study the encoding and decoding process, and develop means of digitally implementing such a system.
- Identifying and overcoming the main OFDM system performance influencing factors.
- Introducing the reader to the IEEE 802.11a standard of OFDM modulation.
- Implementing and IEEE 802.11a based OFDM system in software for use in a Software Defined Radio environment.
- Simulating the IEEE 802.11a OFDM system, measuring its performance and capturing vital results and statistics.
- Implementing a real-time IEEE 802.11a OFDM system, measuring its performance and capturing vital results and statistics.
- Comparing and critically analysing simulation and real-time OFDM performance results.
- Concluding on all findings and results and suggesting on further research in the area.

1.5 Thesis Overview

The structure of this thesis is as follows:

Chapter 1: Introduction

- A literature study into OFDM and SDR, its advantages and disadvantages, also stating the objectives of this thesis and a brief synopsis on the results and findings of the simulations and real-time implementation tests of the OFDM transceiver system.

Chapter 2: An Overview of Digital Modulation Techniques and Frequency Division Multiplexing

- An overview of existing digital modulation techniques, as well as how Frequency Division Multiplexing works. Introducing the fundamental encoding techniques, which forms the basis of OFDM encoding.

Chapter 3: OFDM Mathematics

- A detailed study of the mathematics behind OFDM, describing its workings in both the time domain and the frequency spectrum. The important issue of orthogonality is also discussed and proved.

Chapter 4: Encoding and Decoding OFDM Data Symbols

- A detailed look at the digital implementation of encoding and decoding single OFDM data symbols, and then expanding it to encode and decoded OFDM symbol trains.

Chapter 5: Identifying and Overcoming OFDM System Performance Influences (Problems due to real life factors)

- In practice, data transmission over analogue channels is prone to various performance influencing factors like noise, synchronisation, multipath interference, peak-to-average power ratio issues and communications channel estimation. Here we examine most of these factors, and introduce methods for overcoming them.

Chapter 6: IEEE 802.11a OFDM Standard

- The IEEE 802.11a standard incorporates OFDM modulation with some set parameters. It is a robust standard with the ability to overcome many of the major OFDM problems introduced by real-life factors. In this chapter the IEEE 802.11a standard is examined and its workings described in detail.

Chapter 7: SDR Implementation of an IEEE 802.11a OFDM System

- A working implementation of the IEEE 802.11a system is one of the main purposes of this thesis. This chapter fully describes the design decisions, how the software works, as well as the test platform it is tested on and the test program it is tested with. It also includes some recommendations for future implementations of the system.

Chapter 8: OFDM System Performance Tests and Results

- To determine how well the software works, it will need to be tested. In this chapter the OFDM software is tested, by first decoding simulated OFDM signals, influenced by noise and sampling frequency drift. Finally the real-time buffered implementation is tested. Vital performance statistics are captured and compared against theoretical values as well as other existing systems.

Chapter 9: Conclusions

- Concluding on OFDM and SDR, mentioning the most important facts about the system, like the 2.2 dB implementation loss and the current encoding and decoding times; concluding on the OFDM system performance and recommendations for future OFDM and SDR systems.

Appendix

- The appendices include some mathematical proofs used in the thesis as well as additional information about modulation techniques and the implemented OFDM system.

1.6 Synopsis on Thesis Results

OFDM is a very attractive modulation technique especially for office environments where high data rates are a necessity but where the relatively long multi-path delays and possible narrow-band interference could cause big problems for high-speed single carrier modulation techniques. OFDM also has its fair share of disadvantages and performance influencing factors that are addressed in this thesis. The implemented IEEE 802.11a transceiver system works well but has an approximate 2.2dB

implementation loss. The implementation loss is a mainly the result of noise on the OFDM reference sub-carriers and could be improved by better programming.

1.7 Conclusions

OFDM and SDR has been introduced as well as the plan to combine them and design an IEEE 802.11a OFDM transceiver system in software that can later be ported to SDR platforms. The thesis objectives have also been stated as well as a quick overview of the thesis as a whole. In the next chapter we will look at different digital modulation techniques, which forms the basis of OFDM modulation, as well as more detail on FDMA and how it relates to OFDM.

Chapter 2

An Overview of Digital Modulation Techniques and Frequency Division Multiplexing

2.1 Introduction

OFDM modulation is a multi-carrier modulation technique, which is based on existing digital modulation techniques. OFDM sub-carriers are orthogonal to each other, which means that, under normal circumstances, the sub-carriers do not influence each other in any way. The performance of an OFDM modulated system is thus directly influenced by the performance of the individual sub-carriers. Studying single carrier digital modulation techniques gives us a clearer picture of how a large OFDM system should perform. Furthermore OFDM is a special subset of a multiplexing methodology called Frequency Division Multiple Access (FDMA). Both OFDM and FDMA multiplex multiple sub-carriers in the frequency spectrum. In the case of FDMA systems, adjacent sub-carriers are placed far enough from each other to avoid their individual bandwidths from overlapping, which would result in inter-carrier interference, and degradation in the encoded data. Furthermore, because FDMA sub-carriers are usually decoded separately the distance between sub-carriers should be large enough to allow for the band-pass filtering of the selected sub-carrier; this distance is usually referred to as the guard band. In the case of OFDM systems, the bandwidths of adjacent sub-carriers are allowed to overlap, given that certain criteria are met which insure sub-carrier orthogonality. All the sub-carriers within an OFDM symbol are encoded and decoded together, which removes the need for a guard band, and increases bandwidth efficiency.

2.2 An Overview of Digital Modulation Techniques

The process of encoding digital bit data into an analogue representation of the data is usually referred to as “shift keying”. The most basic “shift keying” techniques encode digital data onto sinusoidal waveforms by manipulating the amplitude, frequency or phase of the sinusoidal waveform.

2.2.1. Amplitude-Shift Keying (ASK)

One of the most basic digital modulation techniques is called Amplitude-Shift Keying (ASK). ASK encodes digital data bits on to a sinusoidal carrier wave by manipulating the amplitude of the sinusoid.

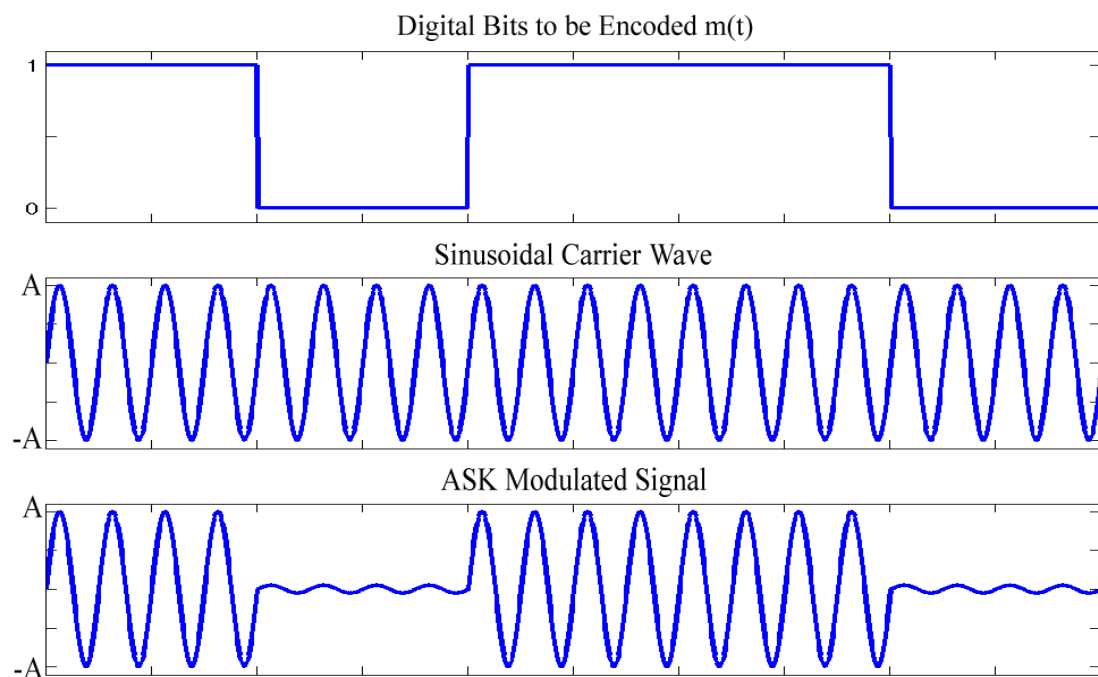


Figure 2.1: An ASK digitally modulated bit stream of $\{1,0,1,1,0\}$. The top figure is the digital data bits $m(t)$, the second figure is the sinusoidal carrier wave and the third figure is the ASK modulated signal.

The digital bits are represented either by a 1 or a 0 and can be expressed as

$$m(t) = \{0,1\}. \quad (2.1)$$

The ASK digitally modulated signal can be expressed as

$$s(t, f_c) = m(t)A \cos(2\pi f_c t + \phi) \quad (2.2)$$

or in a complex form as

$$s(t, f_c) = m(t)A \angle \phi(f_c) = m(t)A [\cos(2\pi f_c t + \phi) + i \sin(2\pi f_c t + \phi)] \quad (2.3)$$

2.2.2 Phase-Shift Keying (PSK)

Another more popular digital modulation technique is called Phase-Shift Keying (PSK). PSK encodes digital data bits onto a sinusoidal carrier wave by manipulating the phase of the sinusoid. PSK is a very popular technique because of its high immunity to noise. The most basic PSK technique is called Binary Phase-Shift Keying (BPSK). The digital bits are represented either by a 1 or a 0 and can be expressed as

$$m(t) = \{0, 1\}. \quad (2.4)$$

The digital bits are mapped to phase angles, and in the case of BPSK, the 0 is mapped to 0 radians and the 1 is mapped to π radians. The mapped bits can be expressed as

$$r(t) = \pi m(t) = \{0, \pi\}. \quad (2.5)$$

The BPSK digitally modulated signal can be expressed as

$$s(t, f_c) = A \cos(2\pi f_c t + r(t)) \quad (2.6)$$

or a in complex form as

$$s(t, f_c) = A \angle r(t)(f_c) = A [\cos(2\pi f_c t + r(t)) + i \sin(2\pi f_c t + r(t))]. \quad (2.7)$$

The same signal can also be expressed as a complex exponential as

$$s(t, f_c) = A \angle r(t)(f_c) = A e^{j2\pi f_c t + r(t)} \quad (2.8)$$

which then finally becomes

$$s(t, f_c) = (A e^{jr(t)}) e^{j2\pi f_c t} = C_{f_c} e^{j2\pi f_c t} \quad (2.9)$$

C_{f_c} is the complex coefficient that describes the amplitude and phase of the carrier wave at frequency f_c . This is a popular method of describing modulated signals and is also used in describing IEEE 802.11a OFDM signals. Refer to Sections 3.2.1, 6.2.2, 6.2.3 and 6.2.5.4 to see how the complex coefficients are implemented in OFDM modulation.

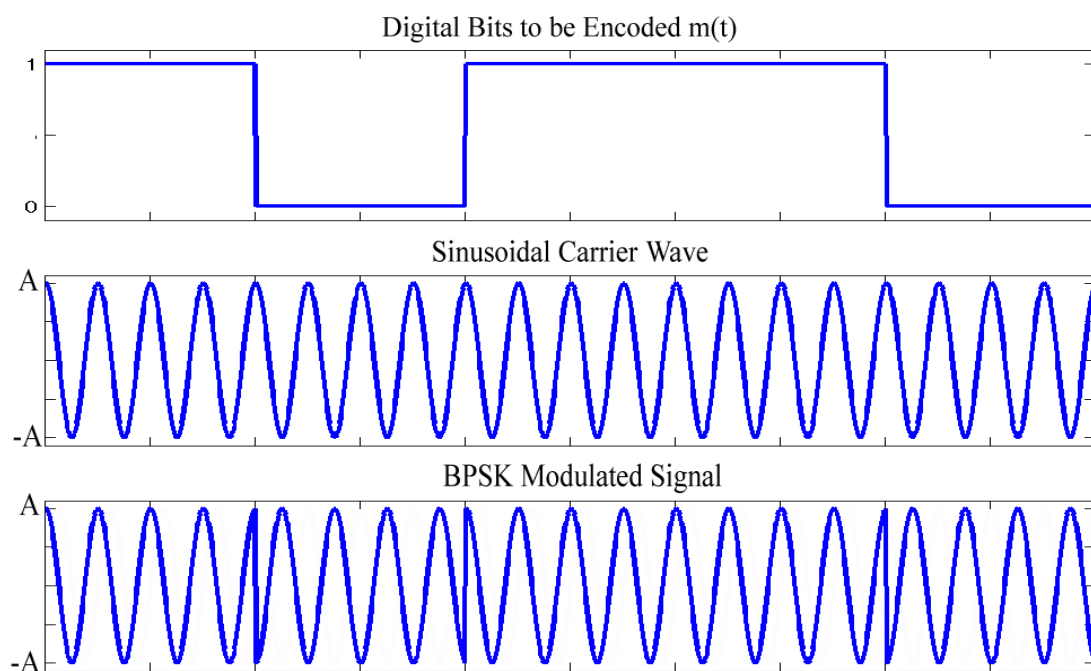


Figure 2.2: A BPSK digitally modulated bit stream of $\{1,0,1,1,0\}$. The top figure is the digital data bits $m(t)$, the second figure is the sinusoidal carrier wave and the third figure is the BPSK modulated signal.

It is possible to increase the encoding capabilities of PSK modulation schemes by mapping additional data bits to additional sinusoid phases. One of the most popular PSK methods is called Quadrature Phase-Shift Keying. It encodes two digital bits on to a sinusoidal carrier by using four possible phase angles. The digital bits it can encode is expressed by

$$m(t) = \{00, 01, 10, 11\}. \quad (2.10)$$

The resulting phase angles is given by

$$r(t) = \{\pi/4, 3\pi/4, 5\pi/4, 7\pi/4\}. \quad (2.11)$$

It is further possible to combine ASK and PSK modulation to produce a modulation technique called Quadrature Amplitude Modulation (QAM). QAM has the ability to encode data bits to various sinusoid phases and amplitudes.

Selecting an appropriate digital modulation scheme is a complicated matter. Many factors has to be taken into account, for example, transmitter complexity and power, receiver complexity and sensitivity, transmission channel quality as well as the effects that noise has on the quality of the modulated signals. Noise influences the modulation schemes in different ways. For the purpose of this thesis, only the QPSK modulation scheme is going to be examined further. For more information about all the different modulation schemes please refer to [7,8,9].

2.3 The Influence of Noise on Digital Modulation

Noise can be described as unwanted random (typically electromagnetic) interference that degrades the quality of signals. There exist two different types of noises.

Additive White Gaussian Noise (AWGN) is interference with a flat frequency spectrum [11]. This means that the long-term influence of AWGN is the same for all the frequency elements in a signal. The source of AWGN is radiation picked up from

radio transmissions and thermal noise picked up by hardware. The effects of AWGN can be estimated and quantified by means of statistical analysis.

Coloured noise is interference with a non-flat frequency spectrum. This means that the long-term influence of coloured noise is different for different frequency elements in a signal. The source of coloured noise could be many things like electrical interference from fast switching power supplies. The effects of coloured noise cannot be easily estimated and influences signal quality in unpredictable ways. Coloured noise will not be discussed in this thesis, since it is not a signal processing problem, but an electronic design problem, which can only be solved by analysing complete electronic system designs, and shielding electronic components from the noise sources.

2.3.1 Quantifying Additive White Gaussian Noise

AWGN signals are purely random and it is impossible to predict its instantaneous value at any time. Since noise has amplitudes that vary randomly with time, it can only be quantified by probability density functions. A probability density function relates a signal value with a probability of such a signal occurring. AWGN has a Gaussian probability density function, given by the formula [10],

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\bar{x})^2}{2\sigma^2}}. \quad (2.12)$$

The two main parameters of a Gaussian probability density function are:

- The mean or expected value of the noise, given by \bar{x} , which is zero for AWGN.
- The standard deviation of the noise, given by σ , is the RMS value of the noise signal.

Since instantaneous noise signals are totally random, it is impossible to predict the possibility of a certain discrete value occurring. It is possible though to calculate the possibility of a noise signal being in a certain range [a,b], and it is given by the probability function $P(x)$,

$$P(a < x < b) = \int_a^b p(x)dx = \int_a^b \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\bar{x})^2}{2\sigma^2}} dx. \quad (2.13)$$

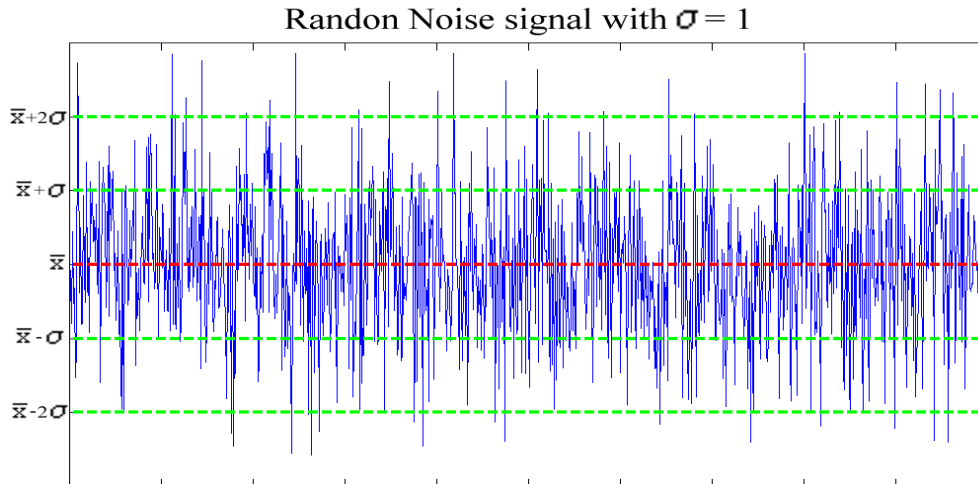


Figure 2.3: An example of a Gaussian Noise signal. (RMS = 1)

It is thus possible to calculate the possibility of a noise signal occurring with a certain range of values, if the RMS value of that noise signal is known.

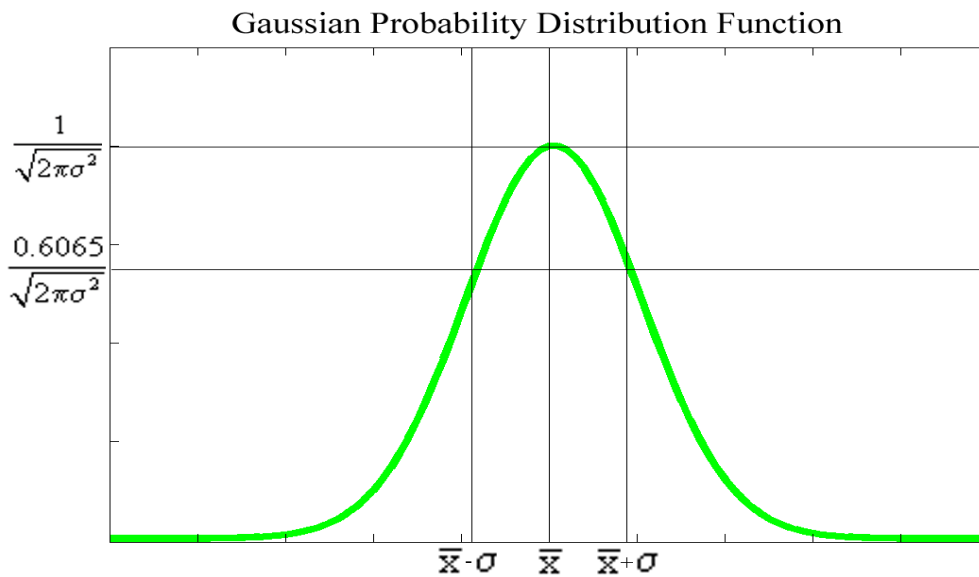
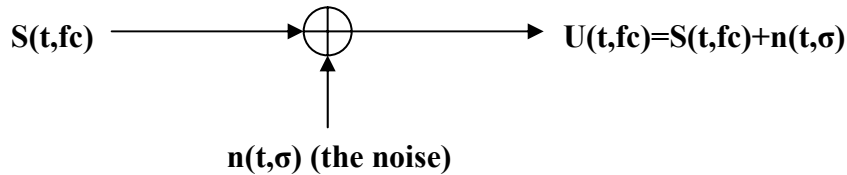


Figure 2.4: The Gaussian Probability Distribution Function.

2.3.2 Signal-To-Noise Ratio

AGWN is additive, which means that the noise signal adds to the existing signal, resulting in a distorted version of the original signal.



It is possible to determine the quality of a digitally modulated signal influenced by AWGN using the probability density function and the standard deviation of the noise signal. Signal quality is defined as the ratio [7] of signal power over noise power, called the Signal-to-Noise Ratio

$$\text{SNR} = \frac{\text{Signal Power}}{\text{Noise Power}} = \left(\frac{s_{\text{rms}}(t, fc)}{n_{\text{rms}}(t, \sigma)} \right)^2. \quad (2.14)$$

Digital modulation uses sinusoidal carrier waves, of which the RMS value can easily be calculated as $(1/\sqrt{2})A$ ¹. The Signal-to-Noise ratio (power) of a digitally modulated sinusoid influenced by AWGN, then becomes

$$\text{SNR} = \left(\frac{1}{\sqrt{2}} \right)^2 \left(\frac{A}{\sigma} \right)^2. \quad (2.15)$$

A popular way of expressing the Signal-to-Noise ratio, is by means of its decibel value,

$$\text{SNR}_{\text{dB}} = 10 \log_{10}(\text{SNR}) = 10 \log_{10} \left\{ \left(\frac{1}{\sqrt{2}} \right)^2 \left(\frac{A}{\sigma} \right)^2 \right\} = 20 \log_{10} \left\{ \left(\frac{1}{\sqrt{2}} \right) \left(\frac{A}{\sigma} \right) \right\}. \quad (2.16)$$

¹ Appendix A.3 (Calculating the RMS value of a sine wave)

2.3.3 Probability of Error in Quadrature Phase Shift-Keying Modulation

Noise degrades the quality of modulated signals. This ultimately leads to modulated signals being demodulated incorrectly and the decoded digital data bits being wrong. Because of the randomness of AWGN, it is impossible to predict the exact locations of incorrectly decoded bits; it is however possible to theoretically predict the amount of incorrectly decoded bits in the long run, and from that calculate error probabilities like the symbol-error rates and bit-error rates.

QPSK modulation encodes two data bits into a sinusoidal carrier wave by altering the sinusoidal carrier wave's phase. We can define the four separate QPSK symbols as

$$s_{11}(t, f_C) \square A \cos(2\pi f_C t + \pi/4); \quad \text{if } m(t)=11, \quad (2.17)$$

$$s_{01}(t, f_C) \square A \cos(2\pi f_C t + 3\pi/4); \quad \text{if } m(t)=01, \quad (2.18)$$

$$s_{00}(t, f_C) \square A \cos(2\pi f_C t + 5\pi/4); \quad \text{if } m(t)=00, \quad (2.19)$$

$$s_{10}(t, f_C) \square A \cos(2\pi f_C t + 7\pi/4); \quad \text{if } m(t)=10. \quad (2.20)$$

If these symbols are then subjected to noise, the resultant noisy symbols can be expressed as

$$u_{11}(t, f_C) = u_{11}(t, f_C) + n(t, \sigma), \quad (2.21)$$

$$u_{01}(t, f_C) = u_{01}(t, f_C) + n(t, \sigma), \quad (2.22)$$

$$u_{00}(t, f_C) = u_{00}(t, f_C) + n(t, \sigma), \quad (2.23)$$

$$u_{10}(t, f_C) = u_{10}(t, f_C) + n(t, \sigma). \quad (2.24)$$

To clearly see the effects of the noise on the modulated symbols, it is useful to display the symbols and the noise on a constellation diagram. A constellation diagram transforms sinusoidal symbols into their amplitude and phase components. Figure 2.5 shows the QPSK constellation diagram with probability density functions showing the effects of noise at each symbol. A QPSK decoder will decode the digital data bits depending upon which quadrant the received symbol is located. It is possible for noise

to add to a symbol and corrupt its amplitude so much that it ends up in an incorrect quadrant, and thus decoding the data bits incorrectly. This is called a symbol error.

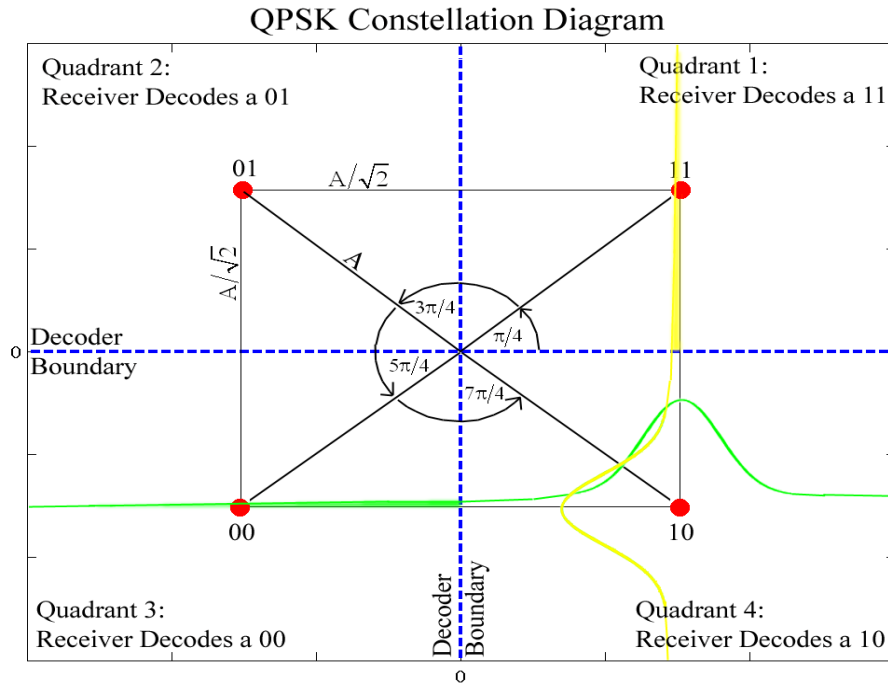


Figure 2.5: A constellation diagram of a QPSK encoded signal with AWGN

In order to calculate symbol error probabilities we first view the problem in **one dimension**. The probability that a QPSK decoder will incorrectly decode a symbol u_{00} given that the correct transmitted symbol was in fact a s_{10} is given by the formula

$$P(u_{00} | s_{10}) = P(-\infty < x < 0) = \int_{-\infty}^0 \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-A/\sqrt{2})^2}{2\sigma^2}} dx. \quad (2.25)$$

The probability function then reduces to

$$P(u_{00} | s_{10}) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-A/\sqrt{2}\sigma} e^{-\frac{(y)^2}{2}} dy. \quad (2.26)$$

Unfortunately this function is not directly solvable and lookup tables are used to determine the results. There exists a function, though, that is closely related to the above probability function. The Q-function [10], is defined as

$$Q(x) = \int_x^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx. \quad (2.27)$$

We can then rewrite the probability function (2.26) in terms of the Q-function defined in (2.27) so that

$$P(u_{00} | s_{10}) = Q\left(\frac{A}{\sqrt{2}\sigma}\right). \quad (2.28)$$

Using the same procedure it is possible to calculate the probability that the QPSK decoder will incorrectly decode a symbol u_{11} given that the correct transmitted symbol was in fact a s_{10} as

$$P(u_{11} | s_{10}) = Q\left(\frac{A}{\sqrt{2}\sigma}\right). \quad (2.29)$$

Furthermore there is also the remote possibility that the QPSK decoder will incorrectly decode a symbol u_{01} given that the correct transmitted symbol was in fact a s_{10} . This is however only possible when the noise signal is large enough so that in the horizontal dimension s_{10} was incorrectly decoded as u_{00} and in the vertical dimension s_{10} was incorrectly decoded as u_{11} . The probability of a symbol s_{10} to be incorrectly decoded as a u_{01} is thus given by

$$P(u_{01} | s_{10}) = P(u_{00} | s_{10})P(u_{11} | s_{10}) \quad (2.30)$$

which gives

$$P(u_{01} | s_{10}) = Q^2\left(\frac{+A}{\sqrt{2}\sigma}\right). \quad (2.31)$$

Finally it is possible to calculate the total probability that a symbol s_{10} will be decoded incorrectly as the probability that s_{10} will either be incorrectly decoded as a u_{11} symbol, a u_{00} symbol or a u_{01} symbol. In cases like this, it is easier to calculate error probabilities through the use of correctness probabilities, which is given by

$$P_C = 1 - P. \quad (2.32)$$

The probability that a symbol s_{10} will be decoded correctly as a symbol u_{10} is thus the product of symbol s_{10} being decoded correctly in terms of the other three quadrants. This is given by

$$P_C(s_{10}) = P_C(u_{00} | s_{10})P_C(u_{11} | s_{10})P_C(u_{01} | s_{10}) \quad (2.33)$$

which becomes

$$P_C(s_{10}) = (1 - P(u_{00} | s_{10}))(1 - P(u_{11} | s_{10}))(1 - P(u_{01} | s_{10})) \quad (2.34)$$

and can then be written in terms of the Q-function as

$$P_C(s_{10}) = \left[1 - Q\left(\frac{A}{\sqrt{2}\sigma}\right) \right] \left[1 - Q\left(\frac{A}{\sqrt{2}\sigma}\right) \right] \left[1 - Q^2\left(\frac{A}{\sqrt{2}\sigma}\right) \right] \quad (2.35)$$

which then finally reduces to

$$P_C(s_{10}) = 1 - 2Q\left(\frac{A}{\sqrt{2}\sigma}\right) + 2Q^3\left(\frac{A}{\sqrt{2}\sigma}\right) - Q^4\left(\frac{A}{\sqrt{2}\sigma}\right). \quad (2.36)$$

Converting to error probability, the probability for the single symbol error in a QPSK decoder is given as

$$P(u_{10} | s_{10}) \approx 2Q\left(\frac{A}{\sqrt{2}\sigma}\right). \quad (2.37)$$

Using the above procedures, it is possible to prove that the symbol error probability for the other combinations is exactly the same:

$$P(u_{00} | s_{00}) = P(u_{01} | s_{01}) = P(u_{10} | s_{10}) = P(u_{11} | s_{11}) \approx 2Q\left(\frac{A}{\sqrt{2}\sigma}\right). \quad (2.38)$$

The total symbol error rate (SER) of a QPSK decoder can finally be calculated as the average symbol error probability of $P(u_{00} | s_{00})$, $P(u_{01} | s_{01})$, $P(u_{10} | s_{10})$ and $P(u_{11} | s_{11})$. The SER thus becomes

$$\begin{aligned} \text{SER} = & P(u_{00} | s_{00})P(s_{00}) + P(u_{01} | s_{01})P(s_{01}) \\ & + P(u_{10} | s_{10})P(s_{10}) + P(u_{11} | s_{11})P(s_{11}) \end{aligned} \quad (2.39)$$

which then becomes

$$\text{SER} = \left[2Q\left(\frac{A}{\sqrt{2}\sigma}\right) \right] [P(u_{00}) + P(u_{01}) + P(u_{10}) + P(u_{11})]. \quad (2.40)$$

It is impossible to calculate the probability of each symbol occurring, but because we know the QPSK decoder will always choose one of the possible four quadrants, the sum of the probabilities are equal to one.

$$[P(u_{00}) + P(u_{01}) + P(u_{10}) + P(u_{11})] = 1. \quad (2.41)$$

The symbol error rate for a QPSK decoder is thus

$$\text{SER} = 2Q\left(\frac{A}{\sqrt{2}\sigma}\right). \quad (2.42)$$

A more useful form of expressing error rates is by using the QPSK Bit Error Rate. A QPSK symbol encodes two digital data bits, but because of the way the digital data

bits are located in the constellation, a single QPSK symbol error, will in most cases result in only one bit error. The bit error rate (BER) is thus only half of the SER for QPSK decoders. BER gives the estimated amount of erroneous decoded bits due to noise. The Bit Error Rate for a QPSK decoder is thus

$$\text{BER} = Q\left(\frac{A}{\sqrt{2}\sigma}\right). \quad (2.43)$$

The symbol error rates and bit error rates can further be expressed as a function of the Signal-to-Noise Ratio, so that

$$\text{SER} = 2Q\left(\sqrt{\text{SNR}}\right) \quad (2.44)$$

and

$$\text{BER} = Q\left(\sqrt{\text{SNR}}\right). \quad (2.45)$$

The SER and BER can be rewritten in another perhaps more popular method of describing the SNR, through the use of the bit energy and the noise power density [8,9]. Firstly the noise power density is equal to the noise power divided by the bandwidth it occupies,

$$N_0 = \frac{N}{B_n}. \quad (2.46)$$

The symbol energy, E_s , of the symbol is equal to the symbol power, C , divided by the symbol rate, R_s , so that,

$$E_s = \frac{C}{R_s}. \quad (2.47)$$

The symbol energy to noise power density ratio, can be calculated as

$$\frac{E_s}{N_0} = \frac{C}{R_s} \frac{B_n}{N}. \quad (2.48)$$

As already mentioned the bit rate of a QPSK encoded signal is twice the symbol rate, because each symbol can encode 2 bits. Equation (2.48) can thus be rewritten in terms of bit energy, E_B , and bit rate, R_B , to give the bit energy to noise power density ratio, also known as the E_b/N_0 ratio [8]:

$$\frac{E_B}{N_0} = \frac{1}{2} \frac{C}{R_B} \frac{B_n}{N}. \quad (2.49)$$

The bit rate is equal to twice the signal rate, which is also the Nyquist sampling rate of the signal. If the receiving filter filters the signal at this frequency, then the noise bandwidth is equal to the bit rate so that $R_B = B_N$. Equation (2.49) can thus be rewritten as

$$\frac{E_B}{N_0} = \frac{1}{2} \frac{C}{N}. \quad (2.50)$$

The power of a sine wave is equal to the square of the RMS amplitude of the sine wave. Thus the symbol power becomes

$$C = A_{\text{RMS}}^2. \quad (2.51)$$

In the same way, the power of an AWGN signal is the square of the RMS of the noise, which is also the standard deviation of the AWGN,

$$N = n_{\text{RMS}}^2 = \sigma^2. \quad (2.52)$$

Equation (2.50) can now be rewritten in terms of (2.51) and (2.52) as

$$\frac{E_B}{N_0} = \frac{1}{2} \frac{A_{\text{RMS}}^2}{\sigma^2}. \quad (2.53)$$

Equation (2.53) can now be rewritten in terms of the SNR power ratio from Equation (2.15) to give

$$\text{SNR} = \frac{2E_B}{N_0}. \quad (2.54)$$

This means that the SER and BER for QPSK signals can now be rewritten in terms of the E_b/N_0 ratio as [9]:

$$\text{SER} = 2Q\left(\sqrt{\frac{2E_B}{N_0}}\right) \quad (2.55)$$

and

$$\text{BER} = Q\left(\sqrt{\frac{2E_B}{N_0}}\right). \quad (2.56)$$

Equation (2.55) and (2.56) corresponds to the results for probability of error binary modulation. [8,9]

Figure 2.6 shows the graph of the error probabilities versus the SNR values.

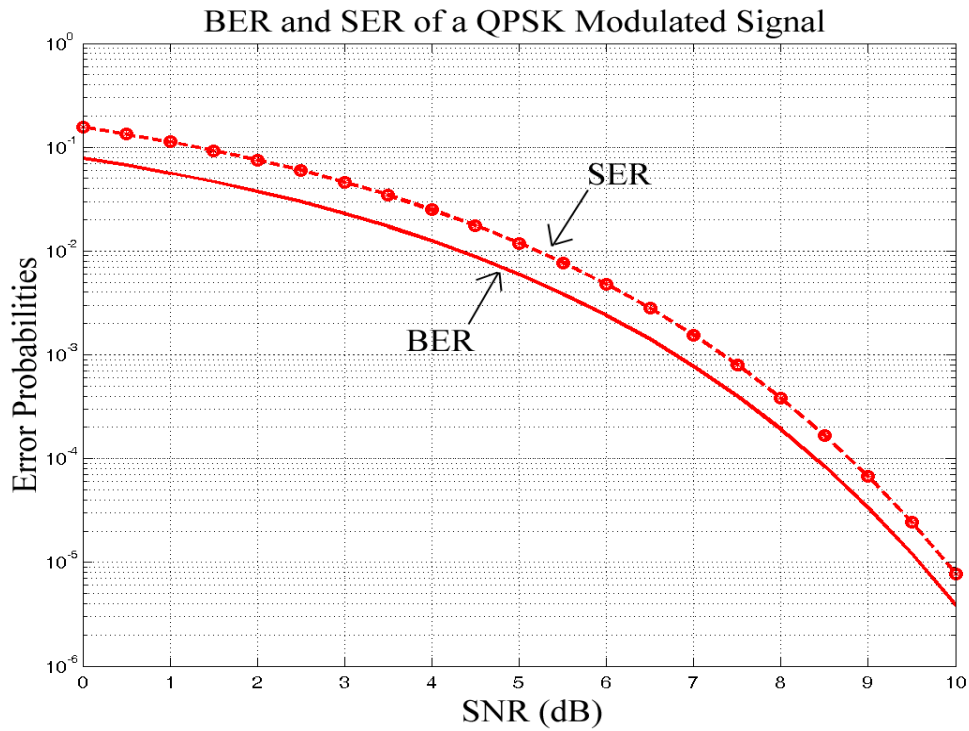


Figure 2.6: BER and SER vs. SNR of a QPSK modulated signal.

Table 2.1 shows the error probabilities versus the SNR values

Signal-to-Noise Ratio (dB)	Bit Error Rate (BER)	Symbol Error Rate (SER)
0	7,86e-2	1,57e-1
1	5,63e-2	1,126e-1
2	3,75e-2	7,5e-2
3	2,3e-2	4,16e-2
4	1,25e-2	2,5e-2
5	6,0e-3	1,2e-2
6	2,4e-3	4,8e-3
7	7,7e-4	1,5e-3
8	1,9e-4	3,8e-4
9	3,36e-5	6,73e-5
10	3,87e-6	7,74e-6

Table 2.1: BER and SER vs. SNR of a QPSK modulated signal

From these results it is clear to see that noise degrades the quality of transmitted signals and causes errors the transmitted digital data. These values are theoretical; a practical system might not be able to achieve these rates, but should always strive to be as close as possible.

2.4 An Overview of Frequency Division Multiplexing

Frequency Division Multiple Access (FDMA) is a methodology that enables multiple signals to simultaneously co-exist in the frequency spectrum, by being placed at non-overlapping frequencies. The original signals are modulated and up-converted at carrier frequencies high enough to be transmitted as electromagnetic waves. In order to decode the data in single FDMA sub-carriers, all the adjacent sub-carriers need to be removed in order to minimise cross-band interference. Bandpass filtering of the selected sub-carrier removes all other sub-carriers. Practical filters cannot have infinitely sharp cut-offs.

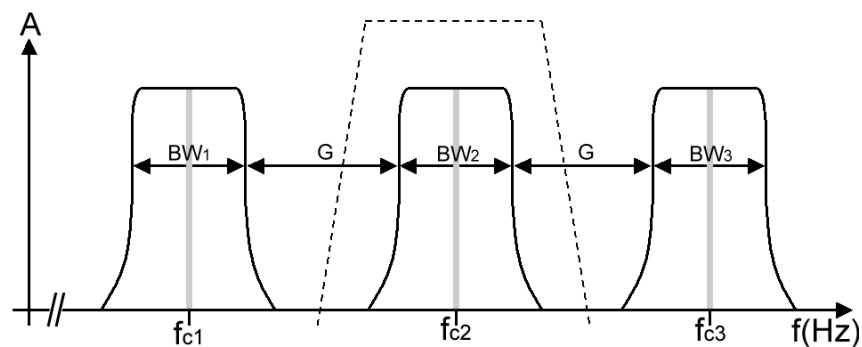


Figure 2.7: Example of FDMA modulated signals, and their guard-bands (G)

This means that the edges of the filter, at which point the filter starts attenuating undesired signals, will not falloff instantaneously. The filters need some bandwidth to falloff and attenuate out of band signals sufficiently. This means that adjacent FDMA sub-carriers can't be placed exactly next to each other; some space between them is needed for the filters. These spaces between sub-carriers are called guard-bands. Guard-bands use up frequency space that could be used for extra sub-carriers, thus reducing bandwidth efficiency.

FDMA is used in many applications, including FM radio transmission, Digital European Cordless Telephones (DECT) and Advanced Mobile Phone System (AMPS) cell phones.

2.5 Conclusions

This chapter introduced some basic digital modulation schemes, which forms the basis of OFDM modulation, including QPSK, which is used in the practical implementation of the OFDM system. The important issue of noise and digital modulation error probabilities were examined up to a point where the theoretical bit- and symbol error rates were calculated. These values will be used later in comparisons with the OFDM system, and to determine its efficiency. Furthermore, OFDM is just a special case of FDMA, and the look at FDMA provides useful insights and shows the advantages of OFDM over FDMA. In the next chapter we look at the mathematics behind OFDM in order to understand it better and to see how it relates to single carrier digital modulation techniques, as the ones examined in this chapter.

Chapter 3

OFDM Mathematics

3.1 Introduction

Before digitally implementing an OFDM system, it is important to study the mathematics behind OFDM systems. An in-depth study of the mathematics will show how digital data bits are encoded into OFDM symbols, and how the digital data bits can be decoded from them as well. The issue of orthogonality will also be discussed and proved in this chapter.

3.2 A Mathematical Approach to OFDM Symbols

An OFDM symbol can be described as the result of a number of summated, digitally modulated sub-carriers at certain very specific frequencies, multiplied with a rectangular window of a very specific length. To completely understand the inner workings of OFDM, it should be studied in both the time domain and the frequency domain.

3.2.1. Time-Domain Analysis of OFDM Symbols

The rectangular window function has an amplitude of A_w , a width of t_w and is also shifted by t_w seconds. The rectangular window is given by

$$w(t) = A_w \Pi\left(\frac{t-t_w}{t_w}\right). \quad (3.1)$$

The sub-carriers all have the same amplitude of A . The summed sub-carriers are given by

$$c(t) = \sum_{c=0}^{C-1} [AS_c(t, f_c)], \quad (3.2)$$

where $S_c(t, f_c)$ is a digitally modulated sine wave as described in Section 2.2.2. The time-domain expression for an OFDM symbol is simply the product of the rectangular windowing function and the sub-carriers, see Figure 3.1, and is given by

$$o_c(t) = w(t)c(t) = \left[\sum_{c=0}^{C-1} [AS_c(t, f_c)] \right] A_w \Pi \left(\frac{t-t_w}{t_w} \right). \quad (3.3)$$

Equation (3.3) can be rewritten so that the sub-carriers can be expressed as complex exponentials as determined in (2.9)

$$o_c(t) = \left[\sum_{c=0}^{C-1} (AC_c e^{-j 2 \pi f_c t}) \right] A_w \Pi \left(\frac{t-t_w}{t_w} \right). \quad (3.4)$$

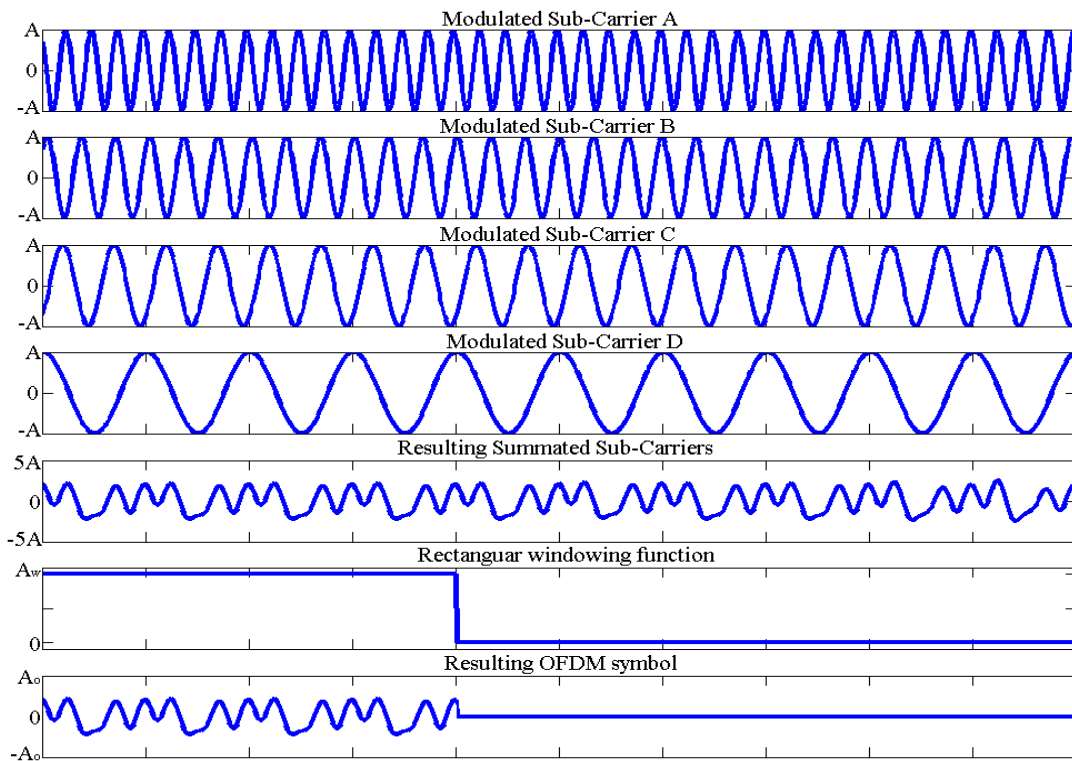


Figure 3.1: A real time-domain OFDM signal: The top four figures are modulated sub-carriers, the fifth figure is the resulting summated sub-carriers, $c(t)$, the sixth figure is the rectangular windowing function, $w(t)$, and the last image is the Resulting OFDM symbol $o_c(t)$.

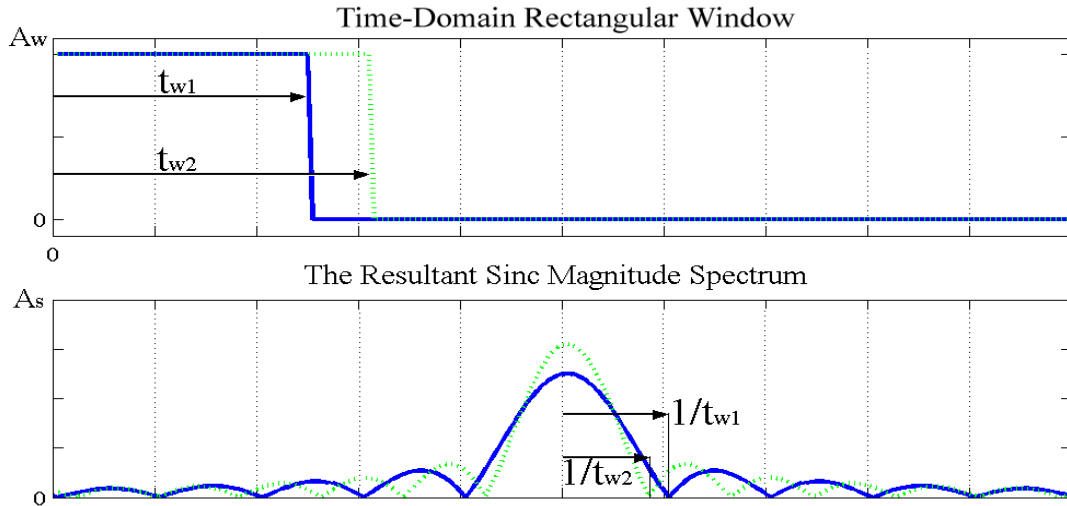


Figure 3.2: The Fourier Transform of the rectangular window: The top figure is the rectangular window. The bottom is the sinc waveform (magnitude spectrum), and the effect of the rectangular window length on the shape of the sinc waveform.

3.2.2. Frequency-Domain Analysis of OFDM Symbols

The spectrum of the OFDM symbol is obtained by using the Fourier Transform:

$$O_c(f) = \int_{-\infty}^{\infty} o_c(t) e^{-j 2 \pi f t} dt. \quad (3.5)$$

By expressing the OFDM symbol in the frequency spectrum, the role of the sub-carrier frequency positions and the rectangular window length becomes much clearer. They are the key variables upon which OFDM orthogonality depends. The Fourier Transform of the time-domain OFDM symbol transforms the modulated sub-carriers into phase carrying impulses in the frequency spectrum, while the rectangular window is transformed into a sinc function. Multiplication of the modulated sub-carriers with the rectangular window in the time domain results in a convolution of the impulses with the sinc function in the frequency spectrum. The final result of the convolution is sinc functions placed at all the positions of the impulses. A sinc waveform is defined as $\text{sinc}(ft) = \frac{\sin(\pi ft)}{\pi ft}$ and shown in Figure 3.2.

The convolution operator is given by the $*$ symbol. The magnitude spectrum of the OFDM symbol is calculated as

$$|O_c(f)| = \left| \sum_{c=0}^{C-1} (A\delta(f + f_c)) * A_w t_w \text{sinc}(f t_w) \right| . \quad (3.6)$$

The magnitude spectrum of the OFDM symbol can thus be simplified to give

$$|O_c(f)| = \left| \sum_{c=0}^{C-1} (A A_w t_w \text{sinc}(t_w (f + f_c))) \right| . \quad (3.7)$$

Equation (3.7) can be described as a summation of sinc-waveforms at the original impulse locations.

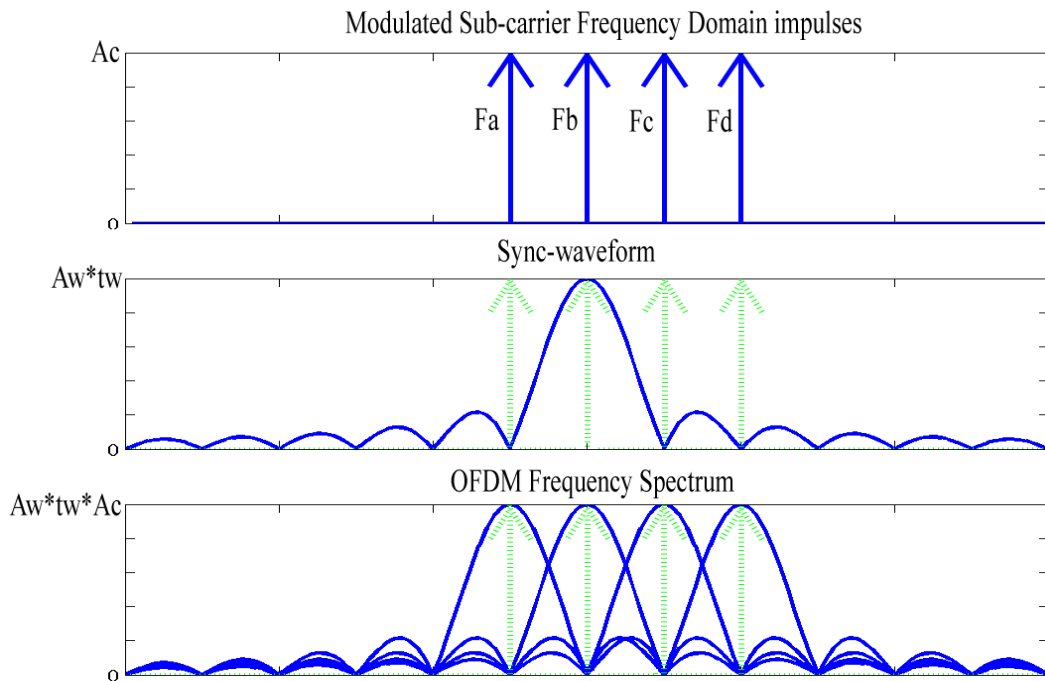


Figure 3.3: An OFDM magnitude spectrum. The top figure is the modulated sub-carrier impulses (time-unlimited), the second figure is the resulting sync-waveform and the last figure, is (time-limited) OFDM symbol spectrum, showing the overlapping bandwidths.

A closer look at the sinc function shows that it contains periodic zeroes, which is a function of the window length and can easily be determined as²

$$\text{sinc}(f_{\text{sinc}} t_w) = \begin{cases} 0; & \text{where } f_{\text{sinc}} = \frac{m}{t_w} ; m \in [1, 2, \dots, \infty] \\ 1; & \text{where } f_{\text{sinc}} = 0 \end{cases} . \quad (3.8)$$

Since all the modulated sub-carriers are windowed using the same rectangular window, the sinc-waveform for each of them will be exactly the same. By choosing

$$f_c = 1/t_w \quad (3.9)$$

it is possible to shape the sinc functions so that the zero-points from all the sub-carriers align, to give a spectrum as in Figure 3.3. The sub-carrier impulses are located at the middle of each sinc-waveform, where the value of the sinc is only one, which means that the sub-carrier information stays unchanged. The OFDM sub-carriers are therefore orthogonal to each other at each sub-carrier frequency.

3.3 The Issue of Orthogonality

From Figure 3.3 it seems that the sub-carriers are orthogonal, when choosing the inter-sub-carrier frequency spacing equal to the reciprocal of the rectangular window length, but it is important to prove that this is true for all possible values of f_c and t_w .

To prove that two arbitrary signals are mathematically orthogonal, the integral of the product of the arbitrary signals needs to be zero:

$$\int_{-\infty}^{\infty} w_1(x) \cdot w_2(x) dx = 0 . \quad (3.10)$$

² Appendix A.2: Finding the zeroes of a sinc waveform

We can now replace signal $w_1(x)$ with the sinc waveform and $w_2(x)$ with a train of equally spaced impulses representing the modulated sub-carriers at the locations of the zero-points of the sinc waveform.

$$\int_{-\infty}^{\infty} \text{sinc}(t_w x) \delta(x - \frac{m}{t_w}) dx \quad ; \quad m \in [1, 2, \dots, \infty]. \quad (3.11)$$

According to [10] Appendix A, the integral of any continuous arbitrary signal with an impulse, will result in the arbitrary function being evaluated at the location of the impulse, so that

$$\int_{-\infty}^{\infty} \phi(x) \delta(x - x_0) dx = \phi(x_0) \text{ where } \phi(x) \text{ is the arbitrary function.} \quad (3.12)$$

This means that (3.11) will reduce to the sinc values at all the impulse locations, so that

$$\int_{-\infty}^{\infty} \text{sinc}(t_w x) \delta(x - \frac{m}{t_w}) dx = \text{sinc}(t_w \frac{m}{t_w}) \quad ; \quad m \in [1, 2, \dots, \infty]. \quad (3.13)$$

According to the results from Appendix A.2 these impulse locations are now located over the sinc zeroes for all the integer values of m between one and infinity

$$\int_{-\infty}^{\infty} \text{sinc}(t_w x) \delta(x - \frac{m}{t_w}) dx = 0 \quad ; \quad m \in [1, 2, \dots, \infty]. \quad (3.14)$$

Thus the integral $\int_{-\infty}^{\infty} \text{sinc}(t_w x) \delta(x - \frac{m}{t_w}) dx$ is zero for all $m \in [1, 2, \dots, \infty]$, which means that all the impulses are orthogonal to the sinc waveform.

This means that OFDM sub-carriers are in fact orthogonal to each other, and that even though the individual sub-carrier bandwidths overlap with each other, they do not (under normal circumstances) interfere with each other.

3.4 Conclusions

In this chapter the mathematics behind OFDM symbols were investigated, and complete mathematical representations of the OFDM symbols created, both in the time domain and the frequency spectrum. The important issue of orthogonality was introduced and sub-carrier orthogonality proved. In the next chapter the discrete-time domain is introduced as well as the reasons why it is used. Methods to convert continuous-time signals, like the mathematical expression from this chapter into discrete-time signals are also determined. It will thus be possible to start implementing OFDM encoders and decoders in a digital environment based on the mathematics derived in this chapter.

Chapter 4

Encoding and decoding OFDM data symbols

4.1 Introduction

The encoding and decoding of single OFDM data symbols is a relatively easy process. Digital data is modulated/mapped onto OFDM sub-carriers in the frequency spectrum and then converted to the time domain (using the Inverse Fourier Transform) for transmission over the communications channel. OFDM decoding follows the same process but just in reverse: the received OFDM symbol is converted to the frequency-domain using a Fourier Transform, and the sub-carriers are then demodulated to retrieve the original digital data. The digital implementation of an OFDM encoder and decoder differs somewhat from the methods described in the mathematical analysis, in that such an encoder and decoder resides in a digital environment and not in a continuous-time domain. It is thus important to first study how digital systems receive and process continuous-time signals, before we start describing the OFDM encoding and decoding processes.

4.2 Analogue and Digital Domains

Real-life signals or analogue signals are also called continuous-time signals and have continuous amplitude and time axes, which means that there is infinitely many points along both axes. Digital systems have only finite memory, and can thus only store signals in a discrete fashion. Digital systems convert continuous-time domain signals to discrete digital signals using devices called Analogue-to-Digital Converters (ADC). ADCs sample the incoming continuous-time signal at a constant rate, called the sampling frequency (F_s), and convert it to digital values, which the system can process and demodulate.

The sampling frequency is used to map the continuous-time signal to its discrete-time signal using the formula

$$x(n) = x(t_n); \quad t_n = \frac{n}{F_s}. \quad (4.1)$$

The amount of samples a certain continuous-time domain signal will occupy is calculated by

$$N = (\text{signal time})(\text{sampling rate}) = t_{\text{Signal}} F_s. \quad (4.2)$$

To convert discrete-time domain signals back to the continuous-time again we use devices called Digital-to-Analogue Converters (DAC) and lowpass (reconstruction) filters.

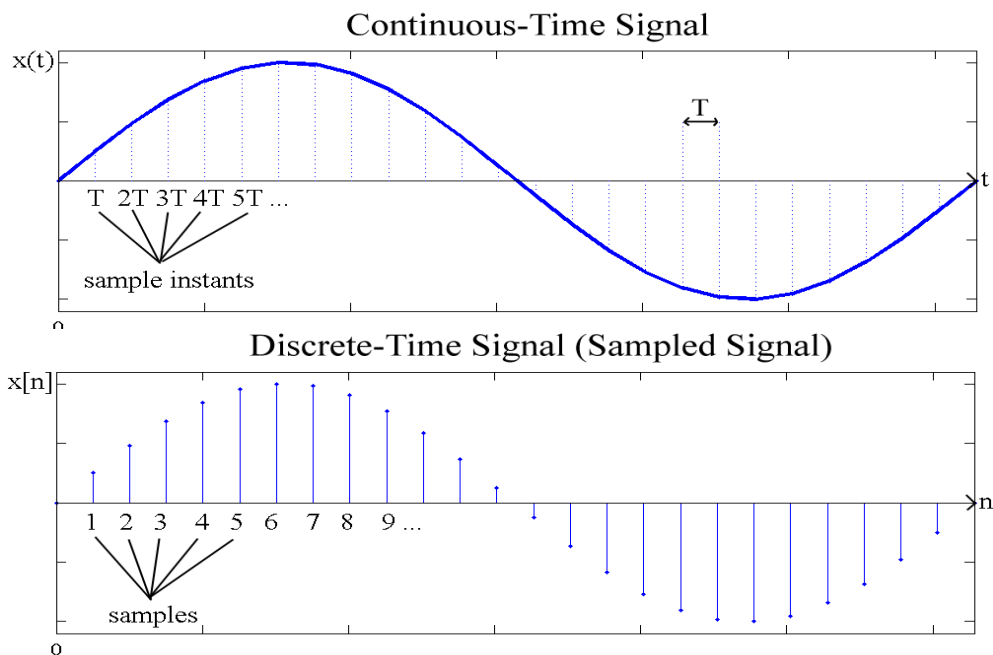


Figure 4.1: The conversion of a continuous-time signal to a discrete-time signal. The top figure is a continuous-time sinusoidal signal, showing the sampling instants. The bottom figure is the equivalent discrete-time signal of the sinusoid.

The discrete-time domain acts very much like the continuous-time domain, which is good, because it means that all the Fourier transform pairs can be modified to work for both these domains. There are a few major differences of which one needs to be aware though. One very important difference between the continuous-time and discrete-time domain is that signal frequencies are always written relative to the sampling frequency, using the relationship

$$\frac{f_c}{F_s} = \frac{k}{N}. \quad (4.3)$$

In the continuous-time domain, f_c is the signal frequency and F_s the sampling frequency. In the discrete-time domain, N is the amount of samples in the signal and $\frac{k}{N}$ is the normalised discrete-time frequency of that signal. Another more obvious difference between the continuous-time domain and the discrete-time domain is that signals in the continuous-time use the “time” parameter, t , to describe signals, where discrete-time signals use the “sample” parameter, n , to describe signals. As an example, the continuous-time signal

$$s(t, f_c) = A \cos(2\pi f_c t + r(t)), \quad (4.4)$$

becomes the discrete-time signal

$$s(n, k) = A \cos\left(2\pi \frac{k}{N} n + r(n)\right). \quad (4.5)$$

One other difference between the continuous-time domain and the discrete-time domain is that an integral in the continuous-time domain becomes a summation in the discrete-time domain. For instance the Fourier Transform in the continuous-time domain:

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j 2\pi f t} dt, \quad (4.6)$$

becomes a Discrete Fourier Transform in the discrete-time domain

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j 2 \pi \frac{k}{N} n} ; \quad (0 < k < N). \quad (4.7)$$

It is thus possible to implement an OFDM encoder and decoder in a digital environment, still based on the original OFDM mathematics, by using the relationships between the continuous-time and the discrete-time domain as shown.

4.3 Encoding OFDM Symbols

Encoding OFDM symbols refers to the process of mapping and modulating digital data bits into an OFDM symbol. For this thesis all OFDM sub-carriers will be modulated using QPSK modulation. Some of the parameters of the encoding process must be calculated beforehand.

The amount of samples needed to store an OFDM symbol is calculated as

$$N_{\text{OFDM}} = (\text{rectangular window time})(\text{sampling rate}) = t_w F_s. \quad (4.8)$$

The amount of QPSK sub-carriers within a single OFDM modulated symbol is N_{SD} and the digital bit capacity of a single QPSK modulated symbol is C_{QPSK} . The digital bit capacity of a single OFDM modulated symbol can be calculated as:

$$C_{\text{OFDM}} = C_{\text{QPSK}_1} + C_{\text{QPSK}_2} + C_{\text{QPSK}_3} + \dots + C_{\text{QPSK}_{N_{\text{SD}}}}, \quad (4.9)$$

which then becomes

$$C_{\text{OFDM}} = (C_{\text{QPSK}})(N_{\text{SD}}). \quad (4.10)$$

The digital data bits for encoding is $B_{\text{IN}}[1 \dots C_{\text{OFDM}}]$.

4.3.1 Encoding Single OFDM Symbols

As already mentioned, the process of encoding OFDM symbols is done in the frequency domain. OFDM symbols with large amounts of sub-carriers could theoretically modulate each sub-carrier separately and add the results from each modulator together, to create an OFDM symbol, but there is an easier way. In the frequency domain, digital data bits are mapped to magnitudes and phase values, and placed at the correct positions in the frequency spectrum. Taking the IDFT of the frequency spectrum will produce the time domain signal. This time domain signal is then equivalent to encoding each sub-carrier separately, but it is much faster and easier. Figure 4.2 illustrates the process of encoding a single OFDM symbol.

Single OFDM symbol Encoding

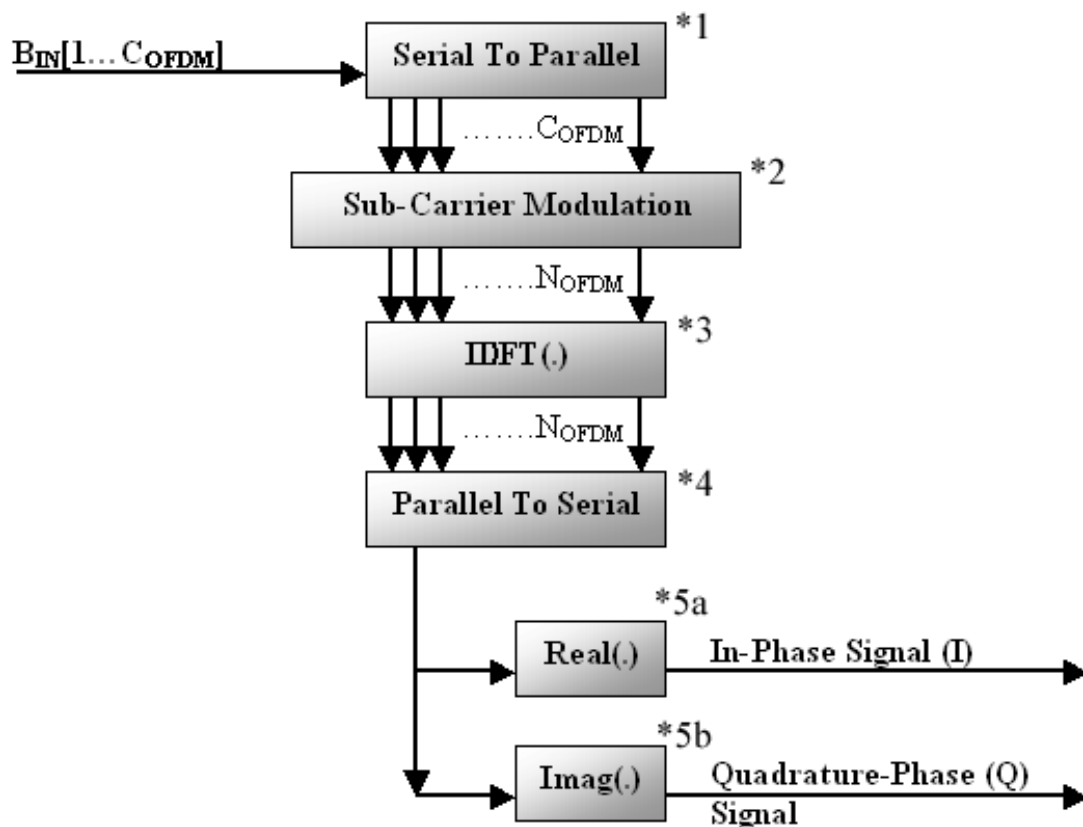


Figure 4.2: A graphical example of the single OFDM symbol encoding process.

Table 4.1 explains the OFDM symbol encoding process.

Nr	OFDM symbol encoding process
*1	First the serial stream of input digital data bits are converted to a parallel stream, so that they can all be processed together.
*2	Secondly the digital data bits are mapped to Amplitude and Phase values and placed at the correct locations in the frequency spectrum (The Modulation Process)
*3	Thirdly the IDFT converts the frequency spectrum to a complex time domain signal, which can be transmitted.
*4	Fourthly the parallel complex time domain signal is reconverted back to a serial stream.
*5a, *5b	Finally the real part of the complex time domain signal becomes the In-Phase Signal a.k.a. the I-channel signal, and the imaginary part becomes the Quadrature-Phase Signal a.k.a. the Q-channel signal
Output	The final I- & Q-channel signals are then really to be transmitted over the communications channel.

Table 4.1: The OFDM symbol encoding process

4.3.2 Encoding Multiple OFDM Symbols

OFDM symbols have the ability to encode many digital data bits, but the capacity of an OFDM symbol is still finite. The digital data capacity of each OFDM symbol is limited to the amount of sub-carriers within each symbol, as well as the amount of digital data bits encoded within each sub-carrier. In a practical system, one would often want to transfer large amount of data across the communications channel. If the amount of digital data bits is more than the capacity of a single OFDM symbol, the digital data is simply encoded across many OFDM symbols and transmitted over the channel. This is called an OFDM symbol train and can also be referred to as the OFDM data packet or the OFDM frame.

Such an OFDM symbol train can be seen as a series of sample-shifted OFDM symbols, expressed as

$$O_{\text{TRAIN}}(n) = o_0(n) + o_1(n - N_{\text{OFDM}}) + o_2(n - 2N_{\text{OFDM}}) + \dots + o_{T-1}(n - (T-1)N_{\text{OFDM}})$$

which becomes

$$O_{\text{TRAIN}}(n) = \sum_{k=0}^{T-1} o_k(n - kN_{\text{OFDM}}). \tag{4.11}$$

Figure 4.3 illustrates the multiple OFDM symbol encoding process.

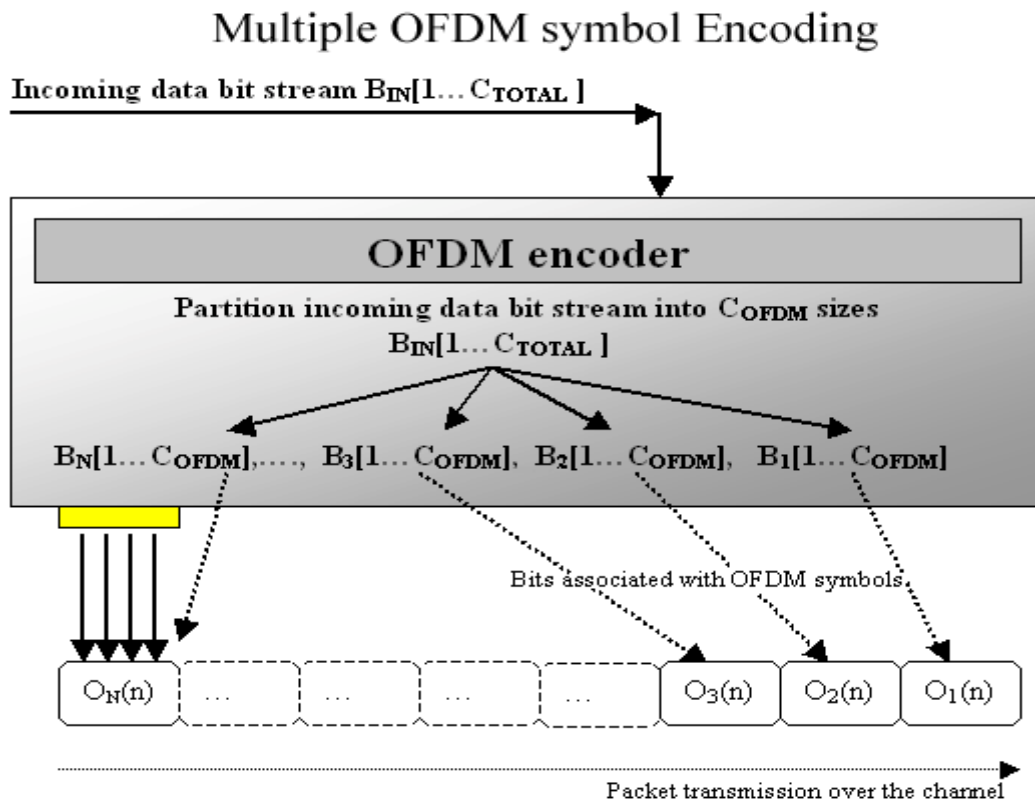


Figure 4.3: A graphical example of the multiple OFDM symbol encoding process.

OFDM symbol trains can theoretically encode unlimited amount of data bits into unlimited OFDM symbols, and transmit it across the communications channel.

4.4 Decoding OFDM Symbols

Decoding OFDM symbols refers to the process of de-mapping and demodulating the digital data bits from OFDM symbols.

4.4.1 Decoding Single OFDM Symbols

As with the encoding of single OFDM symbols, the decoding of single OFDM symbols is also done in the frequency domain, simply because it is easier and more convenient to do so. The complex time-domain signal is converted to the frequency spectrum using a DFT. Afterwards the phases and amplitudes of the sub-carriers at the different frequency locations are demodulated and de-mapped to digital data bits. The data bits are the result of the decoding process. Figure 4.4 illustrates the process of decoding a single OFDM symbol.

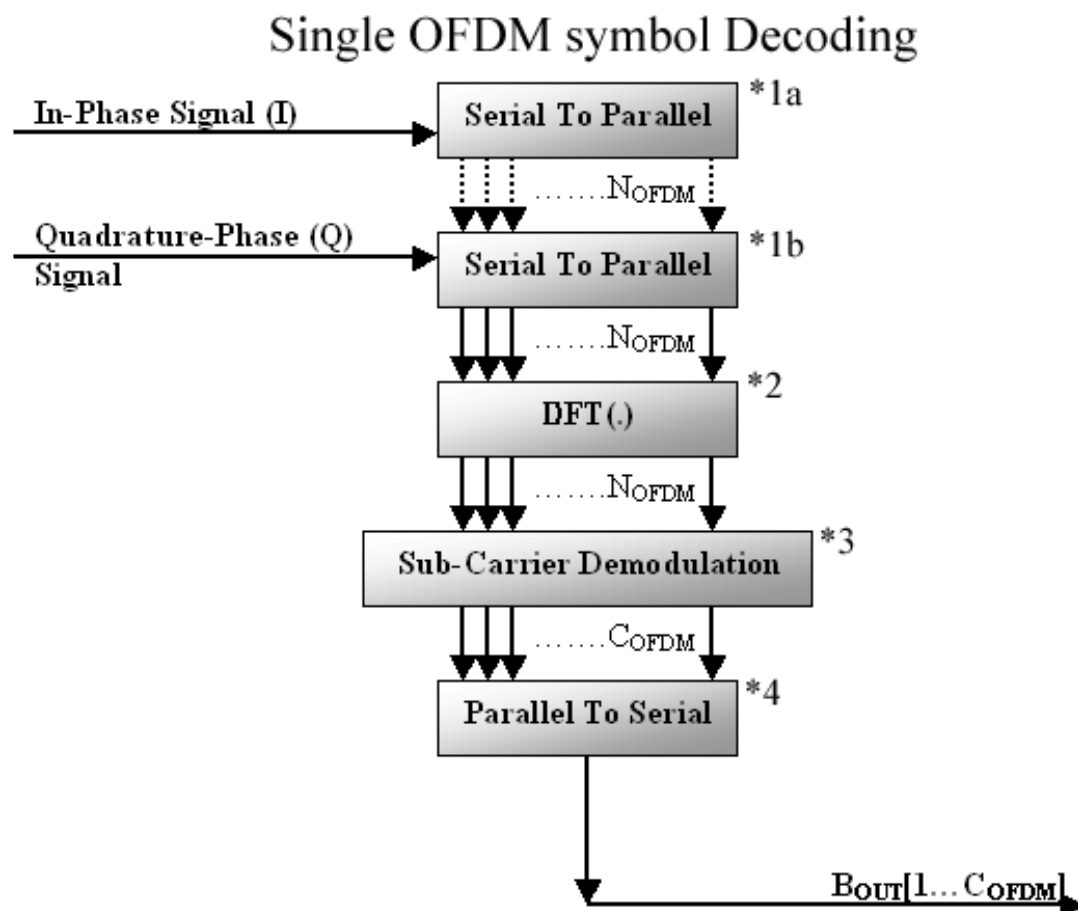


Figure 4.4: A graphical example of a single OFDM symbol decoding process.

Table 4.2 explains the OFDM symbol decoding process.

Nr	OFDM symbol encoding process
*1a, *1b	Firstly the In-Phase and Quadrature-Phase signals are converted to a parallel stream so that they can be processed together.
*2	Secondly the received complex time domain signal is converted to the frequency spectrum using a DFT
*3	Thirdly the Amplitude and Phases of the sub-carriers at different frequencies are mapped and demodulated to digital data bits (The Demodulation Process)
*4	Finally the digital data bits are converted to a serial output bit stream.
Out-put	The serial output bit stream is the result of the OFDM symbol decoding process, and can be passed to an upper layer application.

Table 4.2: The OFDM symbol decoding process

4.4.2 Decoding Multiple OFDM Symbols

Receiving and decoding OFDM symbol trains is the most complicated part of an OFDM communications system. OFDM symbols exist as separate entities. Each OFDM symbol contains information about the bits it encoded only, and has no information about any adjacent OFDM symbols. An OFDM decoder thus needs to be synchronised to each received OFDM symbol. An unsynchronised OFDM decoder runs the risk of decoding only part of an OFDM symbol together with part of an adjacent OFDM symbol. This is called inter symbol interference and results in severe degradation in data quality. In order to create an efficient OFDM symbol train decoder, the issue of synchronisation together with other performance influencing issues needs to be addressed first. It is a major part of this thesis, of which a whole chapter is set aside to discuss and study it. Please refer to Chapter 5 for more details.

Figure 4.5 illustrates the multiple OFDM symbol decoding process.

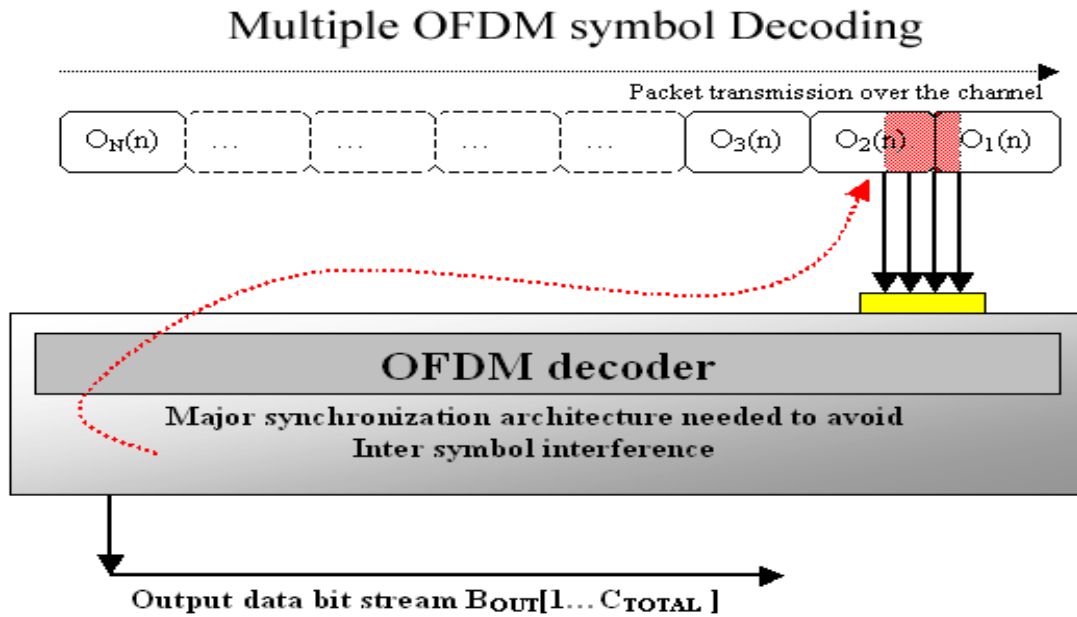


Figure 4.5: An OFDM decoder incorrectly decodes an OFDM symbol due to lack of OFDM symbol synchronisation.

An OFDM symbol train decoder needs to be synchronised with the received OFDM symbol, in order to successfully decode OFDM symbols.

4.5 Conclusions

In this chapter we first discussed the differences between the continuous-time domain in which real-life analogue signals exist and the discrete-time domain in which digital systems process data. The conversion between the two is done using a digital-to-analogue converter and an analogue-to-digital converter. Single OFDM symbol encoding and decoding was also studied, as well as encoding OFDM symbol trains. OFDM symbols are very sensitive to outside disturbances like noise, synchronisation and sampling frequency drift. These issues are discussed in the following chapter and need to be overcome before we can design a successful OFDM symbol train decoder.

Chapter 5

Identifying and Overcoming OFDM System Performance Influences

5.1 Introduction

The previous chapter showed us how to implement an OFDM encoder and how great amounts of data can be encoded across various OFDM symbols and transmitted over the communications channel. The process of decoding single OFDM symbols was also studied up to the point of decoding OFDM symbol trains. Decoding OFDM symbol trains is, to say the least, a very complicated matter. OFDM symbol trains are very finicky to disturbances and outside influences. This makes OFDM decoders very complicated and processor intensive, relative to normal single-carrier digital demodulators. Thus before describing the workings of a complete OFDM decoder, it is necessary to identify the factors which influence the performance of OFDM decoders. Only after identifying these real-life factors can we create means of overcoming them, and design an efficient OFDM decoder.

5.2 System Performance Influences

Identifying the main OFDM system performance influences caused by real-life factors and implementations is the first step in designing an efficient OFDM decoder. Here we discuss the six most important issues.

5.2.1 Noise

As already mentioned, noise is unwanted random interference that degrades the quality of signals. In Chapter 2 we analysed the influence of additive white Gaussian noise on QPSK modulated signals and derived expressions for determining symbol error rates (SER) and bit error rates (BER) of such systems.

We also know, from Chapter 3, that OFDM modulation, in its most basic form, is just a number of summed digitally modulated signals, all windowed using the same rectangular window. We can then ask the question: Can the expressions derived for single-carrier digitally modulated signals be adapted to work for OFDM modulation? The answer is YES, because OFDM sub-carriers are orthogonal to each other. Orthogonality ensures that OFDM sub-carriers remain independent of each other in all ways. This means that the effects of noise on OFDM modulated signals will be a direct result from the influence the noise has on the individual OFDM sub-carriers. The influence of noise on OFDM modulated signals can be determined as follows.

The digital bit capacity of a single QPSK modulated symbol is C_{QPSK} .

The BER of a single QPSK modulated symbol is BER_{QPSK} .

The theoretical amount of error bits per QPSK symbol is $E_{\text{QPSK}} = (C_{\text{QPSK}})(\text{BER}_{\text{QPSK}})$.

The amount of QPSK sub-carriers within a single OFDM modulated symbol is N_{SD} .

The digital bit capacity of a single OFDM modulated symbol is

$$C_{\text{OFDM}} = (C_{\text{QPSK}})(N_{\text{SD}}). \quad (5.1)$$

The theoretical amount of error bits per N_{SD} - amount of QPSK symbols, is also the theoretical amount of error bits per single OFDM symbol

$$E_{\text{OFDM}} = (C_{\text{QPSK}_1})(\text{BER}_{\text{QPSK}_1}) + (C_{\text{QPSK}_2})(\text{BER}_{\text{QPSK}_2}) + \dots + (C_{\text{QPSK}_{N_{\text{SD}}}})(\text{BER}_{\text{QPSK}_{N_{\text{SD}}}})$$

which then becomes

$$E_{\text{OFDM}} = (C_{\text{QPSK}})(\text{BER}_{\text{QPSK}})(N_{\text{SD}}). \quad (5.2)$$

The BER of a single OFDM symbol can finally be calculated as the theoretical amount of error bits per OFDM symbol, divided by the total amount of digital data bits in a single OFDM symbol

$$\text{BER}_{\text{OFDM}} = \frac{E_{\text{OFDM}}}{C_{\text{OFDM}}}, \quad (5.3)$$

which is

$$\text{BER}_{\text{OFDM}} = \frac{(C_{\text{QPSK}})(\text{BER}_{\text{QPSK}})(N_{\text{SD}})}{(C_{\text{QPSK}})(N_{\text{SD}})}$$

and finally becomes

$$\text{BER}_{\text{OFDM}} = \text{BER}_{\text{QPSK}}. \quad (5.4)$$

This means that the bit error rate of a single OFDM modulated symbol is equal to the bit error rate of any of the OFDM sub-carriers within the OFDM symbol, if all the sub-carriers use the same digital modulation. Since we have already deduced formulas to determine QPSK BER for different SNR values, those same graphs can be used as reference when testing the performance of our OFDM decoder later.

Overcoming noise is not an easy task. All communication systems should strive to have an SNR that is as high as possible. This reduces the possibility of bit errors but would never guarantee completely error free transfers. In many radio communication devices, the SNR depends on many things, including the distance between the receiver and transmitter and the temperature of the radio circuitry components [11]. It is thus not always possible to maintain an exceptionally high SNR. Instead of trying to correct the source of the problem, it is sometimes better to see how one can correct the result of the problem, in this case a poor BER. In order to boost data link quality, systems use a technique called Forward Error Correction (FEC).

In short, FEC techniques can automatically correct faulty digital bits by adding extra FEC bits into the digital data stream. This increases transmission quality at the cost of reduced transmission speed. FEC is a large and complicated area of research, which will not be looked at for the purpose of this thesis. For more information about FEC techniques, please refer to [1,13,14].

5.2.2 OFDM Symbol Synchronisation Issues

The most important step in successfully decoding an OFDM symbol train is the synchronisation of the OFDM decoder to the individual OFDM symbols within the OFDM symbol train. This is called OFDM symbol synchronisation. Successful synchronisation of an OFDM symbol train is achieved in two steps, the initial OFDM symbol train synchronisation [15] and the continuous re-synchronising of the OFDM data symbols [16,17].

5.2.2.1 Initial OFDM Symbol Synchronisation

In order for an OFDM decoder to successfully decode an incoming OFDM symbol train, the OFDM decoder needs to be exactly synchronised with the OFDM symbols. An unsynchronised OFDM decoder runs the high risk of decoding information across two adjacent OFDM symbols, which would result in inter symbol interference and degradation in data quality. Final tests done on the OFDM decoder has confirmed that synchronisation is the one factor that effects the results of the decoder much more than any other factor, including noise.

To overcome the issue of initial symbol synchronisation we introduce a new special OFDM symbol called the OFDM preamble symbol. The OFDM preamble symbol does not carry any encoded data and is purely used for synchronisation purposes. The OFDM transmitter transmits a short OFDM preamble train before the actual OFDM data train. The OFDM decoder knows the nature and exact structure of these OFDM preamble symbols and monitors the communication channel for the occurrence of these symbols. The OFDM decoder cross-correlates the received data with a local version of the OFDM preamble symbol. Cross-correlation can be described as a

measure of the similarity between two different data sets, computed by the sum of the cross products between the two datasets at different lags. When the OFDM preamble symbol train finally reaches the receiver, the cross-correlation will produce peak values, showing that there is a high similarity between the received signal and the local OFDM preamble symbol as shown in Figure 5.1.

Multiple OFDM symbol decoding with synchronization

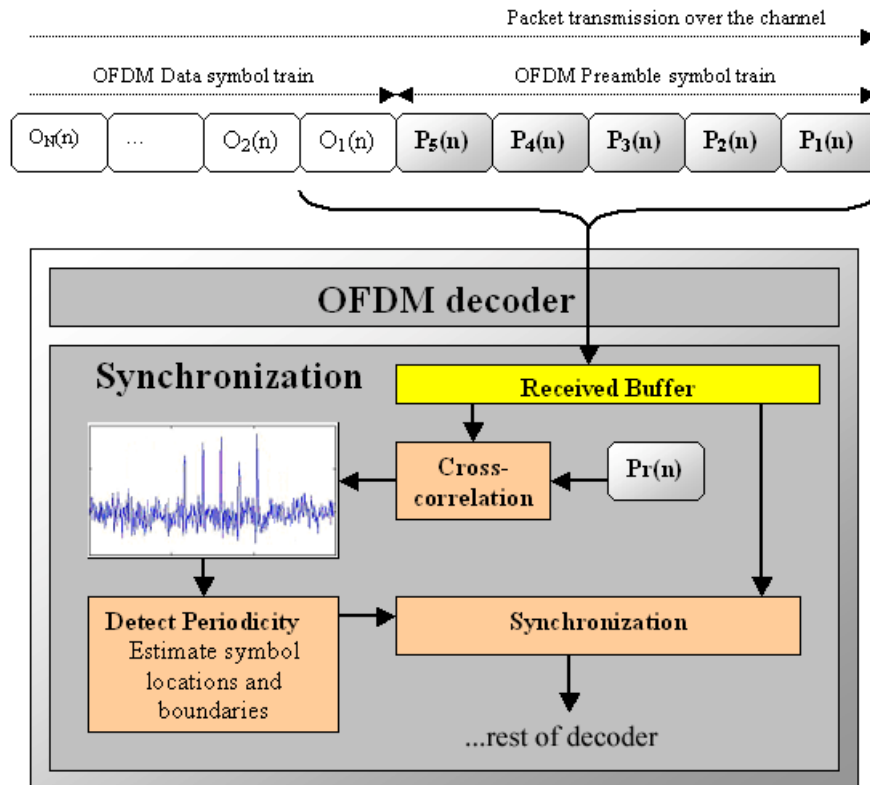


Figure 5.1: A graphical example of an OFDM decoder with initial OFDM symbol synchronisation.

The result of the OFDM preamble train cross-correlated with the local OFDM preamble symbol, is a signal with periodic spikes showing the location of each of the OFDM preamble symbols in the transmission. By examining these peaks in the cross-correlation result, it is possible to exactly determine the start location and boundary of each OFDM preamble symbol, as well as the start location of the following OFDM data symbol train. The OFDM decoder will thus be able to synchronise to the OFDM symbols. A decoder's local OFDM preamble symbol is given by $P_{r_LOCAL}(n)$.

The OFDM preamble symbol is N_p samples in length. The amount of OFDM preamble symbols in an OFDM preamble train is P_N . An OFDM preamble symbol train can then be expressed as

$$P_{\text{Train}}(n) = \sum_{k=0}^{P_N-1} P_r(n - kN_p). \quad (5.5)$$

The cross-correlation between the OFDM preamble symbol and a received signal is

$$r_{Pr,Rx}(k) = \sum_{n=-\infty}^{\infty} R_x(n) P_{r_LOCAL}(n - k). \quad (5.6)$$

If the OFDM decoder then receives an OFDM preamble symbol train

$$R_x(n) = P_{\text{Train}}(n) = \sum_{s=0}^{P_N-1} P_r(n - sN_p), \quad (5.7)$$

the cross-correlation between a local receiver's OFDM preamble symbol and a received OFDM preamble train thus becomes

$$r_{Pr,Rx}(k) = \sum_{n=-\infty}^{\infty} P_{r_LOCAL}(n - k) \sum_{s=0}^{P_N-1} P_r(n - sN_p), \quad (5.8)$$

which can be rewritten as

$$r_{Pr,Rx}(k) = \sum_{n=-\infty}^{\infty} \sum_{s=0}^{P_N-1} P_{r_LOCAL}(n - k) P_r(n - sN_p). \quad (5.9)$$

Since cross-correlation measures the similarity between two datasets, the strongest correlation will occur whenever the decoder's OFDM preamble symbol is aligned with each of the received OFDM preamble symbols within the OFDM preamble train.

From Equation (5.9) we can see that that this happens when the position of the decoders' local OFDM preamble is equal to the location of the received OFDM preambles so that

$$k = sN_p; \quad 0 \leq s \leq P_n - 1. \quad (5.10)$$

This means that the peaks will occur at position of a received OFDM preamble symbol as given by (5.10). For a perfect channel, $P_{r_LOCAL}(n) = P_r(n)$. The following figure shows the cross-correlation peaks located at the rear of each received OFDM preamble symbol.

Initial OFDM Symbol Synchronization

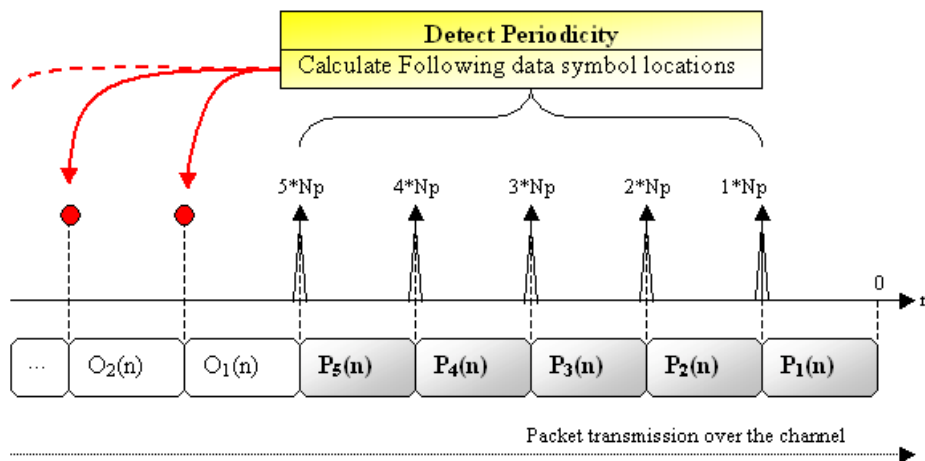


Figure 5.2: A graphical example of initial OFDM symbol synchronisation results using cross-correlation.

By detecting the periodicity of the peaks produced by the cross-correlation of a receivers' local OFDM preamble symbol and the received OFDM preamble train, it is possible to determine the position of each OFDM preamble symbol and thus calculate the start location of the following OFDM data symbols, and synchronise on it.

5.2.2.2 Continuous re-synchronisation of OFDM data symbols

Communications systems use oscillators to generate clock signals for timings that control the systems' sampling frequency. Impurities in the physical makeup of the oscillators results in sampling frequency drift. These impurities are measured in "parts per million" or PPM. As example, a system running at 20kHz, with a oscillator which has a 50 PPM rating, could in actual fact be anything between

$$20\,000\text{Hz} * (1 - 50\text{PPM}) = 19\,999\text{Hz}$$

and

$$20\,000\text{Hz} * (1 + 50\text{PPM}) = 20\,001\text{Hz}.$$

This means that a system which expects 20 000 samples per second, could in actual fact be out with at least 1 sample after 1 second. This is called sampling frequency drift. The OFDM symbols are usually only a few samples in length and the effects of the sampling frequency drift aren't easily observable. This is because the effects are sub-sample in size, which means that an OFDM symbol will become de-synchronised in parts that is smaller than one actual sample. Although this is a very small value, to a system such as an OFDM decoder, which depends on absolute synchronisation, it could severely influence the quality of data. In the long run the effects would become worse and worse and finally result in a completely de-synchronised OFDM decoder. The difference in sampling frequency can be expressed as

$$F_s(\text{decoder}) = F_s(\text{transmitter}) + \Delta F_s.$$

Except for the long-term de-synchronisation effect of the sampling frequency drift, the sub-sample shifts in OFDM symbols also causes degradation in data quality due to the following Fourier transform pair

$$o(n - n_o) \xrightarrow{\text{FFT}} O(n) e^{-j2\pi \frac{k}{N} n_o} \quad (5.11)$$

The Fourier transform above states that any signal which is delayed by a value n_0 , be it sub-sample or super-sample in size, directly influences the phases of all the frequency components inside that signal. The phase influence becomes worse as the frequencies increase. For systems such as the OFDM encoders that encode the digital data into sub-carrier phases, this is a major problem. See Figure 5.3.

The effects of the sampling frequency drift are overcome by using reference carriers, also called virtual carriers. Reference carriers are placed at different frequency positions throughout the OFDM symbol, and they all carry constant phases throughout the whole OFDM symbol train transmission. The OFDM decoder knows the positions and nature of these reference carriers.

In order for an OFDM decoder to compensate for the effects of possible sampling frequency drift, the OFDM decoder, upon receiving an OFDM symbol for decoding, first decodes the reference carriers.

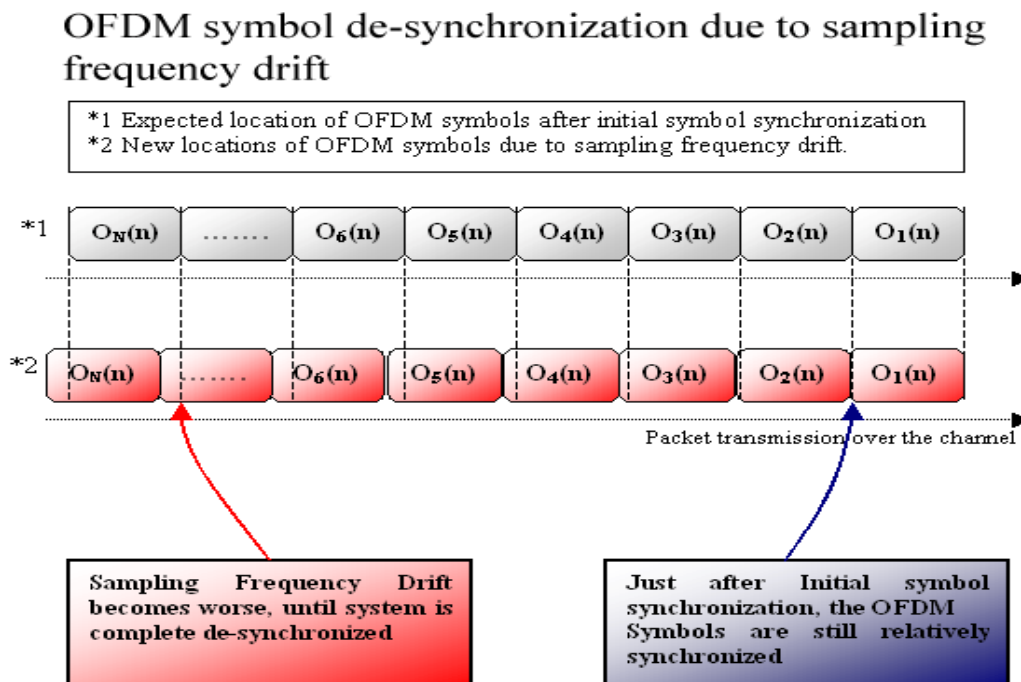


Figure 5.3: OFDM symbol de-synchronisation due to sampling frequency drift.

The OFDM decoder knows the expected reference carrier phases and frequency positions. The receiver can thus compare the received reference phases with the expected reference phases. If there is a difference, the receiver will immediately know that some sort of sample delay has occurred.

The expected reference carrier with digital frequency k and phase ϕ_k can be expressed as

$$O(n, k) = |O(n, k)| \angle \phi_k. \quad (5.12)$$

The influence of a sample delay from (5.11) is

$$e^{-j2\pi\frac{k}{N}n_o} = |1| \angle (2\pi\frac{k}{N}n_o). \quad (5.13)$$

The received reference carrier with a sample delay becomes

$$O(n, k)e^{-j2\pi\frac{k}{N}n_o} = (|O(n, k)| \angle \phi_k) \left(|1| \angle (2\pi\frac{k}{N}n_o) \right) \quad (5.14)$$

which gives

$$O(n, k)e^{-j2\pi\frac{k}{N}n_o} = |O(n, k)| \angle \left(\phi_k + 2\pi\frac{k}{N}n_o \right). \quad (5.15)$$

The phase difference between the expected and received reference carrier is thus

$$\phi_{\square} = 2\pi\frac{k}{N}n_o. \quad (5.16)$$

Since the index value, k , the symbol size, N , and the reference carrier phase difference is known, after some precise phase unwrapping, the sample offset can be calculated as

$$n_o = \left(\frac{N\phi_{\square}}{2\pi k} \right). \quad (5.17)$$

After the sample offset has been calculated, (5.13) can be used to calculate the phase offset at all the data carriers. The phase offsets can then be subtracted from all the data carriers, to retrieve the original phase values. This is called reference carrier phase compensation.

It is very important to note that the reference carriers too, can be influenced by system noise. A noisy and incorrect reference can cause the decoder to incorrectly calculate the sample offset and incorrectly compensate all the data sub-carrier phases in the symbol. It is suggested that in practice more than one reference carrier is used and a mean or weighted decision from all the reference carriers be used when determining the sample offset.

If one can assume that the system oscillator characteristics do not change drastically between OFDM symbols, it is further possible to keep a record of sample offsets between OFDM symbols. The result would be a steady increase or decrease in the sample offset (which, if left unattended, would result in decoder de-synchronisation), which could be used to determine the rate of change in sample offset. Future expected sample offsets could be calculated and used to test against suspicious sample offset outliers.

When the sub-sample offset has reached the point where it is the size of an actual sample, the decoder can just skip or delay one sample to get the decoder precisely synchronised again.

5.2.3 Multipath Effects on OFDM symbols

Multipath delay is a phenomenon which occurs when signals are transmitted and received using electromagnetic waves [5,18,26]. The quickest way for a transmitted signal to travel between a transmitter and a receiver is in a straight line, but non-directional antennae transmit signals in different directions. Sometimes it happens that transmitted signals reflect off objects and still reach the receiver. Due to the longer distance these signals travel, they reach the receiver later than a direct line-of-sight signal, and with a smaller amplitude. These multipath signals (also known as delay spread) cause delayed versions (a time-domain smearing effect) of the original signal to occur at the receiver. Such an arbitrary transmitted signal can be expressed as

$$Tx(t) = As(t). \quad (5.18)$$

The multipath received signal can then be expressed as a number of summed delayed versions of the original signal, with different amplitudes

$$Rx(t) = Bs(t - T_1) + Cs(t - (T_1 + T_{E1})) + Ds(t - (T_1 + T_{E2})). \quad (5.19)$$

As we have already stated, OFDM symbols exist as separate entities of limited length. A delayed OFDM symbol will thus overflow the original OFDM symbol boundaries into the following OFDM symbol and cause inter-symbol interference.

The effects of multipath delay are overcome by cyclically extending the OFDM symbol, by copying the rear part of the OFDM symbol and pasting it to the front of the symbol. The front of the new extended OFDM symbol can thus absorb the delayed copies from the previous symbol and still keep the original OFDM symbol data unaltered. This is called the Cyclic Prefix or Guard Interval of the symbol.

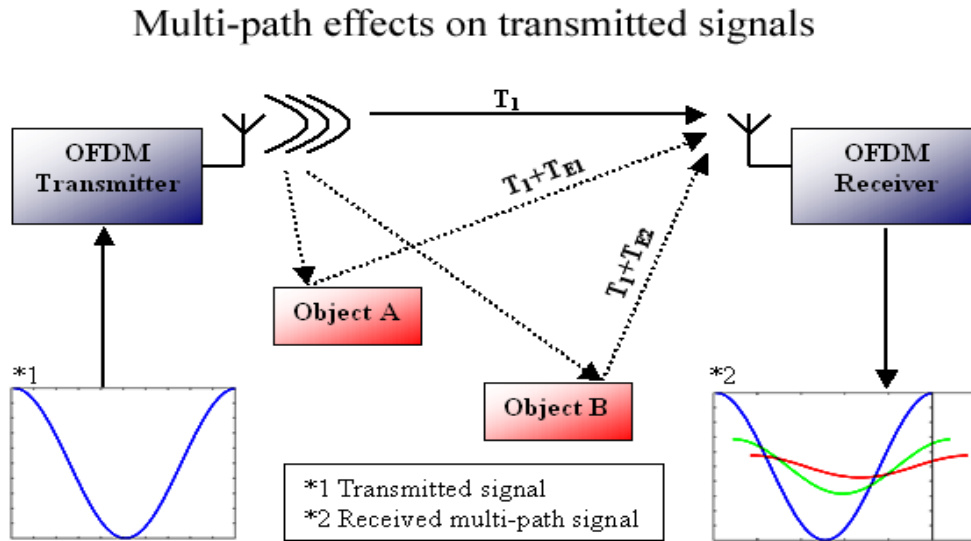


Figure 5.4: Multipath effects causes delayed signals to reach the receiver.

Choosing the length of the guard interval is an interesting subject. The guard interval should be long enough to absorb all the major delay spread. Making the guard interval too long, on the other hand, wastes precious transmission time. It finally comes down to the typical environment the system is going to be used in, and the types of multipath signals the system has to deal with. Multipath fading is a large and complicated area of study. OFDM sub-carriers have long symbol periods and small bandwidths, which are assumed to be flat fading channels. This removes the need for any complicated channel equalisations. In the case of OFDM, guard intervals are sufficient countermeasure for delay spread. The guard interval for an IEEE 802.11a based system is 800 ns, according to the specifications [1]. Figure 5.5 illustrates the effects of delay spread on OFDM symbols and how guard intervals are used to combat it.

The Effects of Delay Spread

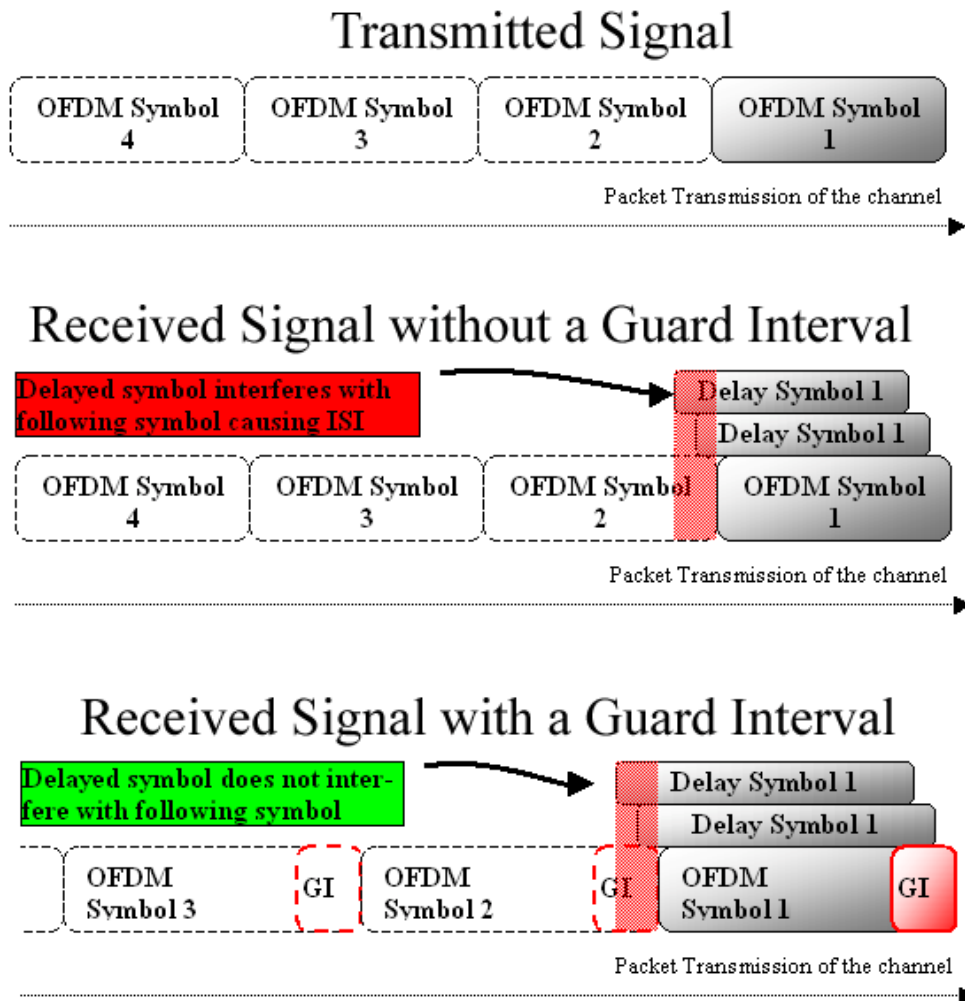


Figure 5.5: The effects of delay spread and use of guard intervals

5.2.4 Peak-to-Average Power Ratio

The Peak-to-Average Power Ratio (PAPR) is a way of measuring the ratio between the peak power of a signal and its average power [5,19,20]. For OFDM modulated signals, where the signal basically consists of a number (N_{ST}) of summed sub-carriers, it is possible for parts of the signal to add constructively and produce peaks that is N_{ST} times larger than the average signal level. For instance, if an IEEE 802.11a based system has a total of $N_{ST}=52$ sub-carriers, the PAPR of the OFDM signal can be calculated as

$$\text{PAPR}_{\text{dB}} = 10 \log_{10} (N_{\text{ST}}), \quad (5.20)$$

which gives

$$\text{PAPR}_{\text{dB}} = 10 \log_{10} (52) = 17.16 \text{dB}. \quad (5.21)$$

A large PAPR is a disadvantage to a system, and has a major influence on the dynamic range of the system as well as the transmitter amplifier.

5.2.4.1 OFDM Dynamic Range Issues

As explained in Chapter 4, digital systems store data in a discrete fashion. Digital data bits are used to represent digital values and the more bits are used, the bigger the value it can store. Continuous-time signals have amplitudes that get mapped to digital amplitude values. The ADCs quantise the analogue values to the nearest digital data value it can represent. The more bits used to store these digital amplitude values, the finer the resolution of the amplitudes, and thus a more detailed signal. Popular quantising levels are 8, 12 and 16 bits, but can go as high as 24 and 32 bits. Signals with high PAPR values, like OFDM signals, have high peaks that are located relatively far from the majority of signal activity. Since the whole signal needs to be mapped to digital values, many of the available bits are “wasted” because they’re only there to fill the range between the peak and the major signal activity. It is thus possible for signals with very high PAPR values to start losing signal resolution, which in terms leads to quantising noise, and a degradation of the data quality.

5.2.4.2 Transmitter Amplifier Issues

A high PAPR becomes an issue when one attempts to transmit the OFDM signals over the air using electromagnetic waves. The analogue front-end utilises a power amplifier that amplifies the OFDM signal to required levels before transmitting it using an antenna. A power amplifier is not inherently linear, but it is made linear by utilising linear areas of operation within the amplifier. The power amplifier must thus

provide gain for every peak level within the signal without warping the signal because of these non-linear regions. Warping of the OFDM signals will negatively effect the quality of the signal and the resulting bit error rate. A high PAPR will result in the power amplifier providing less power to the signals between the peaks. The DC power consumption of power amplifiers is determined by their peak power, which means that OFDM power amplifiers could also be dramatically inefficient. Minimising the PAPR allows a higher average power to be transmitted for a fixed peak power, improving the overall signal-to-noise ratio at the receiver.

Overcoming the effects of a high PAPR is not a very straightforward task. Some methods include the use of peak reduction carriers [21] where additional OFDM sub-carriers are added to the OFDM symbol, specially encoded to reduce peaks in the OFDM symbol. Another method of reducing PAPR is by clipping [22] the OFDM symbol at certain levels and then counteracting the clipping effects by the use of special filters.

5.2.5 Communications Channel Estimation

Channel estimation is the process where the influence of the communication channel on the transmission signal is estimated, in order to predict the received signal, or to counteract its effects [23]. A communication channel, be it a copper wire or radio waves, will sometimes differently attenuate different frequencies in the transmission signal. Such attenuation can be a result of the physical characteristics of the communications channel as well as many other things. The channel transfer function is a transform, which can be used to describe the output of a system as a function of the input. Digital demodulators such as ASK and QAM demodulators which decode digital data as a function of the magnitude of the received signal at certain frequencies, run the risk of incorrectly decoding the data if it happens that the channel is attenuated at the decoding frequencies. The channel transfer function can be expressed in terms of the Laplace transform, which basically operates in the frequency domain.

The transmitted signal $X(s)$ is transformed into the received signal $Y(s)$ by means of the transfer function $H(s)$, by

$$Y(s) = H(s)X(s). \quad (5.22)$$

The transfer function can be expressed as

$$H(s) = \frac{Y}{X}(s). \quad (5.23)$$

One way to counteract the effects of the channel transfer function is for the receiver to transform the received signal by the inverse channel transfer function $H^{-1}(s)$. The inverse channel transfer function can be calculated as

$$H^{-1}(s) = \frac{X}{Y}(s). \quad (5.24)$$

The received signal can then be transformed using the inverse channel transfer function, to produce a new output

$$Y_2(s) = Y(s)H^{-1}(s), \quad (5.25)$$

which then becomes the original input signal

$$Y_2(s) = H(s)X(s)H^{-1}(s) = \frac{Y}{X}(s)X(s)\frac{X}{Y}(s) = X(s). \quad (5.26)$$

In order for the receiver to counteract the effect of the channel transfer function the receiver needs to know what the original input signal was. A perfect candidate for channel estimation is the OFDM preamble symbol.

During the initial symbol synchronisation process, the Laplace Transform or Fourier transform can convert the local preamble symbol $P_{r_LOCAL}(n)$ and the received preamble symbol, $P_r(n)$ (after successful synchronisation) to their respective spectra. The inverse channel transfer function will then be

$$H^{-1}(s) = \frac{P_{r_LOCAL}(s)}{P_r(s)}. \quad (5.27)$$

Any signal received thereafter can then be transformed using the inverse channel transfer function, but it should be noted that it is possible for the characteristics of the channel to change in the long term, and thus the channel transfer function will change as well. It is thus suggested that the transfer function be updated on a regular basis during long periods of communications.

The time-domain representation of the channel transfer function is called the channel impulse response $h(n)$, and can be defined as the response of a system to an impulse as input signal. The impulse response of a communications channel is basically responsible for limiting instantaneous change in signals. This has an effect on OFDM encoded signals, since every following OFDM data symbol is a new signal. The result is a distortion of the front part of each new OFDM symbol and the digital data it carries.

One method of overcoming this distortion is by cyclically extending the OFDM data symbol in the front. The cyclic extension keeps the OFDM symbol periodic, and if the cyclic prefix is longer than the impulse response it will keep the original OFDM symbol unaltered. The cyclic prefix has already been introduced in combating multipath effects, and is now further crucial in combating the effects of communication channel impulse response.

5.2.6 System Performance Influences Conclusions

The six most serious system performance influences have been determined as

- Additive white Gaussian noise.
- Initial OFDM symbol synchronisation at the receiver.
- Continuous OFDM symbol re-synchronisation due to system oscillator impurities.
- Multi-path fading effects on OFDM symbols.
- PAPR issues with high sub-carrier count OFDM systems.
- Communication channel estimation and compensation.

After extensively examining each of these performance influencing factors, and determining means to overcome them, it is now possible to design a effective OFDM receiver system.

5.3 Advanced System Performance Improvements

There exist a few additional, and more complex techniques, which can improve OFDM system performances. These advanced improvements are more specialised in nature and their performance improvements not always guaranteed.

5.3.1 Data Interleaving

Data interleaving is the process where by the digital data bits are encoded to random OFDM sub-carriers[1,3,6,12]. Continuous narrow-band interference of the OFDM signal by external devices, cause bursts of errors in the decoded data stream. Unfortunately forward error correction schemes are not so effective in correcting bursts of errors. By placing the encoded data bits at random OFDM sub-carriers, bursts of erroneous data bits become more random in nature, and thus improve the correcting ability of the forward error correction scheme.

5.3.2 Sub-carrier Adaptive Bit Loading

Adaptive bit loading is a technique where each OFDM sub-carrier modulation scheme is determined by the quality of that OFDM sub-carrier [12,24]. OFDM sub-carriers with high SNR can be given more power and higher-order modulation techniques. Lower-order modulation techniques are assigned to OFDM sub-carriers with poor SNR. Sub-carriers with very bad SNR can even be turned off. Adaptive bit and power loading can improve system efficiency and optimise data performance of the system.

5.3.3 Spatial Multiplexing Techniques

Spatial multiplexing [25], also known as MIMO or Multiple-In Multiple-Out techniques, attempts to increase transmission signal strength by increasing the amount of antennae at the transmitter and receiver. Multipath fading at two different antennae could be uncorrelated over the two different transmission paths. Combining these received signals will result in a SNR higher than the SNR in any of the individual signals. Higher SNR will result in better data quality and lower BER.

5.4 Conclusions

In this chapter we have introduced and studied the six primary OFDM system performance influences. Methods for overcoming them have also been discussed in detail. Any OFDM receiver will need to implement them in order to operate at reasonable performance levels. There exist some further advanced methods for improving system performance. These methods do not guarantee improvements and can come at the price of increased processing power or extra hardware. In the next chapter we study the IEEE 802.11a standard of OFDM encoding. The IEEE 802.11a standard is very robust, and incorporates many of the methods described in this chapter for overcoming the OFDM system performance influences. The SDR implementation of the OFDM transceiver system is also based on the IEEE 802.11a standards.

Chapter 6

IEEE 802.11a OFDM standard

6.1 Introduction

The Institute of Electrical and Electronic Engineers, also known as the IEEE, is a non-profit, technical professional association with hundreds of thousands of members from approximately 175 countries. Through its members, the IEEE is a world leading authority in many technical areas, including telecommunications. The IEEE has over 37 societies that specialise in different technologies and creates standardised rules based on these existing and new technologies. The standardisation of technologies give different companies the chance to produce compatible products that results in competitive markets for their products. The IEEE 802.11 standard is an evolving family of specifications for the wireless transmission of local area networks. The IEEE 802.11a standard is an enhancement of the IEEE 802.11 standard and is tailored for use in high-speed access hubs operating in the frequency range 5.725 GHz to 5.850 GHz [1].

6.2 The IEEE 802.11a Standard

The IEEE 802.11a standard incorporates OFDM modulation to encode digital data bits onto 64 sub-carriers spanning across 20 MHz of bandwidth to provide transmission of data at rates of 6, 9, 12, 18, 24, 36, 48, or 54 Mbps. The standard incorporates methods for overcoming virtually all the real-life performance influences we have discussed in previous chapters, and the result is a robust system based on OFDM modulation for the wireless transmission of computer network data.

6.2.1 The IEEE 802.11a Standard Overview

Some of the features that the IEEE 802.11a standard incorporates:

- Short and Long preamble symbols for initial symbol synchronisation.

- Cyclic extended data symbols to combat multipath fading effects.
- Reference carriers within each data symbol for continuous re-synchronisation.
- Convolutional encoding forward error correction to combat noise effects.
- Bit interleaving for added error correction efficiency.

Four different IEEE 802.11a symbol types exist:

- Short Preamble Symbols
- Long Preamble Symbols
- OFDM Signal Symbol
- OFDM Data Symbol

The IEEE 802.11a data symbol specifications are:

- 48 sub-carriers used for encoding data.
- Sub-carrier encoding options: BPSK, QPSK, 16-QAM and 64-QAM.
- 4 sub-carriers used as reference carriers.

The IEEE 802.11a packet is made up of 10 short preambles followed by 2 cyclically extended long preambles, a signal symbol and then a maximum of 4096 data symbols, as shown in Figure 6.1.

An IEEE 802.11a packet format

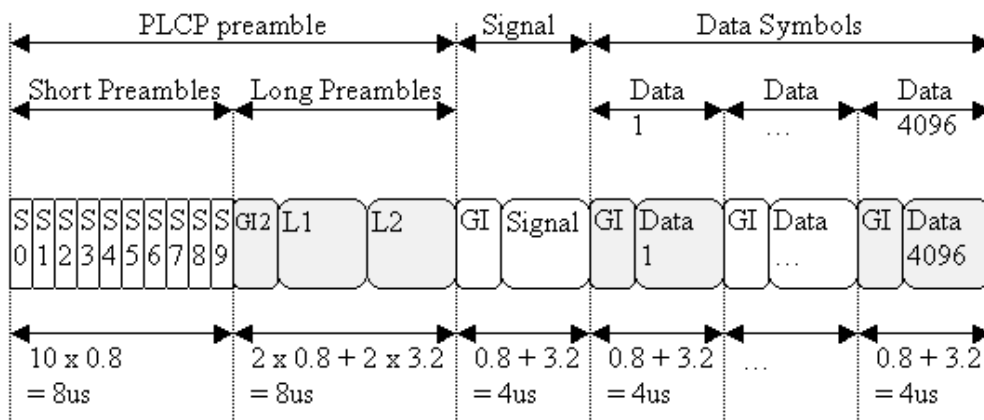


Figure 6.1: An IEEE 802.11a standard packet format

The IEEE 802.11a OFDM symbols use a total of 20MHz bandwidth, which is divided into 64 subcarriers spaced 312.5 kHz apart. Only the first 52 sub-carriers are used, of which 48 are used for data and the 4 as reference carriers. Refer to Table 6.1 for detail timing parameters of the IEEE 802.11a system [1].

Parameter	Value
B_W : OFDM symbol bandwidth	20 MHz
N_{OFDM} : Total amount of OFDM sub-carriers	64
Δ_F : Sub-carrier frequency spacing	0.3125 MHz (20MHz/64)
N_{ST} : Number of Used sub-carriers	52 ($N_{SD} + N_{SP}$)
N_{SD} : Number of Data Sub-carriers	48
N_{SP} : Number of Pilot (Reference) Sub-carriers	4
T_{FFT} : IFFT/FFT Period	3.2 us ($1 / \Delta_F$)
T_{GI} : GI (Cyclic Prefix) Duration	0.8 us ($T_{FFT} / 4$)
T_{SYM} : Complete Symbol Timing (FFT+CP)	4.0 us ($T_{FFT} + T_{GI}$)
T_{GI2} : Cyclic prefix for Long Preambles	1.6 us ($T_{FFT} / 2$)
T_{LONG} : Long Preamble Sequence	8.0 us ($2 \times T_{FFT} + T_{GI2}$)
T_{SHORT} : Short Preamble Sequence	8.0 us ($10 \times T_{FFT} / 4$)

Table 6.1: IEEE 802.11a timing parameters.

The IEEE 802.11a transmission time can thus be calculated as

$$T_{802.11} = T_{LONG} + T_{SHORT} + T_{SYM}(\text{signal}) + (\text{number of OFDM packets})(T_{SYM}) \quad (6.1)$$

which is

$$T_{802.11} = 20\text{us} + (\text{number of OFDM packets})(4\text{us}). \quad (6.2)$$

It is further possible to calculate the “over-the-air” data rate of an IEEE 802.11a system by choosing different sub-carrier modulation schemes as well as different convolutional coding rates. Refer to Table 6.2 for the data rate dependent parameters.

Data rate Megabits/s	Sub-carrier Modulation	Coding rate Convolution	Total Bits / sub-carrier	Total Bits / symbol	Data Bits /symbol
6	BPSK	1/2	1	48	24
9	BPSK	3/4	1	48	36
12	QPSK	1/2	2	96	48
18	QPSK	3/4	2	96	72
24	16-QAM	1/2	4	192	96
36	16-QAM	3/4	4	192	144
48	64-QAM	2/3	6	288	192
54	64-QAM	3/4	6	288	216

Table 6.2: Data rate dependent parameters of an IEEE 802.11a system

A 64-point IFFT or IDFT function converts the spectrum into complex time domain-signals. The complex frequency-domain coefficients used to describe the different OFDM time signals have indices that range from -26 to 26 . These indices map onto the IDFT block as shown in the Figure 6.2.

IEEE 802.11a IDFT function block

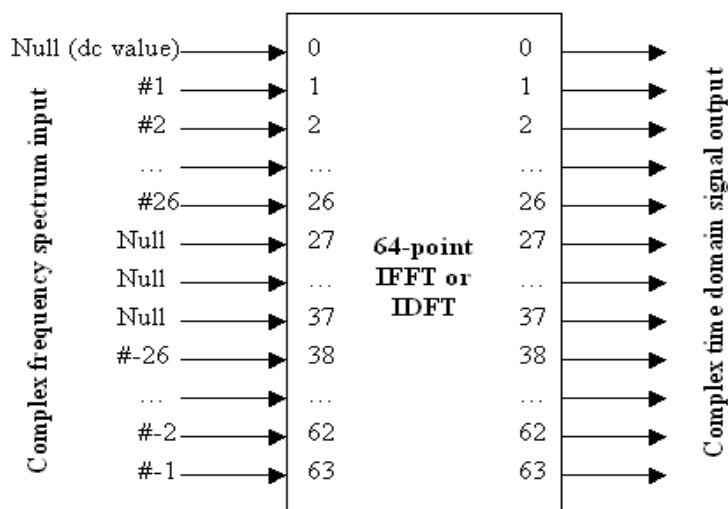


Figure 6.2: An IEEE 802.11a IDFT/IFFT function block

At a sampling rate of 20 MSPS an IEEE 802.11a IDFT function block will produce complex time domain signals of exactly 3.2 us duration. This is referred to as T_{FFT} .

6.2.2 The IEEE 802.11a Short Preamble Symbol

The IEEE 802.11a short preamble symbol train make up the first half of the PLCP preamble, and is exactly 8 microseconds in duration. Each short preamble symbol can be described using the series of complex DFT coefficients

$$s_{-26,26} = \sqrt{(13/6)} \times \{0, 0, 1 + j, 0, 0, 0, -1 - j, 0, 0, 0, 1 + j, 0, 0, 0, -1 - j, 0, 0, 0, \dots \\ \dots, -1 - j, 0, 0, 0, 1 + j, 0, 0, 0, 0, 0, 0, 0, -1 - j, 0, 0, 0, -1 - j, 0, 0, 0, 1 + j, \dots \\ \dots, 0, 0, 0, 1 + j, 0, 0, 0, 1 + j, 0, 0, 0, 1 + j, 0, 0, 0\}. \quad (6.3)$$

The complex time-domain signal is then generated using the Equation (6.4),

$$r_{\text{SHORT}}(t) = w_{\text{TSHORT}}(t) \sum_{k=-N_{\text{ST}}/2}^{N_{\text{ST}}/2} (S_k \cdot e^{j 2 \pi k \Delta_F t}) \quad (6.4)$$

where $w_{\text{TSHORT}}(t)$ is the symbol windowing function.

The short preamble symbol complex coefficients have non-zero values at every fourth index, which results in a signal $r_{\text{SHORT}}(t)$ that has a periodicity of $T_{\text{FFT}}/4 = 0.8\mu\text{s}$. It is thus possible to construct the short preamble train by adding two short preamble symbols together with one half concatenated symbol, and for the resulting short preamble train to still remain periodic over the entire duration.

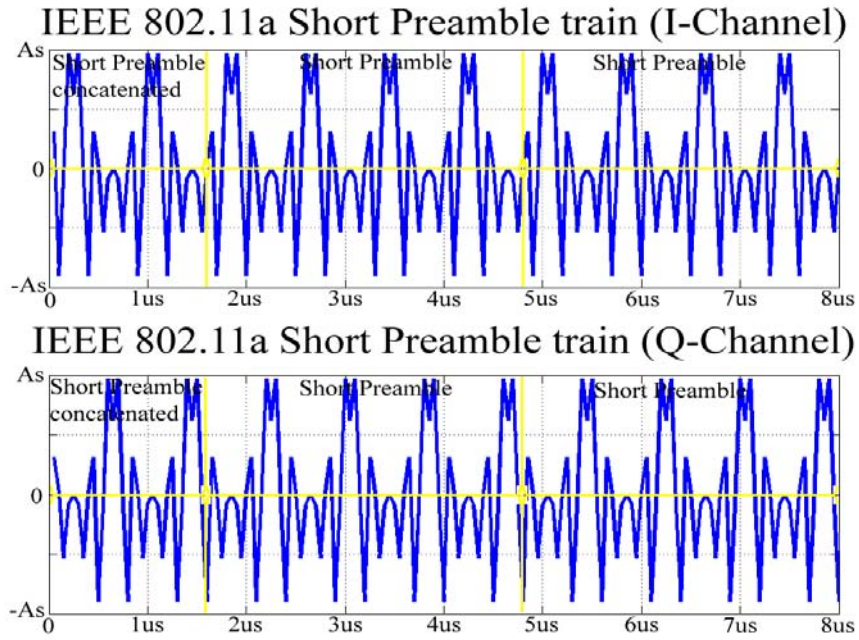


Figure 6.3: The IEEE 802.11a short preamble train.

The short preamble train has a total duration of

$$T_{\text{SHORT}} = (2)T_{\text{FFT}} + (0.5)T_{\text{FFT}}, \quad (6.5)$$

$$T_{\text{SHORT}} = (2)3.2\mu\text{s} + (0.5)3.2\mu\text{s} = 6.4\mu\text{s} + 1.6\mu\text{s} = 8.0\mu\text{s}. \quad (6.6)$$

The short preamble train can finally then be expressed as

$$r_{\text{SHORT_TRAIN}}(t) = r_{\text{SHORT}}(t) + r_{\text{SHORT}}(t - T_{\text{SHORT}}) + \tilde{r}_{\text{SHORT_CONCAT}}(t - 2T_{\text{SHORT}}). \quad (6.7)$$

6.2.3 The IEEE 802.11a Long Preamble Symbol

The IEEE 802.11a long preamble symbol train make up the second half of the PLCP preamble, and is exactly 8 microseconds in duration. Each long preamble symbol can be described using the series of complex coefficients

$$L_{-26,26} = \{1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 0, 1, -1, -1, \dots \\ \dots, 1, 1, -1, -1, -1, 1, -1, -1, -1, -1, -1, 1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1\}. \quad (6.8)$$

The complex time domain signal, is then generated using Equation (6.9),

$$r_{\text{LONG}}(t) = w_{\text{TLONG}}(t) \sum_{k=-N_{\text{ST}}/2}^{N_{\text{ST}}/2} (L_k \cdot e^{j 2 \pi k \Delta_F (t-T_{\text{GI2}})}) \quad (6.9)$$

where $w_{\text{TLONG}}(t)$ is the symbol windowing function.

Each long preamble symbol has duration of T_{FFT} . The final long preamble train is then constructed by adding two long preambles symbols together with a cyclic prefix of the long preamble also known as the long preamble guard interval of T_{GI2} . The long preamble guard interval is exactly half the length of the long preambles

$$T_{\text{LONG}} = 2T_{\text{FFT}} + T_{\text{GI2}}, \quad (6.10)$$

$$T_{\text{LONG}} = 6.4\mu\text{s} + 1.6\mu\text{s} = 8.0\mu\text{s}. \quad (6.11)$$

The long preamble train can finally then be expressed as

$$r_{\text{LONG_TRAIN}}(t) = r_{\text{LONG}}(t) + r_{\text{LONG}}(t - T_{\text{SHORT}}) + \tilde{r}_{\text{LONG_GUARD_INTERVAL}}(t - 2T_{\text{SHORT}}). \quad (6.12)$$

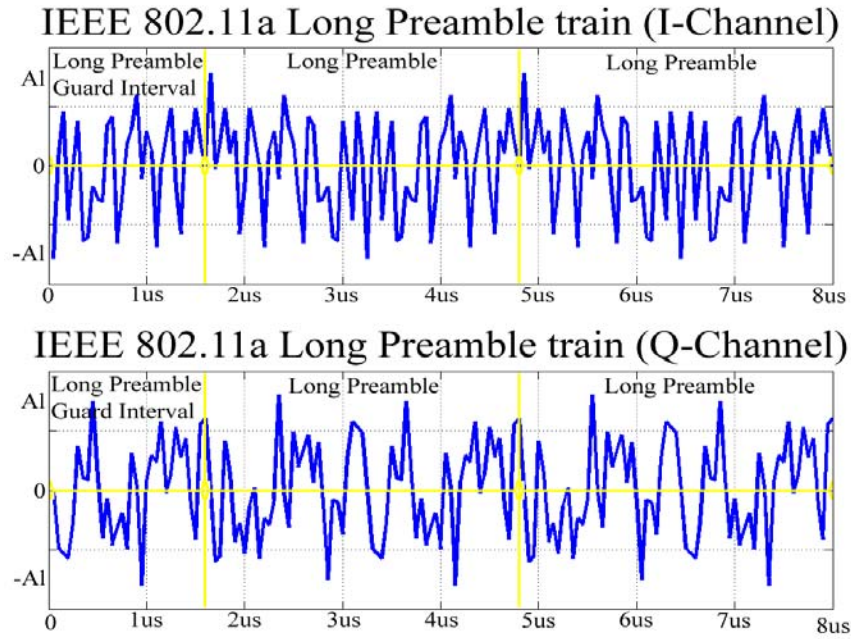


Figure 6.4: The IEEE 802.11a long preamble train.

6.2.4 The IEEE 802.11a Signal Symbol

The IEEE 802.11a signal symbol is a special symbol that is encoded with information about the current transmission. The signal symbol carries information about the amount of OFDM data symbols and their data rates. The 24-bit signal is encoded using a Convolutional encoding rate of $\frac{1}{2}$ to produce a 48-bit value. This 48-bit signal value is mapped with BPSK to the 48 OFDM sub-carriers to generate the OFDM signal symbol. The signal symbol bit assignment is shown below:

IEEE 802.11a Signal Symbol Bit Assignment

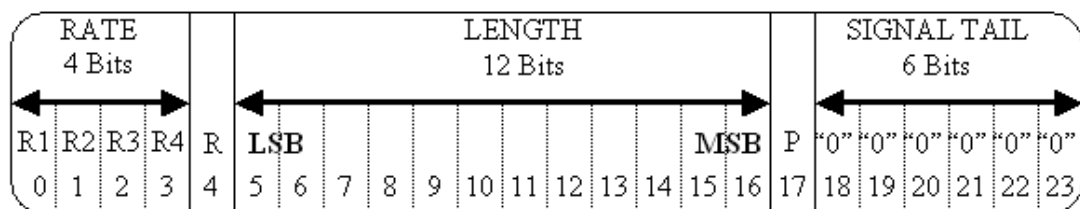


Figure 6.5: The IEEE 802.11a signal symbol bit assignment

The 4-bit rate value contains information about the current data rate of the encoded transmission. Using Table 6.3 together with Table 6.2, it is then possible to exactly determine the data rate as well as the convolutional encoding rate of the transmission. Bit 4 is reserved for future use and bit 17 is an even parity for bits 0 – 16. The signal tail field is set to zero.

Rate (Megabits/s)	R1-R4	Rate (Megabits/s)	R1-R4
6	1101	24	1001
9	1111	36	1011
12	0101	48	0001
18	0111	54	0011

Table 6.3: Contents of the IEEE 802.11a Signal Field

6.2.5 The IEEE 802.11a Data Symbol

The process described in Table 6.4 generates an IEEE 802.11a data symbol.

Nr	Action
1	The digital data set intended for transmission is passed to a convolutional encoder; the result is a data set combined with error correction data.
2	The digital data bits are the mapped to complex coefficients using the appropriate sub-carrier modulation method.
3	The complex coefficients are mapped to the correct frequency spectrum locations.
4	The reference sub-carriers complex coefficients are added to the correct frequency spectrum locations.
5	The resulting complex coefficients are converted to a time domain signal using the IDFT function block
6	A Guard Interval (Cyclic Prefix) is added to the signal
7	The resulting symbol is buffered and prepared for transmission by the hardware.

Table 6.4: IEEE 802.11a Data Symbol generation process

Finally, the complex coefficients which will be converted to a time-domain signal are given by

$$C_{-26,26} = B_{-26,26} + P_{-26,26}. \quad (6.15)$$

6.2.5.4 Converting the Data Symbol to a Time-Domain Signal

The complex time-domain signal is then generated using Equation (6.16),

$$r_{\text{DATA}}(t) = w_{\text{TDATA}}(t) \sum_{k=-N_{\text{ST}}/2}^{N_{\text{ST}}/2} (C_k \cdot e^{j 2 \pi k \Delta_F t}) \quad (6.16)$$

where $w_{\text{TDATA}}(t)$ is the symbol windowing function.

6.2.5.5 Completing the Data Symbol by Adding a Guard Interval

The complex time domain data symbol is finalised by adding a guard interval to the front of the OFDM symbol. The guard interval (cyclic prefix) is exactly $\frac{1}{4}$ of the length the OFDM symbol

$$r_{\text{SYM}}(t) = r_{\text{GI}}(t) + r_{\text{DATA}}(t - T_{\text{GI}}). \quad (6.17)$$

6.2.6 Transceiving the IEEE 802.11a Packets

After an IEEE 802.11a packet is generated, it is ready to be transmitted over the communications medium. For systems transmitting the IEEE 802.11a packets over the air using electromagnetic waves, special precaution must be taken to abide by the rules and regulations on transmission power levels for the relevant country. The final IEEE 802.11a (where C_{PACKETS} is the amount of OFDM data carriers) can be expressed using Equation (6.18),

$$r_{\text{IEEE_802.11a}}(t) = r_{\text{SHORT_TRAIN}}(t) + r_{\text{LONG_TRAIN}}(t - T_{\text{SHORT}}) + r_{\text{SIGNAL}}(t - (T_{\text{LONG}} + T_{\text{SHORT}})) + \sum_{n=0}^{C_{\text{PACKET}}-1} r_{\text{SYM}}(t - (T_{\text{LONG}} + T_{\text{SHORT}} + T_{\text{SYM}}) - nT_{\text{SYM}}). \quad (6.18)$$

The complete IEEE 802.11a OFDM packet signal $r_{\text{IEEE_802.11a}}(t)$ can now be transmitted over the communication channel.

6.3 Conclusions

In this chapter we discussed the IEEE 802.11a standard of implementing an OFDM transceiver system. An IEEE 802.11a OFDM packet or frame consists of a long preamble train followed by a short preamble train, a signal symbol and then the OFDM data symbol train. The IEEE 802.11a standard incorporates many of the methods described in the previous chapter on combating the real-life factors influencing performance. The IEEE 802.11a long and short preambles are used for initial OFDM symbol synchronisation as well as communication channel estimation and determining sampling frequency drift. The IEEE 802.11a signal symbol contains information about the length and the encoding methods used in the current transmission. The IEEE 802.11a data symbols have 48 usable sub-carriers for encoding actual data and 4 reference carriers used for phase compensation and continuous re-synchronisation of the OFDM data symbols. Each OFDM data symbol has a guard interval used in combating the effects of the channel impulse response as well as multipath delays. The IEEE 802.11a standard further incorporates a forward error correction scheme for combating the effects of noise. (Unfortunately it is not implemented in this thesis) IEEE 802.11a is a robust standard for transmitting data in noisy office environments. The processes for creating the long and short preambles, the signal symbols and the data symbols are explained in detail.

The IEEE 802.11a standard defines the different packets with their timings and details on how to generate them, but does not specify how to implement the IEEE 802.11a transmitter or receiver. The implementation is left up to the designer and depends upon many things.

Our implementation of the IEEE 802.11a transceiver system, for instance, is based on the SDR design methodology. This means that the bulk of the processing in the transmitter and receiver will be programmed in a software domain. The next chapter describes the SDR implementation of the IEEE 802.11a transceiver system detail. The IEEE 802.11a transceiver system is programmed in a programming language called C++. The IEEE 802.11a transceiver system is initially implemented on a personal computer for designing and testing, but the final intended hardware platform is an embedded system.

Chapter 7

SDR implementation of an IEEE 802.11a OFDM system

7.1 Introduction

One of the main purposes of this thesis is the study and implementation of an IEEE 802.11a OFDM transceiver system. By now we have already discussed the basic mathematics behind OFDM, which enables us to create routines for encoding and decoding OFDM symbol trains. The IEEE 802.11a implementation of OFDM incorporates features into existing OFDM symbols that aids in the decoding of the OFDM symbol trains, and helps to overcome most of the major negative performance influencing factors found in practical system. The implementation of this system is done by means of functions and routines written in a software programming language called C Plus-Plus (C++). C++ is one of the most popular programming languages in the world, with compilers available for nearly any hardware environment, from personal computers to embedded systems. By writing the OFDM encoder and decoder software in a generalised modular way, it will insure multi platform compatibility, which means that the OFDM software can easily be ported to many different hardware systems, depending upon its needed implementation. This chapter introduces the current hardware implementation as well as the main structures, libraries and functions used by the software and describes in detail the processes of encoding and decoding IEEE 802.11a OFDM packets. This chapter together with Appendices C and D should be used as a guide when working with the actual C++ source code.

7.2 Implementation Overview

The IEEE 802.11a transmitter system is implemented on the following system:

Hardware platform (PC Laptop):

- Intel® Pentium® 4 Processor
- CPU running at 2.8 GHz
- Memory: 512 MB of RAM
- Disk space: 80GB of HDD
- Sound Device: VIA AC'97 Sound Card

Software platform:

- Microsoft Windows XP Professional Version 2002 SP 2
- Borland C++ Builder Professional Version 5.0 Build 12.34
- Port Audio (Soundcard interface software) Version 18.1
- Debugging done with Matlab 6.5 (R13)

The IEEE 802.11 receiver system is implemented on the following system:

Hardware platform (PC Desktop)

- AMD Duron™ Processor
- CPU running at 1.00 GHz
- Memory: 384 MB of RAM
- Disk Space: Total of 180 GB combined HDD space
- Sound Device: VIA AC'97 Sound Card

Software Platform:

- Microsoft Windows XP Professional Version 2002 SP 2
- Borland C++ Builder Professional Version 5.0 Build 12.34
- Port Audio (Soundcard interface software) Version 18.1
- Debugging done with Matlab 6.5 (R13)

An IEEE 802.11a transmitter basically takes incoming network data, adds error correction bytes, modulates the data into OFDM signals with a bandwidth of 20 MHz, upconverts the complex baseband signal to a high frequency carrier and then transmits that signal over the air using electro-magnetic waves. The IEEE 802.11a receiver receives the signal, downconverts it to baseband, decodes the OFDM signal, corrects the faulty data, and returns the original network data as output.

The IEEE 802.11a transmitter built for the purpose of this thesis is based on the standards of the IEEE 802.11a modulation scheme, but does not include everything the specification specifies. The OFDM encoding software takes test input data, but does not add any error correction data to it. The entire OFDM signal is compressed into a 44.1kHz signal and that signal is not upconverted or transmitted over the air. Instead, the complex baseband signal is transmitted over a pair of copper wires to the receiver. The receiver receives the signals and decodes the OFDM signal, returning the original data as output.

The main differences between a fully IEEE 802.11a compliant system and the current implemented system are shown in Table 7.1.

Nr	Parameter	IEEE 802.11a system	Implemented System
1	Transmit/Receive Data	Network Data	Test Data
2	Error Correction	Convolutional Coding	None
3	OFDM signal Bandwidth	20 MHz	44.1 kHz
4	Up conversion to carrier signal	Yes	No
5	Transmission channel	Electromagnetic waves	Stereo copper cables
6	Hardware platform	Embedded Systems	Personal Computer

Table 7.1: Differences between the IEEE 802.11a system and the implemented system.

The lack of an appropriate analogue frontend, ADC and DAC makes it impossible for the system to use the intended 20 MHz of bandwidth for the OFDM signal, to upconvert the signal to the transmission frequency or to transmit the signal using radio waves. These apparent shortcomings do not however stop us from using and testing the OFDM transceiver system. By using the following Discrete Fourier Transform property, it is possible to scale the frequency of the transmitted signal, by increasing or decreasing the time signal by a certain factor:

$$s(an) \xrightarrow{\text{DFT}} \frac{1}{|a|} S\left(\frac{f}{a}\right). \quad (7.1)$$

It is thus now possible to use the ADC and DAC within a personal computer's soundcard to transmit and receive the signals. The intended 20 MHz complex baseband signal can be scaled down to 44.1 kHz, which the computer sound card can handle. The scale factor can thus be calculated as:

$$a = \frac{20\text{MHz}}{44.1\text{kHz}} = \frac{20 \times 10^6}{44.1 \times 10^3} \approx 453.5 \quad (7.2)$$

The IEEE 802.11a OFDM signal can now be scaled down to this lower frequency prior to transmission. A pair of copper stereo cables between the transmitting and receiving computer replaces the analogue front-end and acts as the communications channel.

The lack of the IEEE 802.11a forward error correction scheme at the transmitter and receiver will also not be a problem, since it will force us to examine the received data stream as it is, of which statistics, such as the bit error rates can directly be determined.

7.3 Hardware Implementation

Software Defined Radio is an architecture where all possible processing resides in a software domain. Different software packages performing different digital signal processing and mathematical processes can be combined to define the operation of the

hardware device. As mentioned, one of the advantages of SDR is that the software is portable, so it can be designed and tested on a certain hardware platform, and then later ported to the intended embedded hardware when finished.

The final intended hardware platform for this project is a small, embedded IEEE 802.11a transceiver system and is shown in Figure 7.1.

Intended IEEE 802.11a SDR Transceiver System

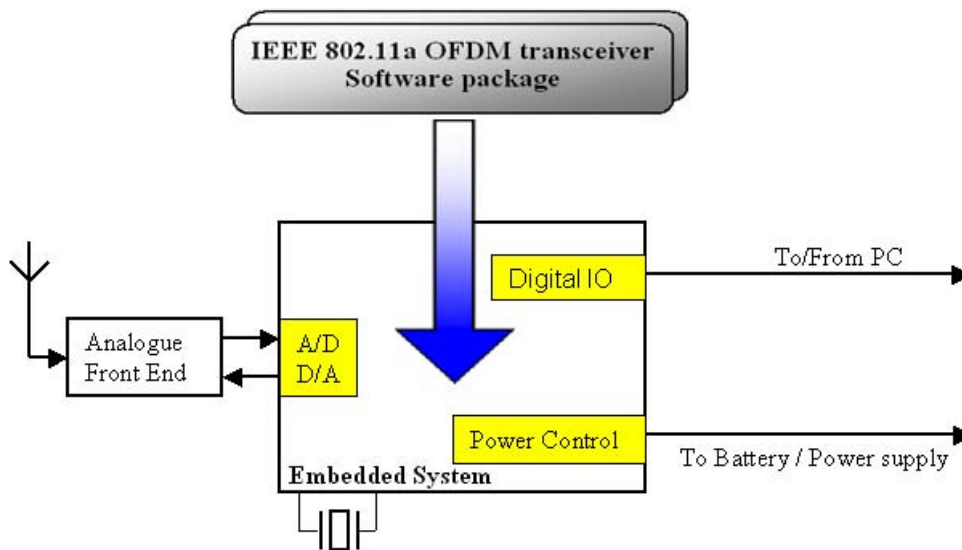


Figure 7.1: The intended IEEE 802.11a SDR transceiver system

The current SDR hardware platform is two personal computers running the IEEE 802.11a software packages. The two personal computers are connected together using copper stereo sound cables that act as the communications channel, shown in Figure 7.2.

Current IEEE 802.11a based SDR Transceiver System

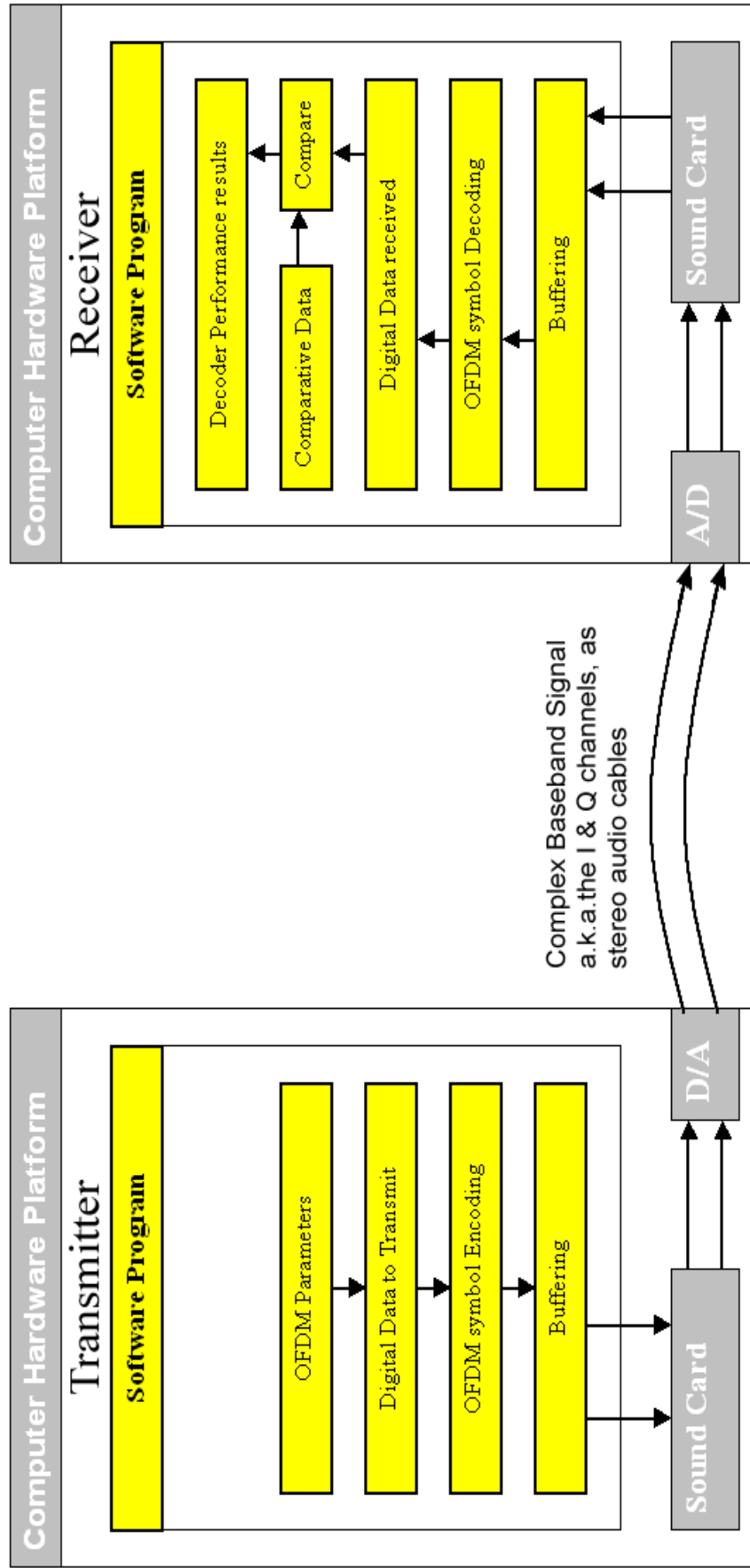


Figure 7.2: The Implemented IEEE 802.11a based system

7.4 Software Implementation

As mentioned, the IEEE 802.11a based OFDM transceiver system is programmed in C++. The complete software package consists of two major parts, namely the OFDM transceiver libraries and the Graphical User Interface (GUI). The OFDM transceiver libraries include the programs for encoding and decoding the IEEE 802.11a based OFDM symbol trains as well as a library for interfacing the transceiver with the current hardware platform through the soundcard. The GUI acts as a simple user interface for demonstrating the capabilities of the OFDM libraries.

7.4.1 The Graphical User Interface

The GUI does not do any serious processing in regards to the OFDM transceiver, but it plays an important part in demonstrating the capabilities of the OFDM libraries, and gives a clear picture of how the system works as a whole. The GUI is titled *OFDM Real time Encoding/Decoding Demo Version 1.00*, and is used as both the OFDM transmitter and the OFDM receiver. For demonstration purposes, the GUI must be running on both the transmitter and receiver system.

The GUI enables the user to perform the following functions:

- Initialise and deinitialise the OFDM libraries.
- Initialise and deinitialise the audio devices.
- Selecting of a suitable audio device from a list of available devices.
- Encoding and transmission of IEEE 802.11a OFDM data:
 - Initiate the OFDM transmitter system. (For transmission of a test image or a test data packet)
 - Load transmission data. (A test image, or test data packet)
 - Displays the transmission data, if it is a test image.
 - Encodes the transmission data into an IEEE 802.11a OFDM data packet.
 - Adds AWGN to the OFDM data packet, if desired.
 - Transmits the IEEE 802.11a data packet across the audio device.

- Transmits the IEEE 802.11a data packet to a local data file, for test purposes.
- Receiving and decoding of IEEE 802.11a OFDM data:
 - Initiate the OFDM receiver system. (For transmission of a test image or a test data packet)
 - Enable or disable the reference carrier phase compensation algorithm in the decoder.
 - Receive and decode the IEEE 802.11a OFDM data from an audio device.
 - Receive and decode the IEEE 802.11a OFDM data from a local file.
 - Displays the received test image.
 - Load comparative data. (An test image or a test data packet)
 - Displays the comparative test image.
 - Compare the decoded received data against the comparative data.
 - Displays the total amount of data bits transferred.
 - Displays the total amount of incorrect data bits received.
 - Displays the Bit-error rate of the received data.
- Calibrates Sound Device Volume Settings:
 - Transmits a full volume sine wave for 10 seconds over the sound device for calibration purposes.
 - Receives the calibration sine wave and determines whether any clipping has occurred, in which case the transmitted volume needs to be reduced.
- Useful user feedback messages in the main message window, shows the user exactly what the program is doing.

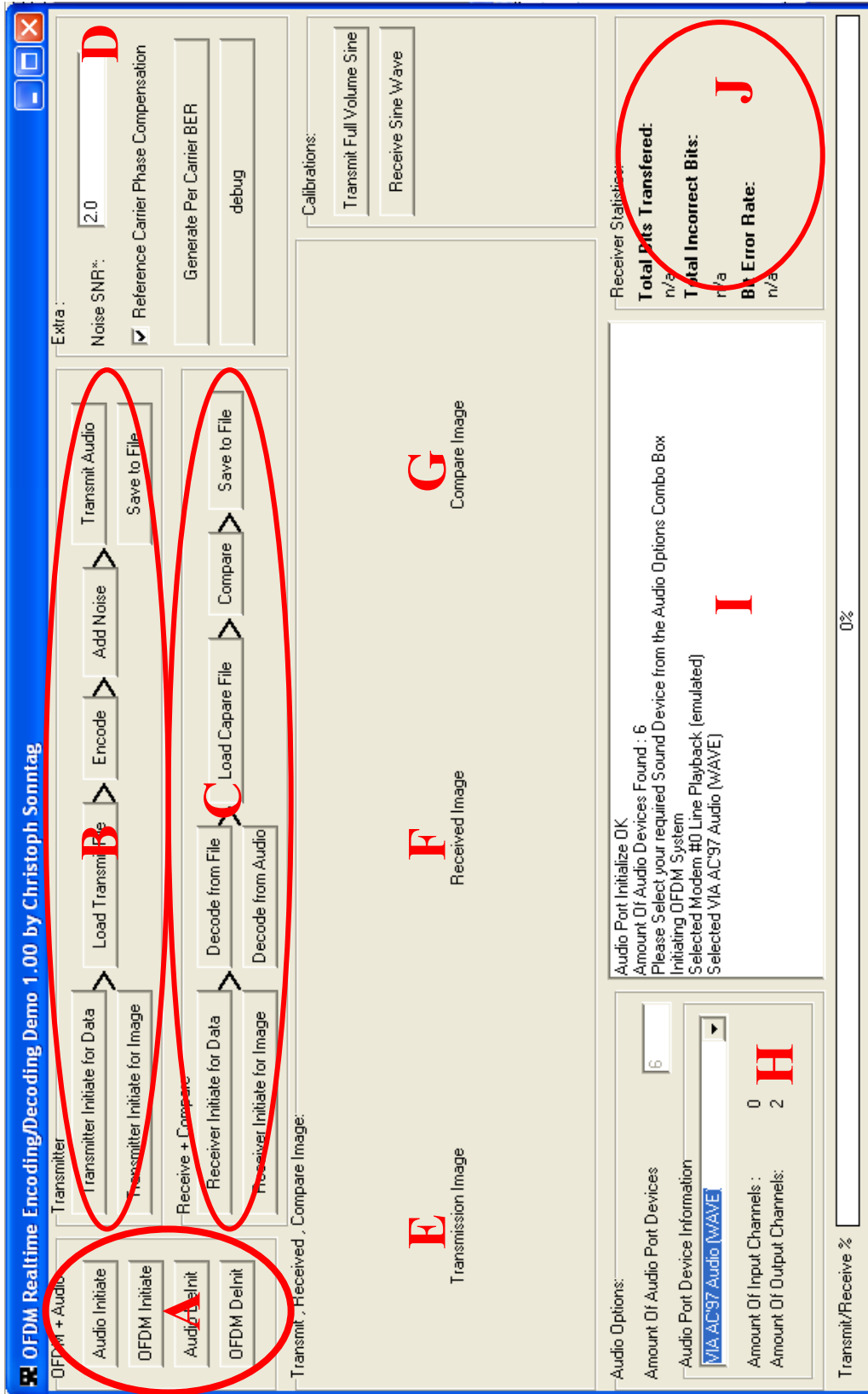


Figure 7.3: The OFDM Encoding/Decoding Demo GUI

Table 7.2 explains some of the functions available on the OFDM Encoding/Decoding Demo GUI. The Code parameter refers to the marked locations on Figure 7.3.

Code	Functions	Function Details
A	Audio Initiate, Audio Deinitiate, OFDM Initiate, OFDM Deinitiate	Initialisation and de-initialisation of the OFDM libraries and the audio devices. At start up, one needs to Initiate the Audio and OFDM.
B	Transmitter initiate for Data; Transmitter Initiate for Image; Load transmit file; Encode; Add Noise; Transmit Audio; Save to file	OFDM Transmitter routines: Initiating the OFDM transmitter for transmitting a test image or a test data packet; loading the data; encoding the IEEE 802.11a OFDM packet; adding AWGN to the transmission; Transmitting the signal over the sound device or to a local test file
C	Receiver Initiate for Data; Receiver Initiate for Image; Decode From File; Decode From Audio; Load comparer File; Receive Audio, Compare; Save To File	OFDM Receiver routines: Initiating the OFDM receiver for receiving and decoding a test image or a test data packet; decoding the signal from file or from the sound device; Load the comparative data; Comparing the received data to the comparative data; Saving the received transmission to file.
D	Noise SNR	Setting transmit AWGN SNR in dB
E	Transmission Image	Displays the transmission test image
F	Received Image Display	Displays the received test image
G	Comparative Image	Displays the comparative test image
H	Audio Options	Audio Device Options and selection menu
I	Message Window	The message window shows current program activity and messages
J	Receiver-Comparer Statistics	Receiver data statistics shows amount of received bits, erroneous received bits and BER

Table 7.2: Guide to the functions of the OFDM Encoding/Decoding Demo GUI

7.4.2 The Software Structures

C++ structures are basically a group of variables and buffers grouped under a common name for ease of use. The OFDM transceiver software uses five main structures to group the different variables that are used by the different software libraries. Structures are passed to and from functions that process the variables contained within them.

The five different structures are:

1. The OFDM specifications structure
2. The OFDM transmitter structure
3. The OFDM receiver structure
4. The OFDM comparer structure
5. The OFDM audio structure

7.4.2.1 The OFDM specification structure

The main OFDM specification structure contains information on the physical makeup of the OFDM signal, which includes OFDM frequency and timing specifications. This structure is known by its C++ name: **OFDM_Specification_struct** and is located in the C++ file: **OFDM.h**. For a list of complete structure details please refer to Appendix C.1.

An overview of the information stored in the OFDM specification structure is:

- OFDM frequency specifications.
- OFDM signal timings.
- OFDM signal sample sizes.
- OFDM sub-carrier modulation specifications.
- OFDM data sub-carrier frequency spectrum locations.
- OFDM reference sub-carrier frequency spectrum locations.
- OFDM reference sub-carrier values.

- IEEE 802.11a OFDM long preamble buffer.
- IEEE 802.11a OFDM long preamble guard interval buffer.
- IEEE 802.11a OFDM short preamble buffer.
- IEEE 802.11a OFDM signal symbol buffer.
- The Discrete Fourier Transform (DFT) twiddle factors.
- The Inverse Discrete Fourier Transform (IDFT) twiddle factors.
- The real and imaginary time domain buffers for the IDFT and DFT.
- The real and imaginary frequency spectrum buffers for the IDT and DFT.
- The magnitude and phase frequency spectrum buffers for the IDFT and DFT.
- Boolean checks to indicated an initiated OFDM specification structure.

7.4.2.2 The OFDM transmitter structure

The OFDM transmitter structure contains information about the transmission data, the transmission OFDM encoding process and contains pointers to the signal transmission arrays. This structure is known by its C++ name: **OFDM_transmitter_struct** and is located in the C++ file: **OFDM_transmitter.h**. For a list of complete structure details please refer to Appendix C.2.

An overview of the information stored in the OFDM transmitter structure is:

- The total amount of bytes and bits in the transmission.
- The total amount of data symbols in the transmission.
- Sub-carrier and OFDM modulation information.
- The buffers that store the transmission bits and bytes.
- The transmission signal I (real) & Q (imaginary) buffers.
- A single symbol temporary I (real) & Q (imaginary) buffers.
- The transmit filename and file type.
- The Signal-to-Noise ratio of the added noise. (if used)
- Boolean checks to indicate initiated transmit data and buffers.
- Boolean checks to indicate complete transmission encoding.

7.4.2.3 The OFDM receiver structure

The OFDM receiver structure contains information about the process of receiving and decoding OFDM signals. The structure is known by its C++ name: **OFDM_receiver_struct** and is located in the C++ file: **OFDM_receiver.h** . For a list of complete structure details please refer to Appendix C.3.

An overview of the information stored in the OFDM receiver structure is:

- The expected amount of OFDM data symbols.
- Sub-carrier and OFDM modulation information.
- The receiving signal I (real) & Q (imaginary) buffers.
- A primary decoding/receiving buffer.
- Receive-from-file filename and information.
- Receiver state machine current state information.
- Energy threshold values. (State 1)
- IEEE 802.11a short preamble cross-correlation signal buffers. (State 2)
- IEEE 802.11a short preamble cross-correlation results. (State 2)
- IEEE 802.11a long preamble cross-correlation signal buffers. (State 2)
- IEEE 802.11a long preamble cross-correlation results. (State 2)
- Combined IEEE 802.11a short and long preamble cross-correlation signal buffers. (State 2)
- Combined IEEE 802.11a short and long preamble cross-correlation results. (State 2)
- IEEE 802.11a short preamble start position. (State 2)
- IEEE 802.11a long preamble start position. (State 2)
- IEEE 802.11a signal symbol start position (State 2)
- IEEE 802.11a first data symbol start position. (State 2)
- Channel transfer function details from IEEE 802.11a long preamble. (State 3)
- IEEE 802.11a signal symbol information. (State 3)
- Miscellaneous decoding variables. (State 4)
- OFDM data sub-carrier frequency spectrum locations. (State 4)

- OFDM reference sub-carrier frequency spectrum locations.
- Buffers to store the decoded data bits and bytes.
- Buffer to store the OFDM sub-sample offset values.
- Boolean checks to indicate decoding errors.
- Boolean checks to indicate decoding completion.

7.4.2.4 The OFDM comparer structure

The OFDM comparer structure contains information about the comparative data, which is used to compare the received data to. It also contains some statistics about the received OFDM data. The structure is known by its C++ name: **OFDM_comparer_struct** and is located in the C++ file: **OFDM_comparer.h**. For a list of complete structure details please refer to Appendix C.4.

An overview of the information stored in the OFDM comparer structure is:

- The comparative data.
- The total data bits of the comparative data.
- The total data bits of the received data.
- The total amount of erroneous data bits in the received data.
- The bit-error-rate (BER) of the received data.
- A Boolean check to indicate the comparing process is complete.

7.4.2.5 The OFDM audio structure

The OFDM audio structure contains information about the audio device interface and links it to the received/transmitted data. The structure is known by its C++ name: **OFDM_audio_struct** and is located in the C++ file: **OFDM_audio.h**. For a list of complete structure details please refer to Appendix C.5.

An overview of the information stored in the OFDM audio structure is:

- A Boolean check to indicate that the audio initialisation is complete.

- Information about the number of audio devices found on the system.
- Information about each audio device on the system.
- Information about the current selected audio device.
- Boolean checks to indicate selected and ready audio devices.

There are two other structures used by the independent audio libraries to interface with the transmission data. They act as temporary interface structures and stores information about current audio device activities and buffers. The two structures are known by their C++ names as, PaSoundCardPlaybackDataStruct and paSoundCardRecordedDataStruct. They are never directly used or altered by the OFDM software.

7.4.3 The Software Libraries

The IEEE 802.11a OFDM transceiver libraries include all the functions, variables and structures for encoding and decoding the IEEE 802.11a based OFDM symbols and symbol trains. Libraries are basically container files, which houses different structures, variables and functions. It is the part of the software that can be ported for use on different hardware platforms. Included in these libraries is the OFDM audio interface library that interfaces the transceiver with the current hardware implementation.

The six main OFDM libraries are:

1. The main OFDM library.
2. The OFDM mathematics library.
3. The OFDM transmitter library.
4. The OFDM receiver library.
5. The OFDM comparer library.
6. The OFDM audio interface library.

These libraries work together to process the data and encode or decode OFDM symbol trains.

7.4.3.1 The Main OFDM Library

The main OFDM library is basically used to create and initialise the main OFDM structures, buffers and variables, and contains some basic functions, which can be used by all the other libraries. The two C++ files which contains the main OFDM library are: **OFDM.cpp** and **OFDM.h**. The Main OFDM library is basically the container file for the OFDM specifications structure, and all the functions to use and manipulate it. For a list of function details please refer to Appendix C.6.

The functions within the main library basically do the following:

- Initiates the OFDM specifications structure.
- Cleans time and frequency DFT/IDFT buffers.
- The IDFT function block.
- The DFT function block.
- Converts the real and imaginary frequency spectrum to its equivalent magnitude and phase spectrum.
- Creates the DFT and IDFT twiddle factors.
- Creates the IEEE 802.11a long preamble symbol.
- Creates the IEEE 802.11a long preamble guard interval.
- Creates the IEEE 802.11a short preamble symbol.
- De-initiates the OFDM specification structure.

7.4.3.2 The OFDM Mathematics Library

The OFDM mathematical library contains mathematical functions that is used in the transmission and receiving of OFDM data symbol trains. The two C++ files which contains the OFDM mathematic library are: **OFDM_Math.cpp** and **OFDM_Math.h**. For a list of function details please refer to Appendix C.7.

The functions within the mathematical library basically do the following:

- Cross-correlation between two data sets.
- Search for maximum values and their locations inside buffers.
- Search for minimum values and their locations inside buffers.
- Calculate the root-mean-square (RMS) of a buffer.
- Map data bits (or arrays thereof) to QPSK phases.
- De-map single QPSK phases (or arrays thereof) to data bits.
- Convert a 8-bit binary array to a byte (or arrays thereof).
- Convert a byte (or arrays thereof) to an 8-bit binary array.
- Rounding of a float-type variable.
- Return the reverse of an array.
- Return the conjugate of an array.
- Convert a normalised float to a 16-bit word and then to two 8-bit bytes.
- Convert two 8-bit bytes to a 16-bit word and then a normalised float.
- Calculate the absolute value of a float-type C++ variable.
- Return the quadrant in which a phase is located.
- Calculate the amount of differences between two data sets.
- Unwrap a phase value and calculate the minimum distance to 0 degrees.
- Unwrap a phase value and calculate the minimum distance to 180 degrees.

7.4.3.3 The OFDM Transmitter Library

The OFDM transmitter library contains functions and a structure used in the encoding and transmitting of IEEE 802.11a OFDM data streams. The two C++ files which contains the OFDM transmitter library are: **OFDM_transmitter.cpp** and **OFDM_transmitter.h**. The OFDM transmitter library contains the OFDM transmitter structure and functions for initiating and manipulating it. For a list of function details please refer to Appendix C.8.

The functions within the OFDM transmitter library basically do the following

- Initiates the OFDM transmitter structure.
- Get the filename and load the transmission data. (image or data packet)
- Adds all the IEEE 802.11a preambles to the transmission buffer.
- Adds the IEEE 802.11a signal symbol to the transmission buffer.
- Convert IEEE 802.11a OFDM data phases to complete IEEE 802.11a data symbols.
- Adds a IEEE 802.11a data symbol to the transmission buffer.
- Saves the IEEE 802.11a transmission to file.
- Adds AWGN to the IEEE 802.11a transmission file.

7.4.3.4 The OFDM Receiver Library

The OFDM receiver library contains functions and a structure used in the receiving and decoding of IEEE 802.11a OFDM data streams. The two C++ files which contains OFDM receiver library are: **OFDM_receiver.cpp** and **OFDM_receiver.h**. The OFDM receiver library houses the OFDM receiver structure and functions for initiating and manipulating it. For a list of function details refer to Appendix C.9.

The functions within the OFDM receiver library basically do the following:

- Initiates the OFDM receiver structure, by calling functions to:
 - Create and clear the receiving buffers.
 - Setup the receiving buffers and variables.
- Get the filename, the data and data size when decoding from local test files.
- Start decode from a local file, by loading the test file into a transmission buffer and then calling the decoder state machine.
- Start decode from a sound buffer, by linking the received sound buffer to the decoder state machine.
- Start the decoder state machine. (buffer decoder)
- Decoder State 1, Signal Energy Detection.

- Decoder State 2, IEEE 802.11a short and long preamble train cross-correlation. (complete initial synchronisation)
- Decoder State 3, extract the IEEE 802.11a long preamble information for use in channel transfer function estimation, extract data from the signal symbol.
- Decoder State 4, extract and decoder IEEE 802.11a data from the transmission.
- Decoder State 4, calls the functions that:
 - Create phase shifts from a range sub-sample offsets.
 - Adds the phase shifts to current reference phase values.
 - Calculate best-fit solution for sub-sample offset.
 - Compensate OFDM phases for sub-sample offset.
- Save the received transmission to file.
- Update the GUI received image.

7.4.3.5 The OFDM Comparer Library

The OFDM comparer library contains functions and a structure used in comparing the received IEEE 802.11a OFDM data stream with comparative data, to produce performance result statistics. The two C++ files which contains the OFDM comparer library are: **OFDM_comparer.cpp** and **OFDM_comparer.h**. The OFDM comparer library also houses the OFDM comparer structure and functions for initiating and manipulating it. For a list of function details refer to Appendix C.10.

The functions within the OFDM comparer library basically do the following

- Initiates the OFDM comparer structure.
- Get the filename and load the comparative data.
- Compare the received data with the comparative data.

7.4.3.6 The OFDM Audio Library

The OFDM audio library contains functions and a structure used by the OFDM transceiver program to interface with the hardware audio devices. The two C++ files which contains the OFDM audio library are: **OFDM_audio.cpp** and **OFDM_audio.h**. The OFDM audio library also houses the OFDM audio structure and functions for initiating and manipulating it. For a list of function details refer to Appendix C.11.

The functions within the OFDM audio library basically do the following;

- Initiates the OFDM audio structure and the audio devices.
- Callback functions for audio playback and recording.
- Select appropriate audio devices.
- Playback the IEEE 802.11a encoded transmission.
- Playback the calibration sine wave.
- Record the IEEE 802.11a encoded transmission.
- Record the calibration sine wave.
- Evaluate the calibration sine wave.
- Close the audio devices.

7.4.4 Conclusions on the software implementation

So far, the software structures and libraries has been introduced and quickly explained, but as separate entities they can't perform any real tasks. Specific library functions and structures together with the GUI need to be linked together to form a logical chain of events which will produce the desired results, such as encoding digital data into IEEE 802.11a OFDM packets, or receiving the IEEE 802.11a OFDM packets and decoding them into digital data. The next section will attempt to explain in detail what happens in each part of the IEEE 802.11a transceiver system, in regards to function, variables, buffers, structures and libraries.

7.5 The Complete IEEE 802.11a Transceiver System

The intended IEEE 802.11a hardware platform and the basic software structures and libraries have been introduced, and it is now possible to describe the implemented IEEE 802.11a transceiver in detail.

7.5.1 The IEEE 802.11a Transmitter

The implemented IEEE 802.11a transmitter can be explained using the following flow diagram in Figure 7.4.

Implemented IEEE 802.11a OFDM Transmitter

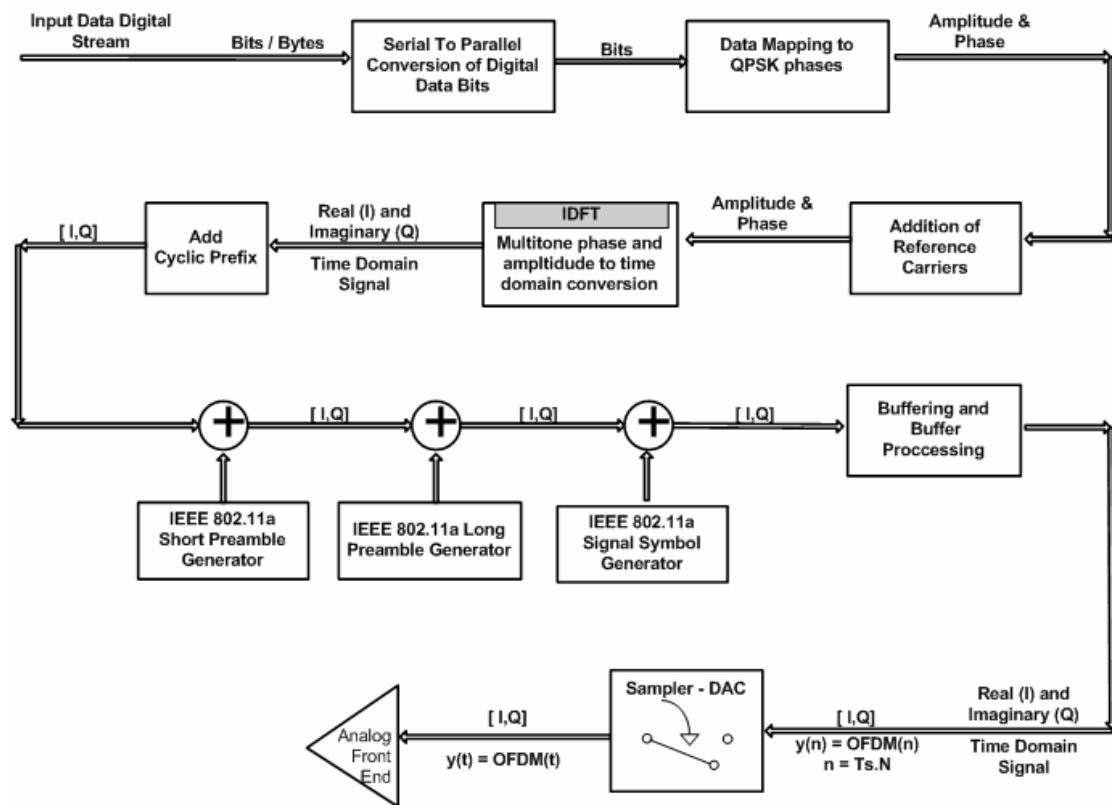


Figure 7.4: The implemented IEEE 802.11a OFDM transmitter flowchart

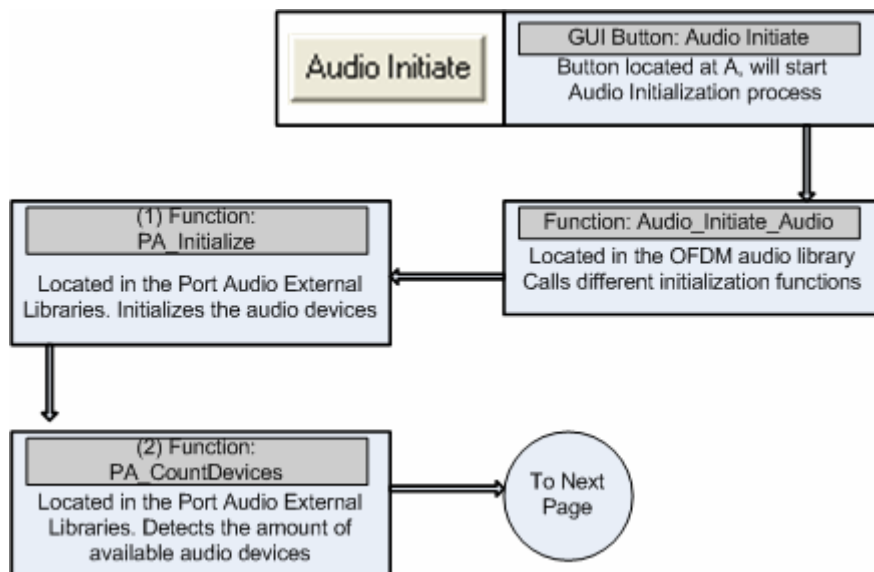
7.5.1.1 The Encoding/Transmitting Process Overview:

The complete IEEE 802.11a encoding process can be summed up as:

1. Initiating the OFDM audio system.
2. Initiating the main OFDM specifications.
3. Selecting an appropriate sound device for output. (if needed)
4. Initiating the OFDM transmitter for a test data packet or a test image.
5. Loading the appropriate test data.
6. Encoding the test data into an IEEE 802.11a OFDM packet.
7. Adding AWGN to the transmission. (if needed)
8. Transmitting the IEEE 802.11a OFDM packet to file or to the sound device.

7.5.1.2 Initiating The OFDM Audio system

Initiating the OFDM audio system is done by pressing the “Audio Initiate” button located at “A” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **Audio_Initiate_Audio** function located in the **OFDM_audio** library. The flowchart in Figure 7.5 explains the processes, functions and actions involved in initiating the OFDM audio devices



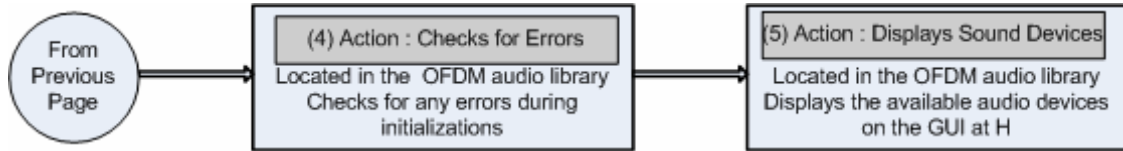


Figure 7.5: The OFDM audio initiation flowchart

7.5.1.3 Initiating The Main OFDM Specifications

Initiating the main OFDM specifications is done by pressing the “OFDM Initiate” button located at “A” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **OFDM_Initiate_All** function located in the **OFDM** library. The flowchart in Figure 7.6 explains the processes, functions and actions involved in initiating the main OFDM specifications.

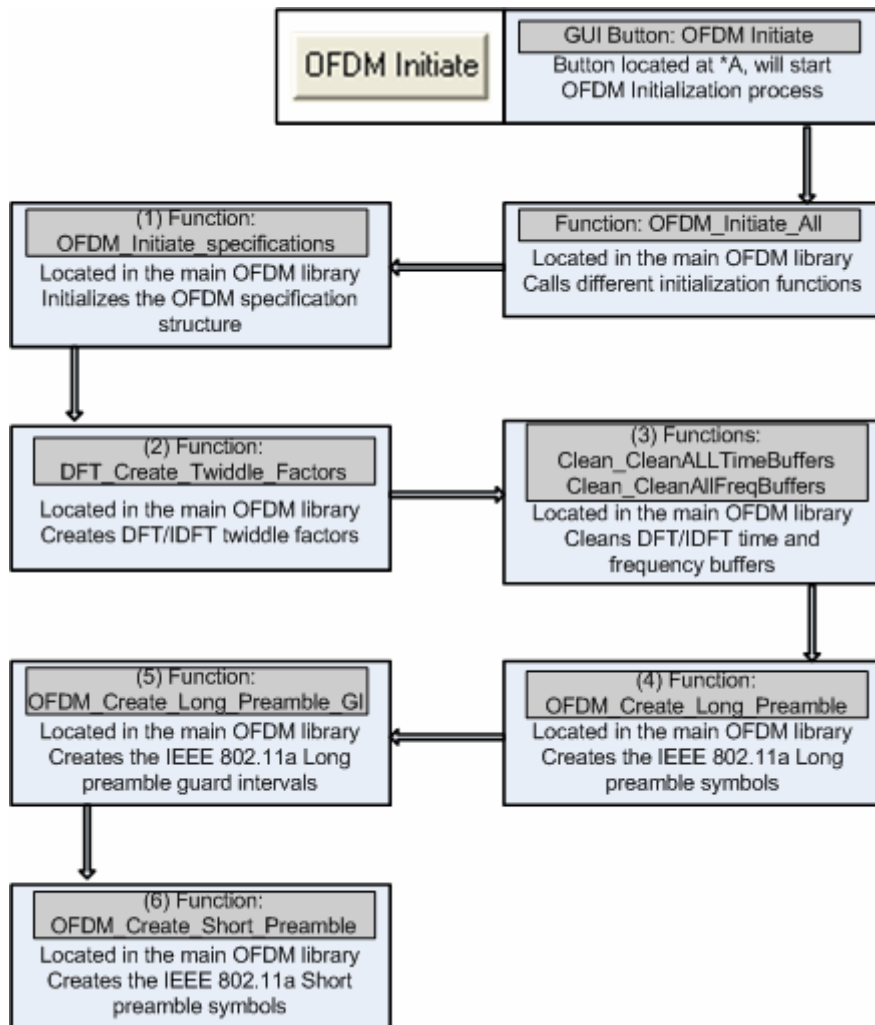


Figure 7.6: The main OFDM specifications initiation flowchart

7.5.1.4 Selecting an Appropriate Sound Device For Output

Selecting an appropriate sound device with output capabilities is done by selecting one of the available sound devices in the combo box located at “H” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the function: **Audio_Change_Audio_Device_With_Combobox** located in the **OFDM_audio** library. The flowchart in Figure 7.7 explains the processes, functions and actions involved in selecting different sound devices

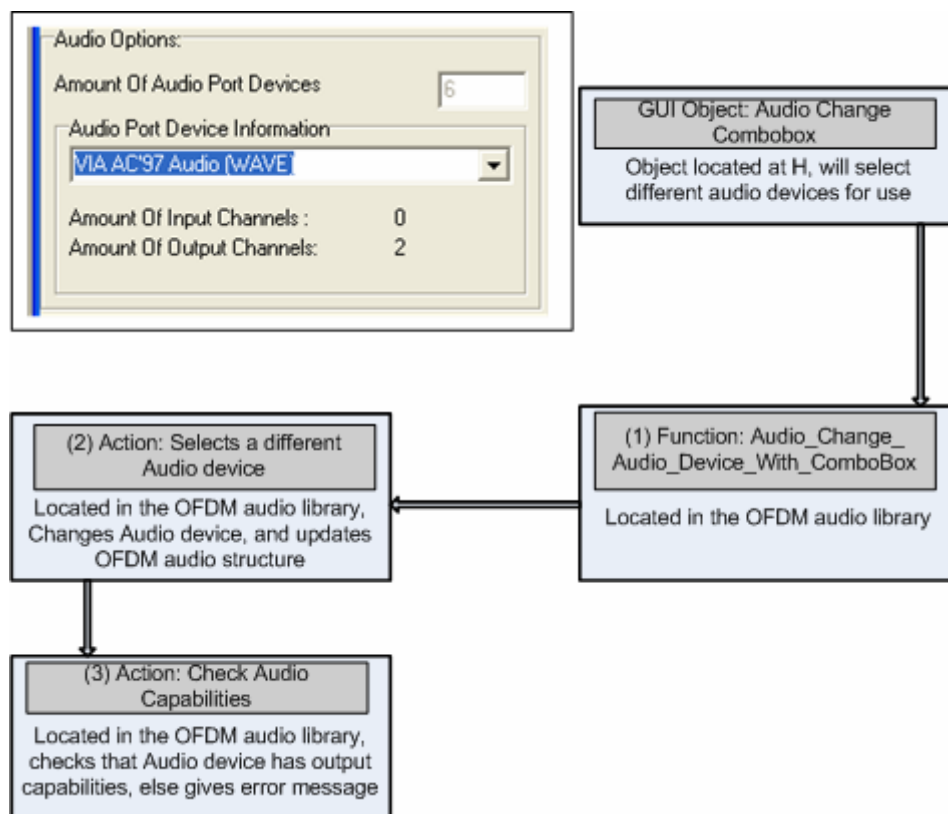


Figure 7.7: The selection of an appropriate sound device for output flowchart

7.5.1.5 Initiating The OFDM Transmitter

Initiating the OFDM transmitter for the transmission of a test data packet is done by pressing the “Transmitter Initiate for Data” button located at “B” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **OFDM_Transmitter_Init_Data_Transmitter** function located in the **OFDM_transmitter** library.

Initiating the OFDM transmitter for the transmission of a test image is done by pressing the “Transmitter Initiate for Image” button also located at “B” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **OFDM_Transmitter_Init_Image_Transmitter** function also located in the **OFDM_transmitter** library. The flowchart in Figure 7.8 explains the processes; functions and actions involved in initiating the OFDM transmitter for both test data and test image transmission.

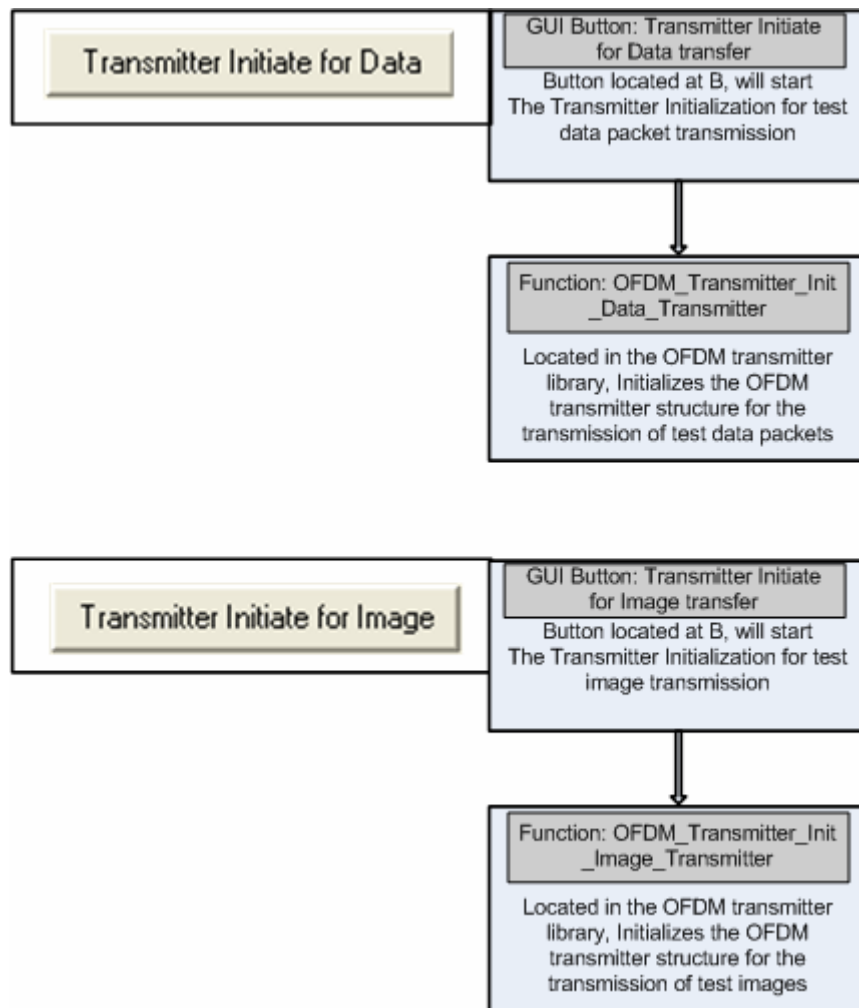


Figure 7.8: The OFDM transmitter initiation flowchart

7.5.1.6 Loading The Appropriate Test Data

The appropriate test data is loaded by pressing the “Load Transmit File” button located at “B” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **OFDM_Transmitter_Load_Transmit_File** function located in the **OFDM_transmitter** library. The flowchart in Figure 7.9 explains the processes, functions and actions involved in loading the test data.

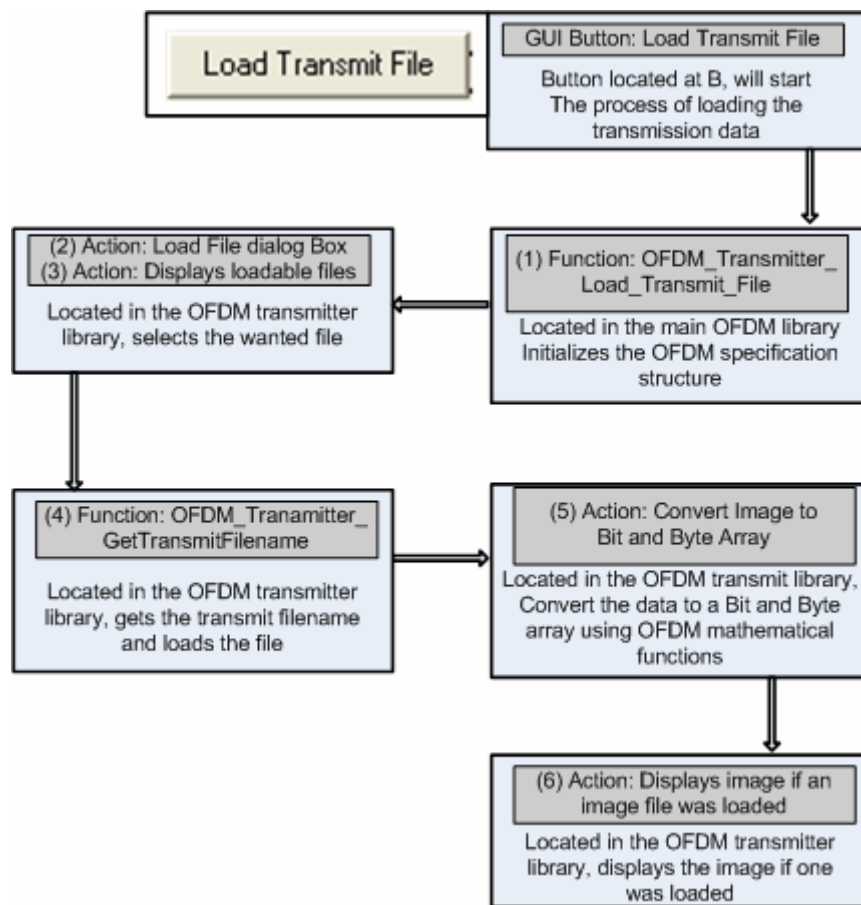


Figure 7.9: The loading of a transmission file flowchart

The GUI program is capable of loading and transmitting two types of data. The first is a test image and the second is a smaller test data packet. The test images are mainly used for demonstrative purposes and simulation tests and are relatively large in size. The test data packets are smaller in size and are mainly used in the real-time buffered system tests.

The test image parameters are chosen very specifically so that its OFDM encoded data occupies exactly 4096 data symbols, which is also the maximum allowed OFDM data symbols according to the IEEE 802.11a specifications. The image parameters are shown in Table 7.3.

Image Parameter	Value
Image format	Standard uncompressed Bitmap
Image Height	192
Image Width	256
Image Colour Depth	8 bits (1 byte) greyscale

Table 7.3: Test image parameters.

The total amount of image bits can be calculated as the product of its height, width and image colour depth,

$$\text{Total Image Bits} = (\text{Width})(\text{Height})(\text{Colour Depth}), \quad (7.3)$$

which is

$$\text{Total Image Bits} = (256)(192)(8) = 393216. \quad (7.4)$$

The total amount of digital data bits an IEEE 802.11a OFDM packet can encode can be calculated as the product of the bits per data carrier, the amount of data carriers per OFDM symbol and the total amount of OFDM data symbols,

$$\begin{aligned} \text{Total Encoded Bits} = & (\text{Total OFDM data symbols}) \\ & \times (\text{Data Carriers per Symbol}) \\ & \times (\text{Bits per Data Carrier}), \end{aligned} \quad (7.5)$$

which is

$$\text{Total Encoded Bits} = (4096)(48)(2) = 393216. \quad (7.6)$$

The results from (7.4) and (7.6) are the same, which proves the fact that the test images will occupy exactly 4096 OFDM data symbols.

The test data packets are smaller than the test images, and have only 6144 bytes or 49152 bits each. Rewriting Equation (7.5), we can calculate the amount of OFDM data symbols the test data packets will occupy as

$$\text{OFDM data symbols} = \frac{(\text{Total Encoded Bits})}{(\text{Data Carriers per Symbol})(\text{Bits per Data Carrier})}, \quad (7.7)$$

which is

$$\text{OFDM data symbols} = \frac{(49152)}{(48)(2)} = 512. \quad (7.8)$$

The test data packets are small with an estimated 1-second transmission time over the computer sound card, in opposed to the estimated 7-second transmission time of the test images. As mentioned, the small data packets are mainly used in the real-time buffered transmission test. It reduces the total testing time due to the massive amount of tests needed to statistically validate the system.

7.5.1.7 Encoding The Test Data

Encoding the test data into an IEEE 802.11a OFDM packet is done by pressing the “Encode” button located at “B” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **OFDM_Transmitter_Encode** function located in the **OFDM_transmitter** library. The flowchart in Figure 7.10 explains the processes, functions and actions involved in encoding the IEEE 802.11a OFDM packet.

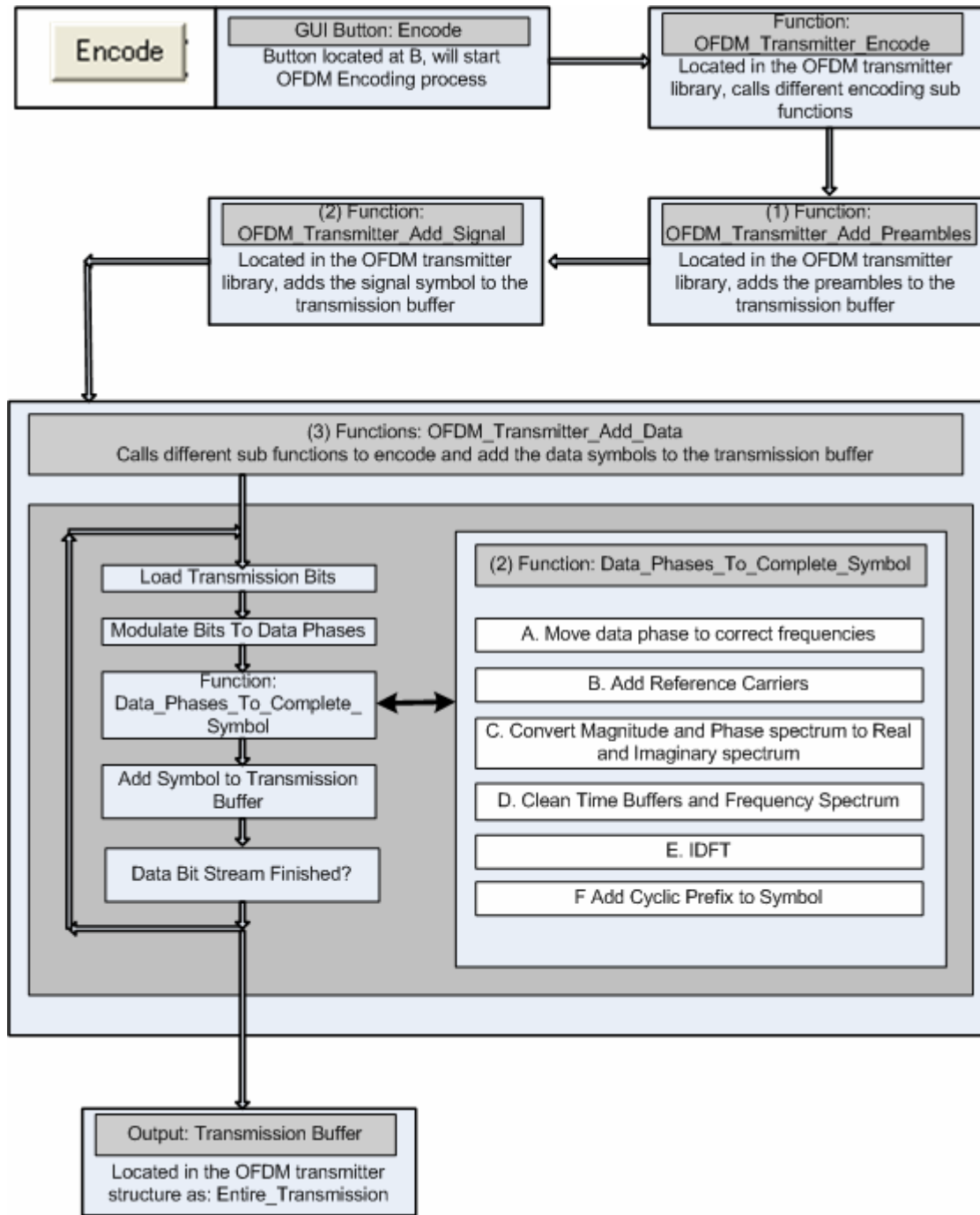


Figure 7.10: The encoding of an IEEE 802.11a OFDM packet flowchart

7.5.1.8 Adding AWGN To The Transmission

Adding AWGN to the transmission is done by pressing the “Add Noise” Button located at “B” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **OFDM_Transmitter_Add_Noise** function located in the **OFDM_transmitter** library. The SNR of the AWGN is determined by the value entered at “Noise SNR” text box located at “D” on the OFDM encoding/decoding Demo GUI in Figure 7.3. The flowchart in Figure 7.11 explains the processes, functions and actions involved in adding AWGN to the transmission signal.

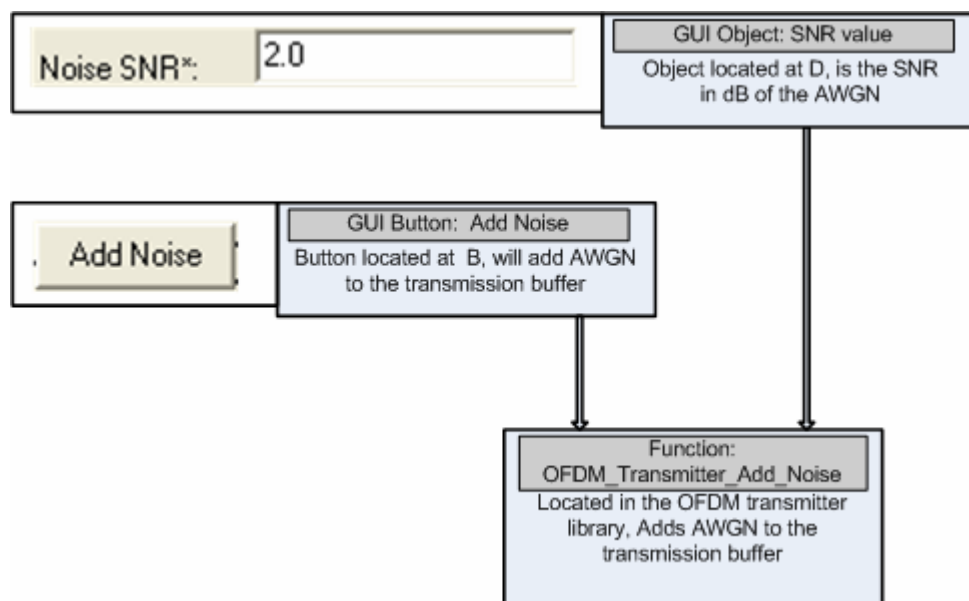


Figure 7.11: Adding AWGN to the transmission signal flowchart

7.5.1.9 Transmitting The IEEE 802.11a OFDM Packet

Transmitting the IEEE 802.11a OFDM packet to the sound device is done by pressing the “Transmit Audio” button located at “B” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **Audio_Play_Transmission** function located in the **OFDM_audio** library. Transmitting the IEEE 802.11a OFDM packet to a file is done by pressing the “Save to File” button located at “B” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **OFDM_Transmitter_Save_Transmission_To_Custom_File** located in the

OFDM_transmitter library. The flowchart in Figure 7.12 explains the processes, functions and actions involved in adding AWGN to the transmission signal.

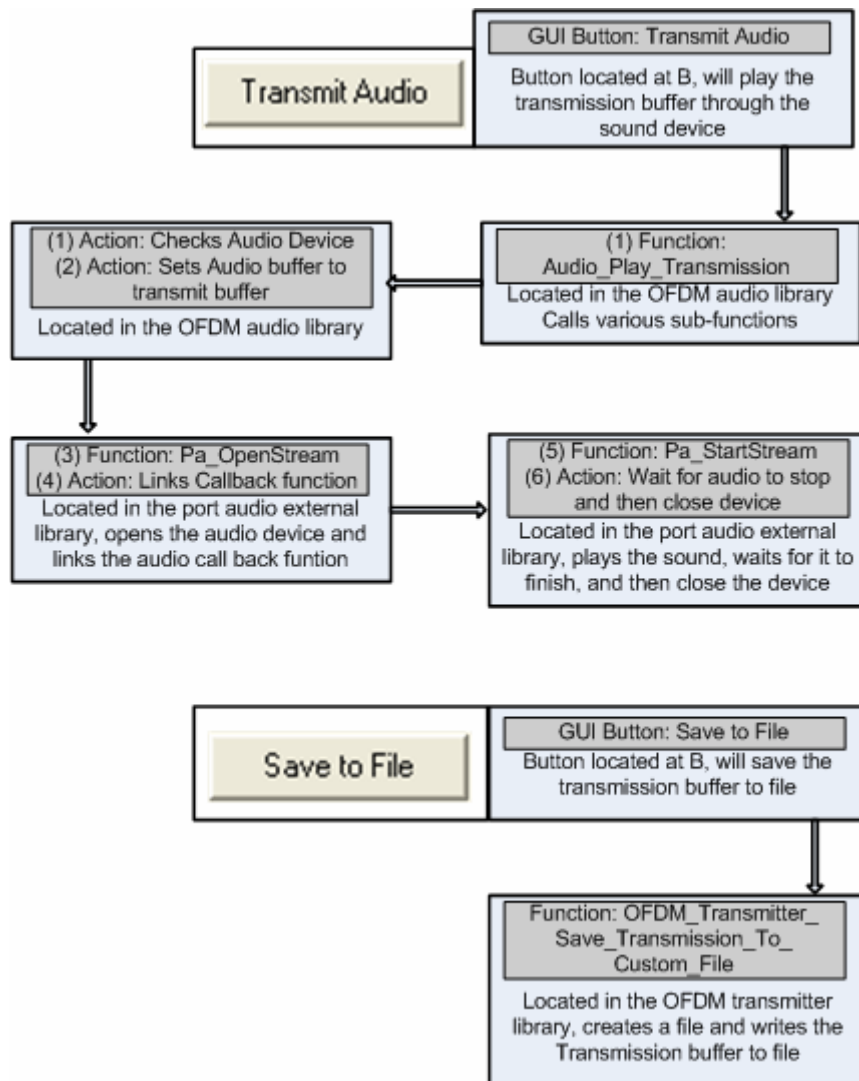


Figure 7.12: Transmitting the IEEE 802.11a OFDM packet flowchart

7.5.1.10 Conclusions On The Encoding/Transmitting Process

Sections 7.5.1.2 through 7.5.1.9 attempts to logically explain the process of encoding and transmitting IEEE 802.11a OFDM packets. The real software libraries, structures and functions are in actual fact much more complicated. It is recommended that the information about the graphic user interface (GUI) the software structures, the software libraries, the explained processes involved, together with Appendices C and D be used as a guide when working with the raw C++ source code.

7.5.2 The IEEE 802.11a Receiver

The implemented IEEE 802.11a receiver can be explained using the following flow diagram.

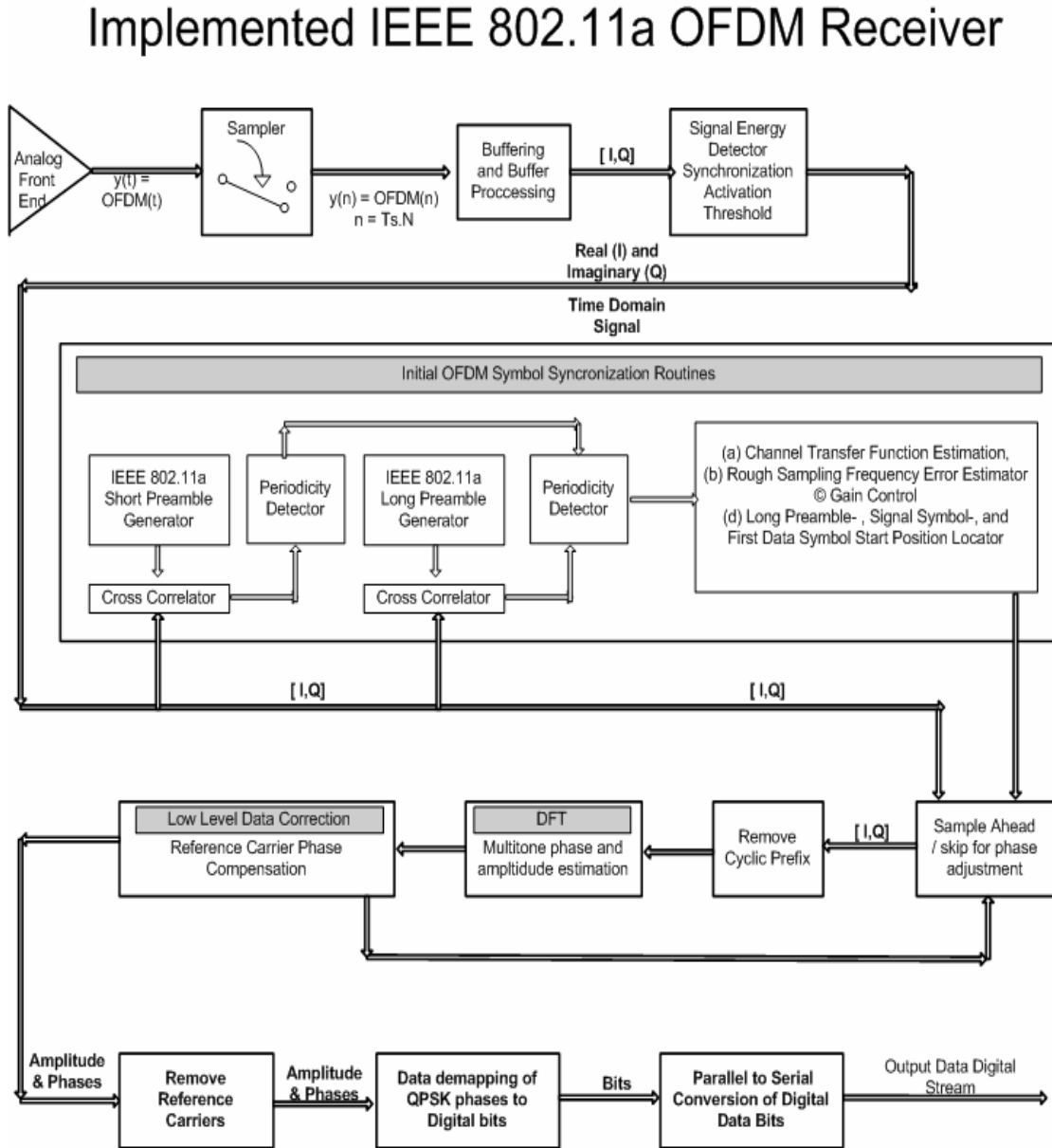


Figure 7.13: The implemented IEEE 802.11a OFDM receiver flowchart

7.5.2.1 The Receiving/Decoding Process Overview

The complete IEEE 802.11a decoding process can be summed up as:

1. Initiating the OFDM audio system.
2. Initiating the main OFDM specifications.
3. Selecting an appropriate sound device for input, if desired.
4. Initiating the OFDM receiver for a test data packet or a test image.
5. Decoding IEEE 802.11a signals.
6. Loading a compare file.
7. Comparing received and comparative data.
8. Saving the transmission to file.

7.5.2.2 Initiating The OFDM Audio System

Initiating the OFDM audio system is done by pressing the “Audio Initiate” button located at “A” on the OFDM encoding/decoding GUI in Figure 7.3. This process is exactly the same as in the case of the IEEE 802.11a transmitter; refer to Section 7.5.1.2 for more detail.

7.5.2.3 Initiating The Main OFDM Specifications

Initiating the main OFDM specifications is done by pressing the “OFDM Initiate” button located at “A” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This process is exactly the same as in the case of the IEEE 802.11a transmitter, so please refer to Section 7.5.1.3 for more detail.

7.5.2.4 Selecting an Appropriate Sound Device For Input

Selecting an appropriate sound device with input (recording) capabilities is done by selecting one of the available devices in the combo box located at “H” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This process is exactly the same as in the case of the IEEE 802.11a transmitter: Refer to Section 7.5.1.4 for more detail.

7.5.2.5 Initiating The OFDM Receiver

Initiating the OFDM receiver for the transmission of a test data packet is done by pressing the “Receiver Initiate for Data” button located at “C” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **OFDM_Receiver_Init_Data_Receiver** function located in the **OFDM_receiver** library. Initiating the OFDM receiver for the transmission of a test image is done by pressing the “Receiver Initiate for Image” button also located at “C” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **OFDM_Receiver_Init_Image_Receiver** function also located in the **OFDM_receiver** library. The flowchart in Figure 7.14 explains the processes; functions and actions involved in initiating the OFDM receiver for both test data packets and test image transmission.

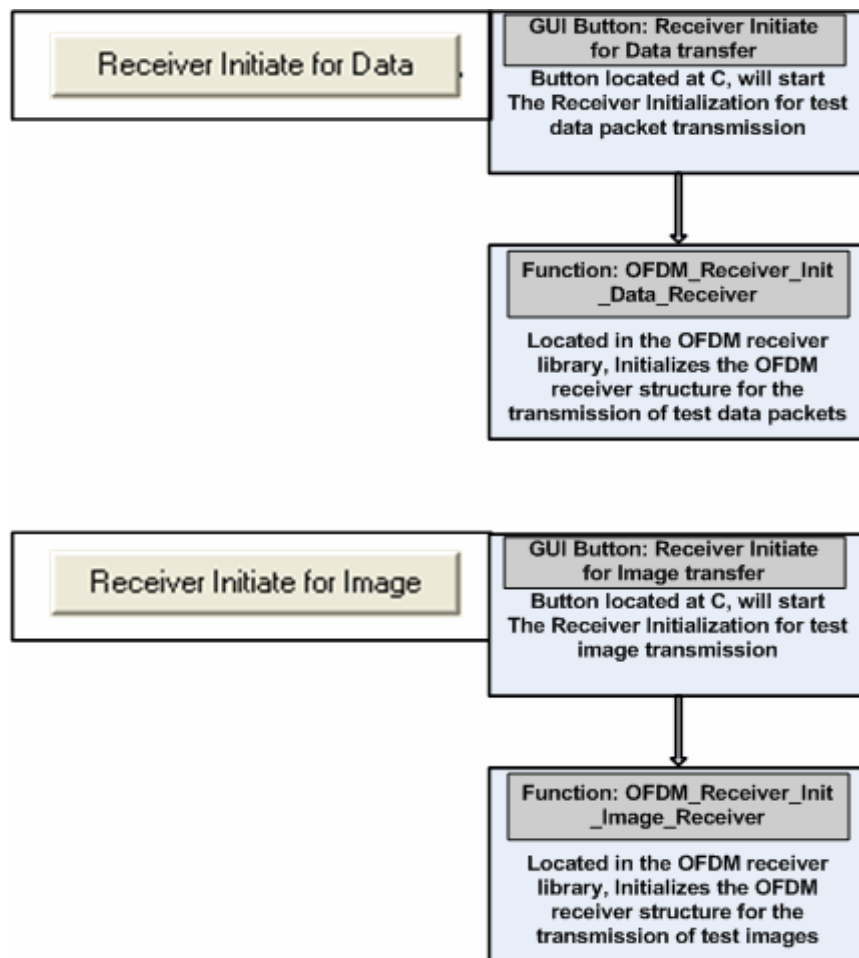


Figure 7.14: The OFDM receiver initiation flowchart

7.5.2.6 Decoding IEEE 802.11a Signals

Decoding IEEE 802.11a signals from file is done by pressing the “Decode from File” button located at “C” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **OFDM_Receiver_Start_Decode_From_File** function located in the **OFDM_receiver** library. Decoding IEEE 802.11a signals from the audio device is done by pressing the “Decode from Audio” button located at “C” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **Audio_Record_Transmission** function located in the **OFDM_audio** library. The flowchart in Figure 7.15 to 7.21 explains the processes; functions and actions involved in decoding IEEE 802.11a signals from local files and from the audio device.

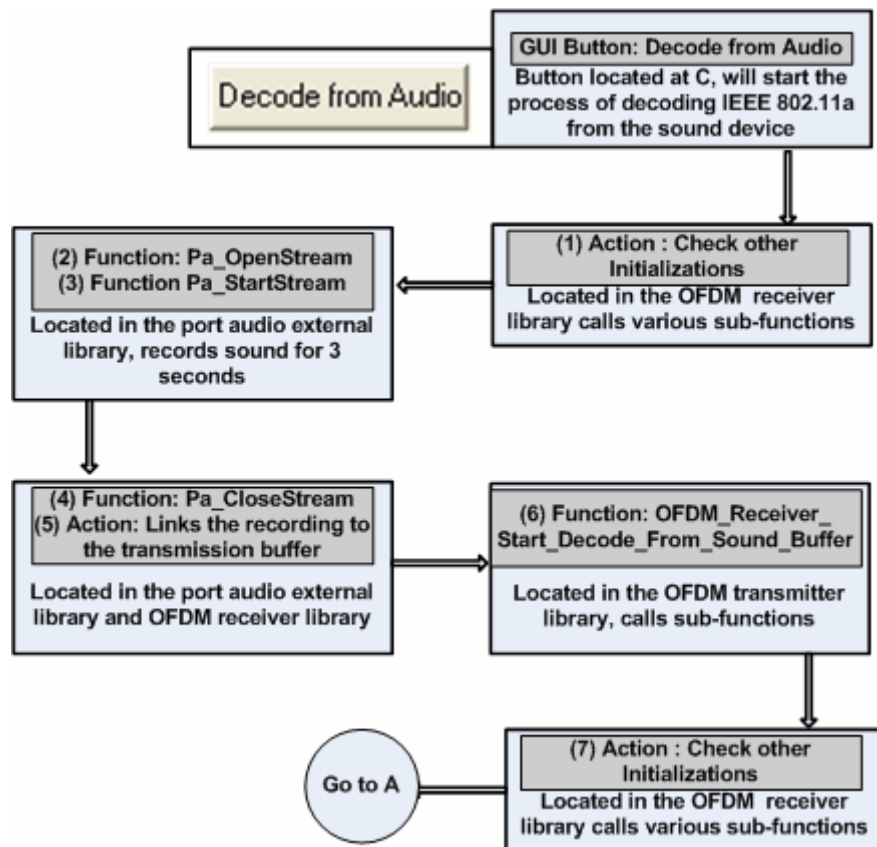


Figure 7.15: Decoding IEEE 802.11a signals from the audio device flowchart

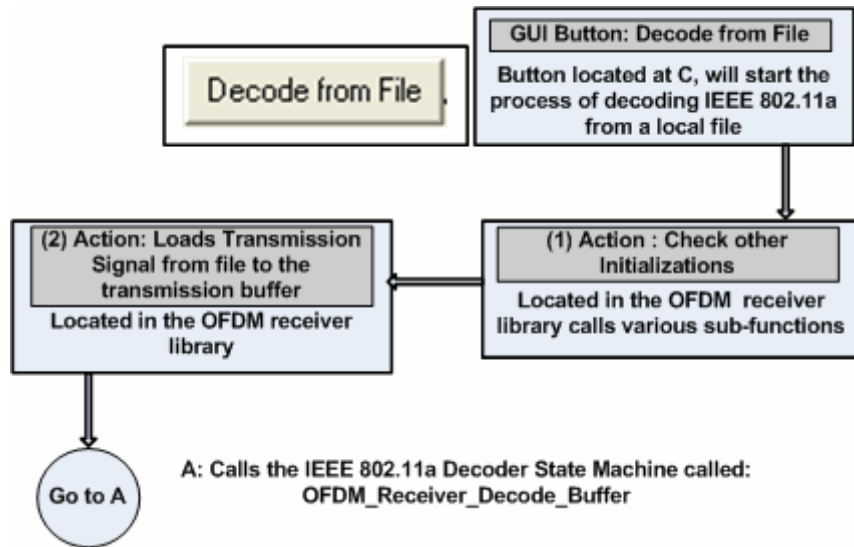


Figure 7.16: Decoding IEEE 802.11a signals from a local file flowchart

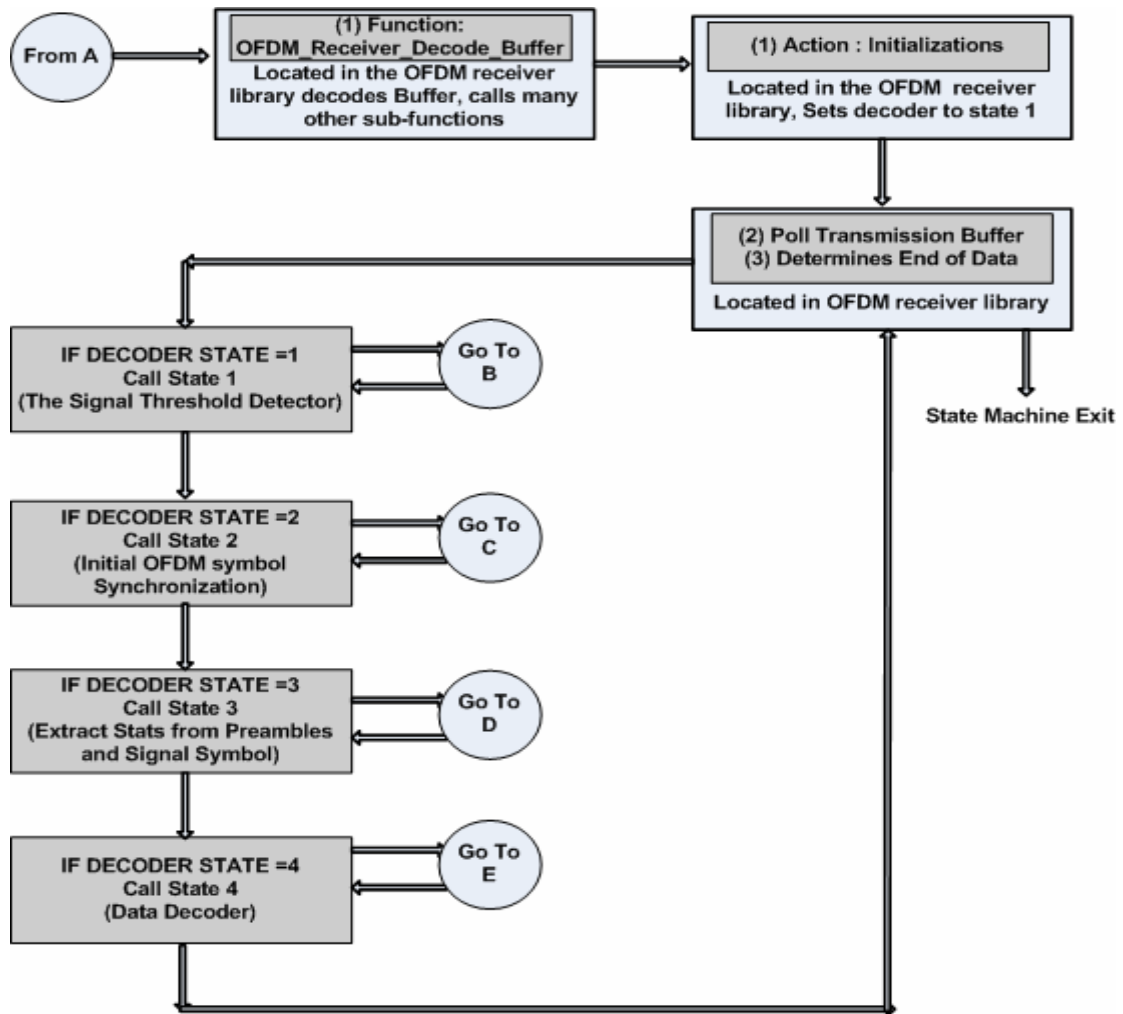


Figure 7.17: IEEE 802.11a decoder state machine flowchart

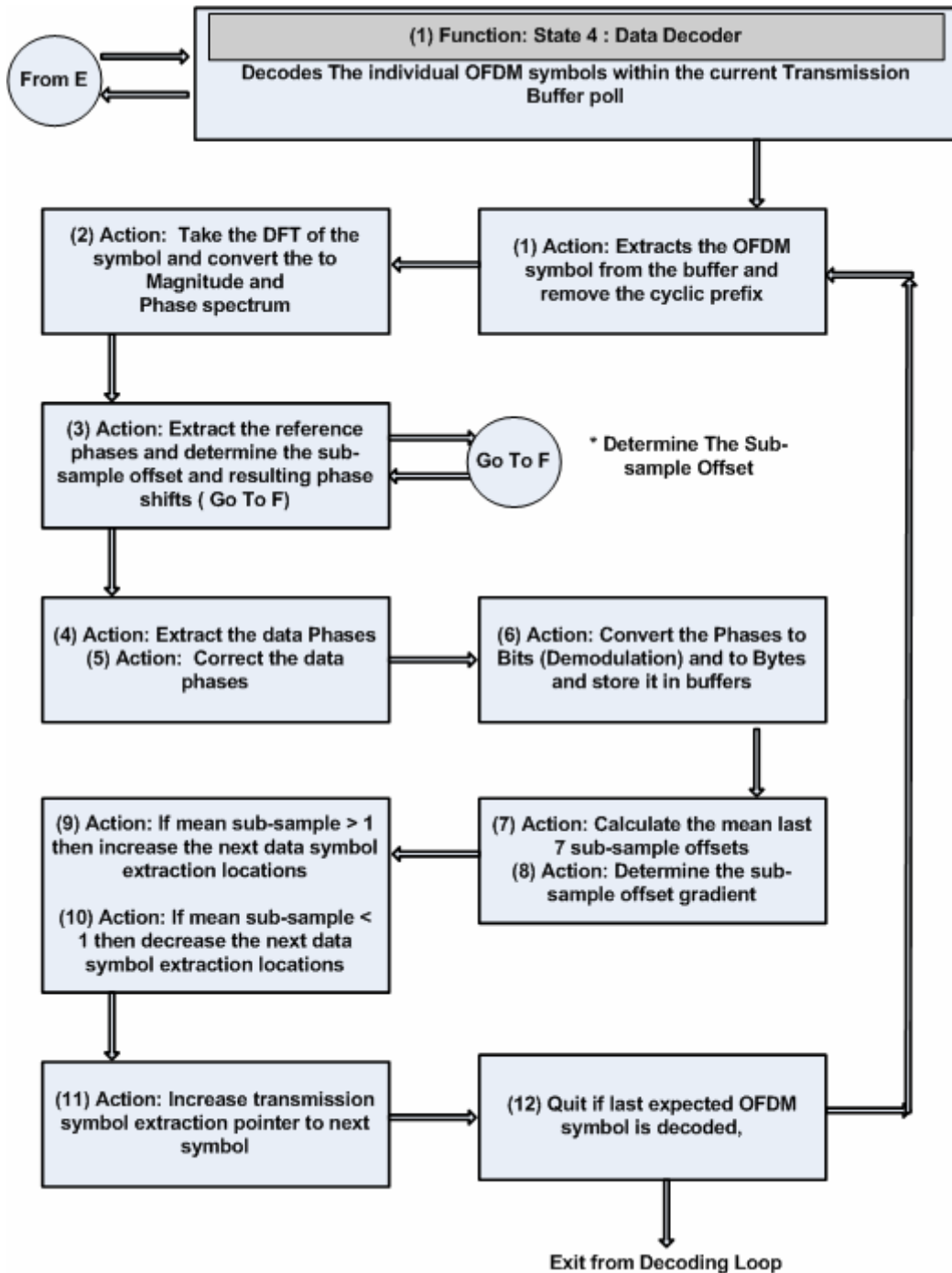


Figure 7.18: IEEE 802.11a decoder state 4, data decoder flowchart

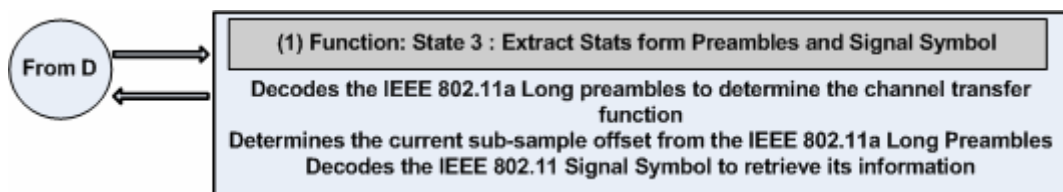


Figure 7.19: IEEE 802.11a decoder state 3, statistics extraction flowchart

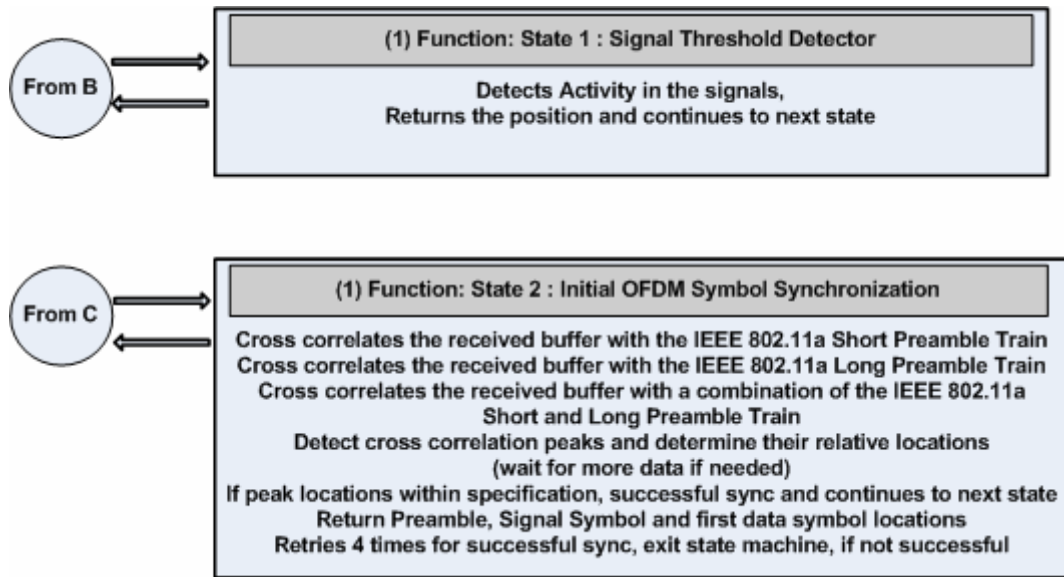


Figure 7.20: IEEE 802.11a decoder state 1 and 2

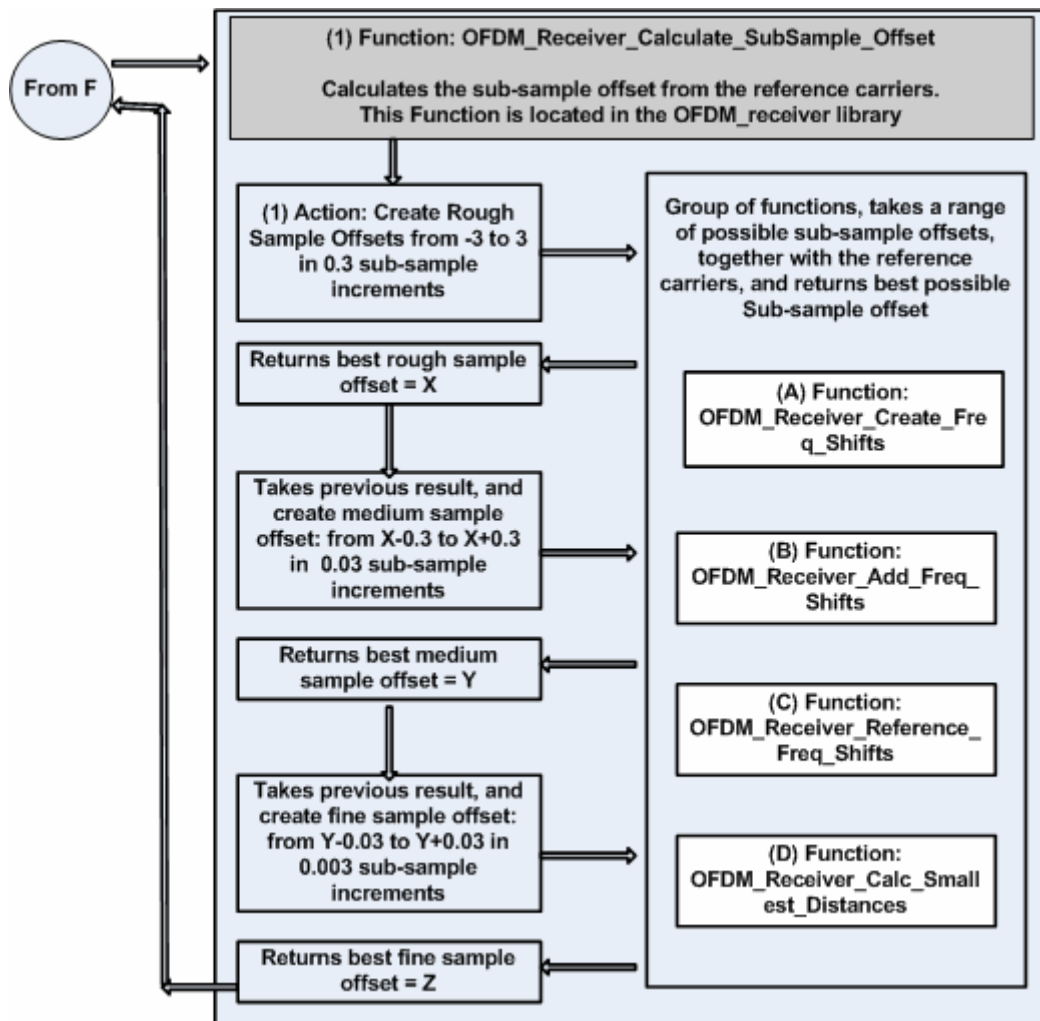


Figure 7.21: IEEE 802.11a decoder sub-sample offset estimator flowchart

7.5.2.7 Loading a Compare File

Loading a compare file is done by pressing the button “Load Compare File” located at “C” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **OFDM_Comparer_LoadCompareFile** function located in the **OFDM_comparer** library. The flowchart in Figure 7.22 explains the processes; functions and actions involved in loading a compare file.

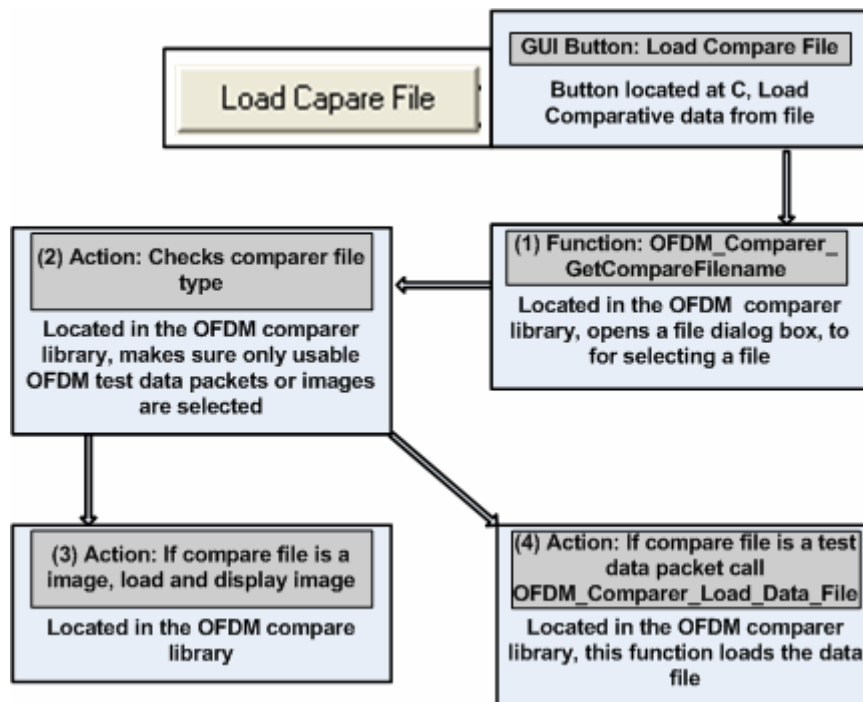


Figure 7.22: Loading a compare file flowchart

7.5.2.8 Comparing Received Data

Comparing the received data to comparative data and determining the performance statistics is done by pressing the button “Compare” located at “C” on the OFDM encoding/decoding Demo GUI in Figure 7.3. This action will call the **OFDM_Comparer_Compare** function located in the **OFDM_comparer** library. The flowchart in Figure 7.23 explains the processes; functions and actions involved in comparing data.

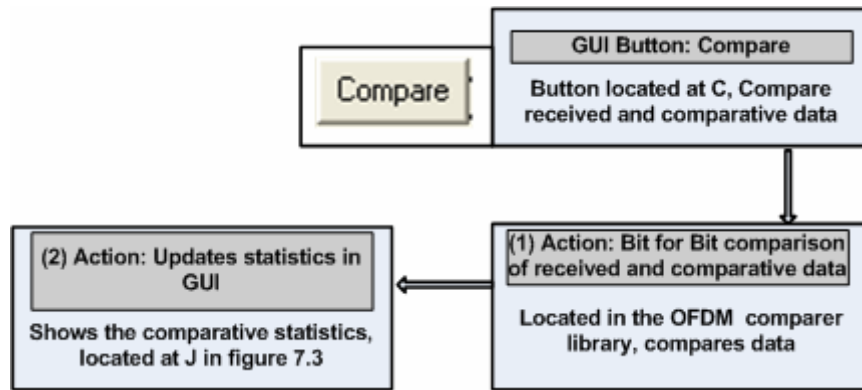


Figure 7.23: Comparing received data flowchart

7.5.2.9 Saving The Transmission to File

Saving the received transmission to file is done by pressing the button “Save To File” located at “C” on the OFDM encoding/decoding Demo GUI in Figure 7.3. The action will call the **OFDM_Receiver_Save_Transmission_To_Custom_File** function located in the **OFDM_Receiver** library. The flowchart in Figure 7.24 explains the processes; functions and actions involved in comparing data.

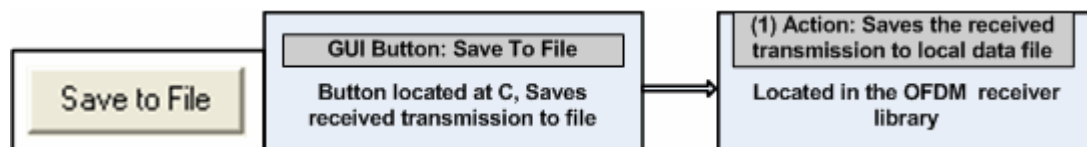


Figure 7.24: Saving the received transmission to file flowchart

7.5.2.10 Conclusions On The Receiving/Decoding Process

As mentioned earlier, the receiving and decoding of IEEE 802.11a OFDM data as explained in Sections 7.5.2.2 through 7.5.2.9 attempts to logically explain the processes involved, but should only be used as a guide. The real software libraries, structures and functions are in actual fact much more complicated. It is recommended that the information about the graphic user interface (GUI), the software structures, the software libraries, the explained processes involved, together with appendices C and D be used as a guide when working with the raw C++ source code.

7.5.3 Conclusions On The IEEE 802.11a Transceiver System

Sections 7.5.1 and 7.5.2 discussed the IEEE 802.11a transmitter and the IEEE 802.11a receiver processes in detail. The implemented IEEE 802.11a transceiver system can now be tested to determine how well it works.

7.6 Conclusions

This chapter introduced the SDR implementation of the IEEE 802.11a based transceiver system. Two personal computers were used as the hardware platforms. Their sound devices (sound cards) were connected together using stereo copper wires which acted as the communications channel. The IEEE 802.11a OFDM transceiver software is written in C++ and basically consists of a demonstration GUI and 6 software libraries. The demonstration program has the ability of encoding data into IEEE 802.11a OFDM packets and transferring it over the communications channel using the computer sound card. On the receiving computer, the demonstration program can receive the incoming data using the soundcard and decode the IEEE 802.11a OFDM packets and retrieve the original data. All the software structures, libraries and processes have been explained in this chapter, and should be used as a guide when working with the original source code. In the next chapter the performance of the implemented IEEE 802.11a transceiver system will be tested, to determine how well it works. The receiver system will be put through a series of simulation tests as well as real-time tests and influenced with different performance influencing factors to validate its performance.

Chapter 8

OFDM System Performance Tests and Results

8.1 Introduction

To determine how well the IEEE 802.11a based OFDM transceiver system works, it is necessary to test it. By testing different parts of the system and using different types of test data, it is possible to get a clear picture of how the system will perform in real-life situations. It also helps to identify parts of the system that needs to be improved.

8.2 Data Performance Tests and Results

The purpose of the IEEE 802.11a based OFDM transceiver system is to take digital data from a source, manipulate it using the rules set out in the IEEE 802.11a standards and finally deliver the digital data to a destination device. The most important test of such a system would thus be a test to determine the quality of the received digital data. The whole purpose of the IEEE 802.11a standard is to create a system that will maintain the best possible data quality (also known as the quality of service, since the service is the delivery of digital data) in the environment it is destined to be used in. Data performance tests test the quality of the received digital data. The result of these tests is the BER, which represents the expected amount of errors per unit of data in the receiver data. Data performance tests are done on simulation data first, to predict the data quality for different expected real-life factors, but in a controlled environment. The tests are then expanded to include controlled and measured transmissions over the communications channel, to gather statistics about the channel and sound devices and finally real-time communications tests. The three most important data performance tests that will be conducted on the IEEE 802.11a OFDM packets are shown in Table 8.1.

Nr	Test	Reason
1	General encoding and decoding tests without any performance influencing factors.	1. Test whether the basic encoding and decoding processes work. 2. Test the initial OFDM symbol synchronisation ability of the system.
2	AWGN data performance tests.	1. Test the receiver in the presence of noise. 2. Comparing the BER at different SNR against predicted values, will determine how effective the sub-carrier modulation works.
3	Sampling frequency drift tests.	1. This will test the continuous re-synchronisation and phase compensation ability of the receiver. 2. Test the influence of the re-synchronisation algorithms on normal data.

Table 8.1: The three most important data performance tests.

8.2.1 The Test Data Set

As mentioned in Chapter 7, The OFDM GUI program loads graphical images as test data for the IEEE 802.11a OFDM packets. It is easier to see the effects of real life performance influencing factors on the images than on random data. Five images were used in the data performance tests:



Figure 8.1: Test Image 1

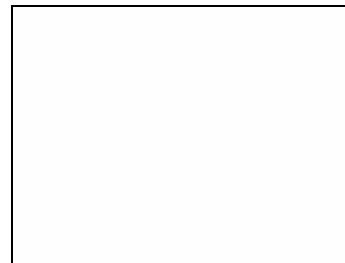


Figure 8.2: Test Image 2 (just white)

Parameter	Value
Test type and number:	Simulation Test 01
Test data images:	Test Image1, Test Image2, Test Image 3, Test Image 4, Test Image 5
Test data set:	10 encodes of each test data image. A total of 50 tests.
Total amount of digital bits transferred:	50 x 393216 = 19 660 800 bits = 18.75 Megabits
Reference carrier phase compensation:	Disabled
Data influenced by AWGN?	No
Data influenced by sampling frequency drift?	No

Table 8.2: Simulation Test 1 parameters

The results for this test are shown in Table 8.3.

	Amount of times encoded/ decoded	Amount of bits encoded/decoded	Decoding Errors	BER
Test Image 1	10	3932160	0	0
Test Image 2	10	3932160	0	0
Test Image 3	10	3932160	0	0
Test Image 4	10	3932160	0	0
Test Image 5	10	3932160	0	0
Total	50	18,75 Megabits	0	0

Table 8.3: Simulation Test 1 results

The results for this test show that all the encoded data were decoded without any errors. This means that the initial OFDM symbol synchronisation routines, as well as the basic IEEE 802.11a OFDM data decoder works correctly when there is no interference.

8.2.2.2 Simulation Test 2

In this test, the five test images were encoded into IEEE 802.11a OFDM test data packets. The encoded signals were subjected to different SNR of AWGN. To compare the BER of the OFDM encoded signal to the theoretically predicted QPSK results calculated in Chapter 2, the AWGN added to the OFDM signals is referenced to each individual OFDM sub-carrier and not to the OFDM signal as a whole. The relationship can be calculated as [7] (2.14)

$$\text{SNR}_{\text{dB_QPSK}} = 10 \log_{10} \left(\frac{\text{QPSK Signal Power}}{\text{Noise Power}} \right). \quad (8.1)$$

Since the IEEE 802.11a OFDM signal has $N_{\text{ST}} = 52$ QPSK sub-carriers, the signal has 52 times more power,

$$\text{SNR}_{\text{dB_OFDM}} = 10 \log_{10} \left(\frac{52 \times \text{QPSK Signal Power}}{\text{Noise Power}} \right), \quad (8.2)$$

which becomes

$$\text{SNR}_{\text{dB_OFDM}} = 10 \log_{10} \left(\frac{\text{QPSK Signal Power}}{\text{Noise Power}} \right) + 10 \log_{10} (52) \quad (8.3)$$

and can finally be written as

$$\text{SNR}_{\text{dB_OFDM}} = \text{SNR}_{\text{dB_QPSK}} + 17.16\text{dB}. \quad (8.4)$$

In this simulation test, the 17.16dB difference between the OFDM and single QPSK signals is ignored to see how the BER vs. SNR graphs look relative to each other. Reference carrier phase compensation was disabled as part of a combined test to determine its influence on the receiving/decoding process.

The expected results of this test should be values very close to values in Figure 2.6 and Table 2.1, which is the theoretically predicted BER for a QPSK modulated signal. The parameters for this test are shown in Table 8.4.

Parameter	Value
Test type and number:	Simulation Test 02
Test data images:	Test Image1, Test Image2, Test Image 3, Test Image 4, Test Image 5
Test data set:	5 encodes of each test data image at 10 SNR levels. A total of 250 tests.
Total amount of digital bits transmitted:	$250 \times 393216 = 98\,304\,000$ bits = 93.75 Megabits
Reference carrier phase compensation:	Disabled
Data influenced by AWGN?	Yes, 0dB to 10dB, in 1dB increments
Data influenced by sampling frequency drift?	No

Table 8.4: Simulation Test 2 parameters

The results for this test are shown in the Figure 8.6.

The BER of the simulated OFDM signal is the same as the BER of the theoretically predicted QPSK modulated signal as shown by Equation (2.45), Figure 2.6 and Table 2.1. These results further confirm the orthogonality of OFDM encoded signals and means that the receiver and decoder can correctly decode IEEE 802.11a OFDM packets in the presents of AWGN.

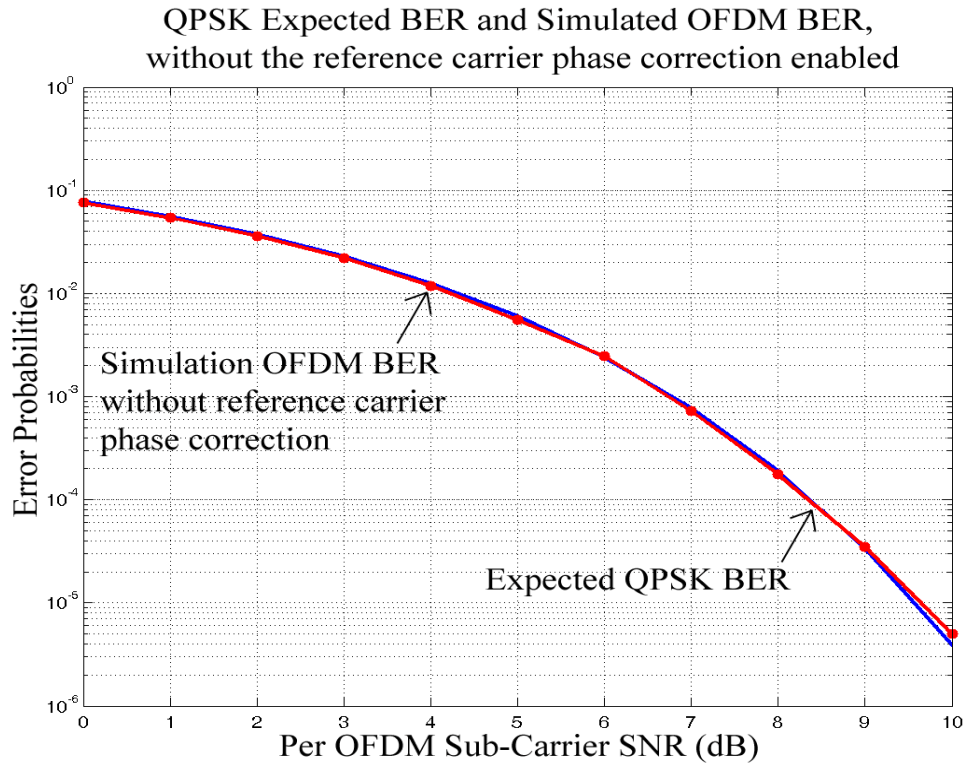


Figure 8.6: The results of Simulation Test 2

8.2.2.3 Simulation Test 3

In this test, the five test images were encoded into IEEE 802.11a OFDM test data packets. The encoded signals were subjected to four sampling frequency drifts of –203 PPM, –102 PPM, 102 PPM and 203 PPM. With this test we can determine whether sampling frequency drift and the resulting sub-sample offset errors really have such a big influence on the receiver/decoder, as thought. Reference carrier phase compensation was disabled, which means the system will have no way of combating sampling frequency drifts and the resulting sub-sample shifts they introduce. The expected results of this test is a very poor performance, due to the de-synchronisation effects of the sampling frequency drift, the decoder should become de-synchronised very early in the decoding process and decode the bulk of the data incorrectly. The parameters for this test are shown in Table 8.5.

Parameter	Value
Test type and number:	Simulation Test 03
Test data images:	Test Image1, Test Image2, Test Image 3, Test Image 4, Test Image 5
Test data set:	5 encodes of each Test Data Image at the 4 different sample frequency drifts. A total of 100 tests.
Total amount of digital bits transferred:	$100 \times 393216 = 39\,321\,600$ bits = 37.5 Megabits
Reference carrier phase compensation:	Disabled
Data influenced by AWGN?	No
Data influenced by sampling frequency drift?	Yes, -203 PPM, -102 PPM, 102 PPM and 203 PPM

Table 8.5: Simulation Test 3 parameters

The results for this test are shown in Table 8.6.

	Amount of files encoded/ decoded	BER at -203 PPM	BER at -102 PPM	BER at 102 PPM	BER at 203 PPM
Test Image 1	5	0.49	0.483	0.485	0.49
Test Image 2	5	0.49	0.484	0.485	0.49
Test Image 3	5	0.49	0.484	0.485	0.49
Test Image 4	5	0.49	0.483	0.485	0.49
Test Image 5	5	0.49	0.480	0.484	0.49
Average	5	0.49	0.483	0.485	0.49

Table 8.6: Simulation Test 3 results

The results from this test clearly show that the system performed shockingly poor. The average BER for all the tests was 0.487. This value is very close to 0.5, which means the system was really just decoding random data. The decoder had an equal chance of decoding a bit as a 1 or a 0. It thus becomes clear that reference carrier phase compensation is very important in OFDM decoding in to keep the receiver synchronised to the OFDM symbols.

The next test would have been a combined AWGN and sampling frequency drift test, where the test images are subjected to different levels of AWGN together with different sampling frequency drifts, with the reference carrier phase compensation algorithm still disabled.

However, due to the fact that the current test scores had a BER of approximately 0.5, which is the worst possible score, corrupting the signal further with AWGN will have no real effect of the results.

The combined AWGN and sampling frequency drift test, without reference carrier phase compensation will thus not be preformed.

8.2.2.4 Simulation Test 4

In this test, the five test images were encoded into IEEE 802.11a OFDM test data. The test data was NOT altered in any way, and it was given to the receiver to decode. Reference carrier phase compensation was enabled as part of a combined test to determine its influence on the receiving/decoding process. The expected results of these tests are errorless decoded data. The parameters for this test are shown in Table 8.7.

Parameter	Value
Test type and number:	Simulation Test 04
Test data images:	Test Image1, Test Image2, Test Image 3, Test Image 4, Test Image 5
Test data set:	10 encodes of each Test Data Image. Total of 50 tests
Total amount of digital bits transferred:	$50 \times 393216 = 19\,660\,800$ bits = 18.75 Megabits
Reference carrier phase compensation	Enabled
Data influenced by AWGN?	No
Data influenced by sampling frequency drift?	No

Table 8.7: Simulation Test 4 parameters

The results for this test are shown in Table 8.8

	Amount of times encoded/ decoded	Amount of bits encoded/decoded	Decoding Errors	BER
Test Image 1	10	3932160	0	0
Test Image 2	10	3932160	0	0
Test Image 3	10	3932160	0	0
Test Image 4	10	3932160	0	0
Test Image 5	10	3932160	0	0
Total	50	18,75 Megabits	0	0

Table 8.8: Simulation Test 4 results

The results of this test show that all the encoded data were decoded without any error. This means that the reference carrier phase compensation algorithm does work, when there is no AWGN or sampling frequency drift.

8.2.2.5 Simulation Test 5

In this test, the five test images were encoded into IEEE 802.11a OFDM test data packets. The encoded signals were subjected to different SNR of AWGN. In this test the reference carrier phase compensation was enabled. By comparing these results to the results from Simulation Test 2, it should be possible to determine whether AWGN has any influence on the reference carrier phase compensation algorithm. The expected results of this test should be values very close to that of Simulation Test 2. The parameters for this test are shown in Table 8.9.

Parameter	Value
Test type and number:	Simulation Test 05
Test data images:	Test Image1, Test Image2, Test Image 3, Test Image 4, Test Image 5
Test data set:	5 encodes of each Test Data Image at 10 different SNR levels. Total of 250 tests.
Total amount of digital bits transferred:	$250 \times 393216 = 98\,304\,000$ bits = 93.75 Megabits
Reference carrier phase compensation:	Enabled
Data influenced by AWGN?	Yes, from 0dB to 10dB, in 1dB increments
Data influenced by sampling frequency drift?	No

Table 8.9: Simulation Test 5 parameters.

The results for this test are shown in Figure 8.7.

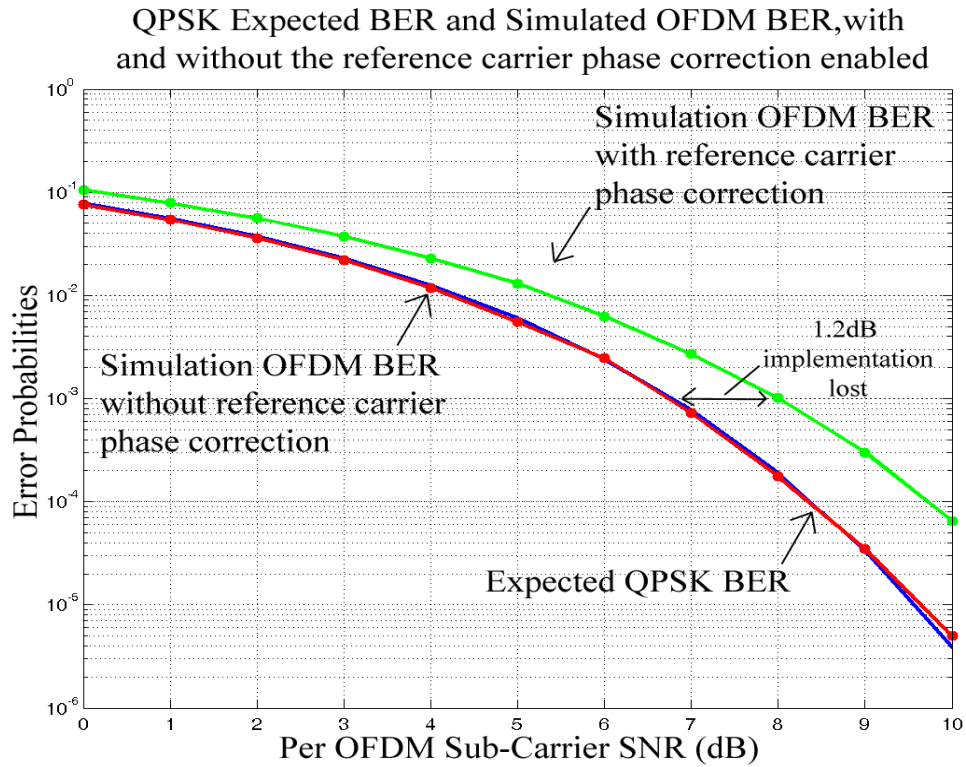


Figure 8.7: BER vs. SNR of the OFDM signal (with and without the reference carrier phase correction enabled) and a QPSK modulated signal.

From these results we can clearly see a difference between the AWGN tests where the reference carrier phase compensation algorithm where enabled and disabled. The average difference is approximately 1.2 dB. The reason for this is that AWGN influences all the OFDM sub-carriers, including the reference carriers. There is no induced sampling frequency drift in this test, but the noise on the reference carriers do influence the reference carriers' phases. This fools the algorithm in thinking that there is some small sampling frequency drift. The reference carrier phase compensation algorithm then attempts to correct these apparent drifts and by doing so causes more decoding errors. This is referred to as the reference carrier phase compensation implementation loss due to AWGN. Better algorithm programming could improve this, but probably never eliminate it completely.

8.2.2.6 Simulation Test 6

In this test, the five test images were encoded into IEEE 802.11a OFDM test data packets. The encoded signals were subjected to four sampling frequency drifts of –203 PPM, -102 PPM, 102 PPM and 203 PPM. With the reference carrier phase compensation ability enabled, we can determine how well the algorithm compensates sampling frequency drift, when no noise is present. The expected result of this test is an errorless performance. The reference carrier phase compensation algorithm should be able to effortlessly correct the sampling frequency drift and compensate for the resulting phase errors. The parameters for this test are shown in Table 8.10.

Parameter	Value
Test type and number:	Simulation Test 06
Test data images:	Test Image1, Test Image2, Test Image 3, Test Image 4, Test Image 5
Test data set:	5 encodes of each Test Data Image at 4 different sample frequency drifts. A total of 20 tests.
Total amount of digital bits transferred:	$20 \times 393216 = 7\,864\,320$ bits = 7.5 Megabits
Reference carrier phase compensation:	Enabled
Data influenced by AWGN?	No.
Data influenced by sampling frequency drift?	Yes, -203 PPM, -102 PPM, 102 PPM and 203 PPM

Table 8.10: Simulation Test 6 parameters.

The results for this test are shown in Table 8.11.

	Amount of files encoded/decoded	BER at -203 PPM	BER at -102 PPM	BER at 102 PPM	BER at 203 PPM
Test Image 1	5	0	0	0	0
Test Image 2	5	0	0	0	0
Test Image 3	5	0	0	0	0
Test Image 4	5	0	0	0	0
Test Image 5	5	0	0	0	0
Average	5	0	0	0	0

Table 8.11: Simulation Test 6 results.

The results from this test show that the reference carrier phase compensation algorithm was able to correctly re-synchronise and compensate the induced phase errors in the OFDM signals subjected to sampling frequency drift.

8.2.2.7 Simulation Test 7

In this test, the five test images were encoded into IEEE 802.11a OFDM test data packets. The encoded signals were subjected to different SNR of AWGN as well as four sampling frequency drifts of -203 PPM, -102 PPM, 102 PPM and 203 PPM. The combined AWGN and sampling frequency drift tests, with the reference carrier phase compensation enabled, simulates the most important performance influencing factors, and should give results which closely correlates with the results from the real-time implementation tests later. The parameters for this test are shown in Table 8.12.

Parameter	Value
Test type and number:	Simulation Test 07
Test data images:	Test Image1, 2, 3, 4 and 5
Test data set:	2 encodes of each Test data Image at 10 different SNR levels, and 4 different sample frequency drifts. Total of 400 tests.

<i>Parameters for simulation test 7 continues...</i>	
Parameter	Value
Total amount of digital bits transferred:	400 x 393216 = 157 286 400 bits = 150 Megabits
Reference carrier phase compensation:	Enabled
Data influenced by AWGN?	Yes, from 0dB to 10dB, in 1dB increments
Data influenced by Sample frequency drift?	Yes, -203PPM, -102PPM, 102PPM and 203PPM

Table 8.12: Simulation Test 7 parameters.

The results for this test are displayed in Figure 8.8.

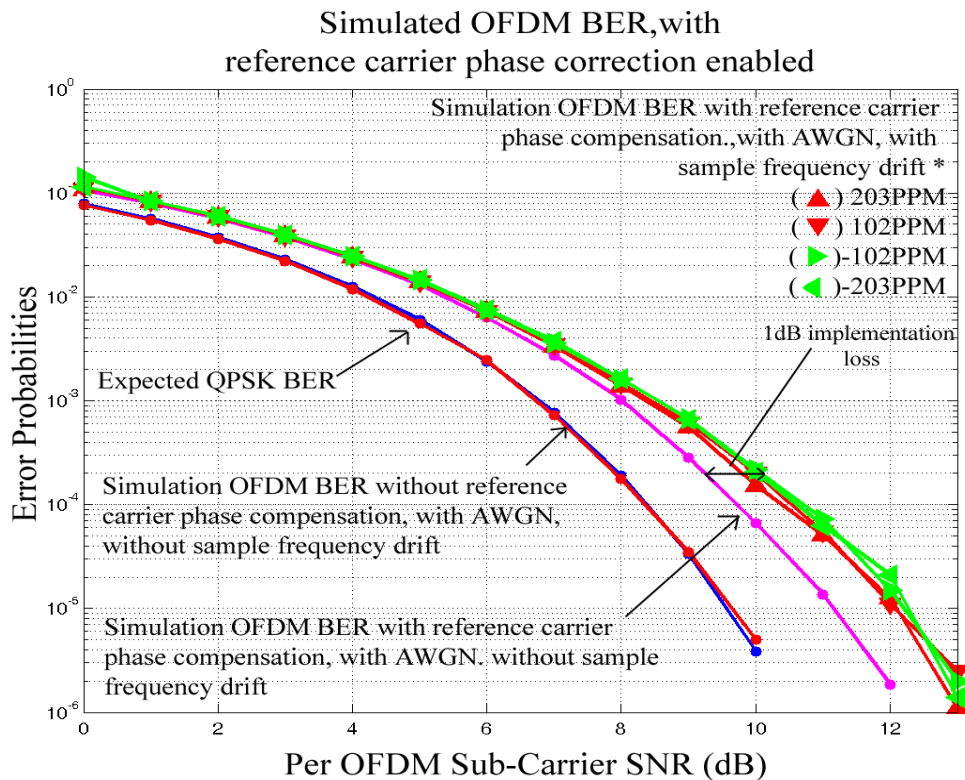


Figure 8.8: Simulation BER vs. SNR of the OFDM signal with reference carrier phase compensation, with AWGN, and sampling frequency drift.

The results from this test follow the results from the previous test, but shows an additional implementation loss of approximately 1dB at a SNR of 10dB (purely due to the sampling frequency drift).

8.2.2.8 Simulation Test Conclusions

From the seven simulation tests we can conclude that

- The OFDM receivers' initial OFDM symbol synchronisation algorithm works.
- The OFDM receiver decodes data successfully when there are no influencing factors involved.
- The OFDM receiver sub-carrier demodulation algorithms correctly decode OFDM sub-carriers in the presence of AWGN.
- Reference carrier phase compensation is clearly needed when OFDM data packets are subjected to sampling frequency drifts.
- The reference carrier phase compensation works.
- The reference carrier phase compensation algorithm has a 1.2dB implementation loss due to AWGN only.
- The reference carrier phase compensation algorithm has a 1 dB implementation loss due to sampling frequency drift.

8.2.3 Initial Sound Device Transmission Tests

Initial sound device transmission tests specifically test the sound device and the communications channel. These tests encode test data into IEEE 802.11a OFDM packets and then transmit it over the communications channel. The receiving system receives the incoming transmission and then records it a local file. This local file, representing the IEEE 802.11a OFDM packet, is then decoded using the same procedures and algorithms as the previous simulation tests. The extensive simulation test results, and the already tested OFDM decoding algorithms, should give us a fairly good idea of how the system will react to these transmission tests as well as the real-time tests later.

8.2.3.1 Communications Channel Estimation

As described in Section 5.2.5, channel estimation is done to counteract the effects of an uneven channel transfer function. To determine the communication channel and sound device transfer function, we need to create a signal with a flat and infinite bandwidth. The impulse signal does this, as shown in the following Fourier Transform pair

$$\delta(n) \xrightarrow{\text{DFT}} 1. \quad (8.5)$$

The impulse was transmitted over the communications channel and recorded at the receiver. The impulse response is shown in the Figure 8.9.

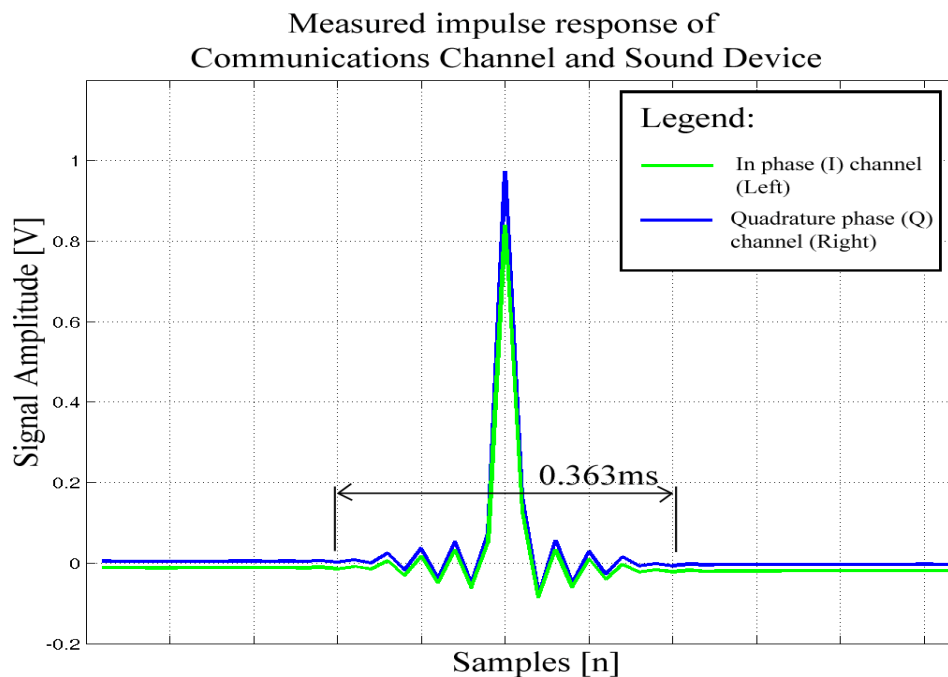


Figure 8.9: Impulse response of the communications channel and sound device

The communication channel and sound device transfer function was calculated by taking the DFT of the IEEE 802.11a long preamble. As we can see from Figure 8.10, the IEEE 802.11a receiver will indeed need to compensate for the effects of the communication channel transfer function.

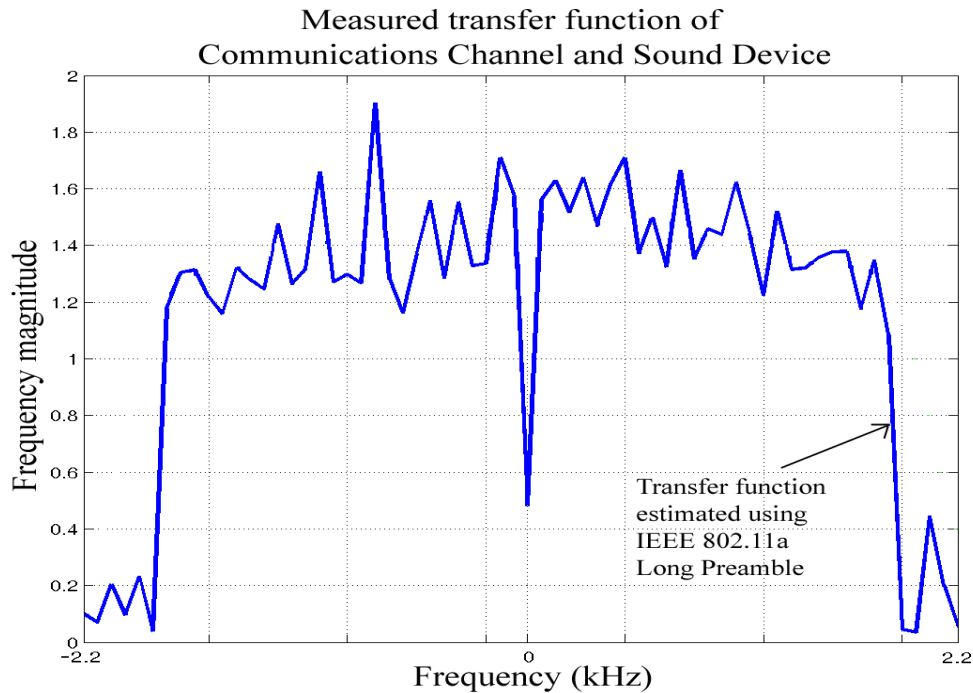


Figure 8.10: Transfer function of the communications channel and sound device

8.2.3.2 Determining Sampling Frequency Drift.

The reference carrier phase compensation algorithm can calculate the sound devices' sampling frequency drift during the OFDM data symbol decoding process. The algorithm keeps a record of all the decoded OFDM data symbols' calculated sub-sample offset. By determining the rate of change of the sub-sample offset during the decoding process, it is possible to calculate the sampling frequency drift.

The gradient is calculated from Figure 8.11, as

$$\text{Grad}(A, B) = \frac{(\text{Sub-sample offset at Point B}) - (\text{Sub-sample offset at Point A})}{(\text{Symbol number at point B}) - (\text{Symbol number at point A})},$$

$$\text{Grad}(A, B) = \frac{(1312) - (1224)}{(1.0377) - (0.1145)} = \frac{88}{0.9232} = 95.32$$

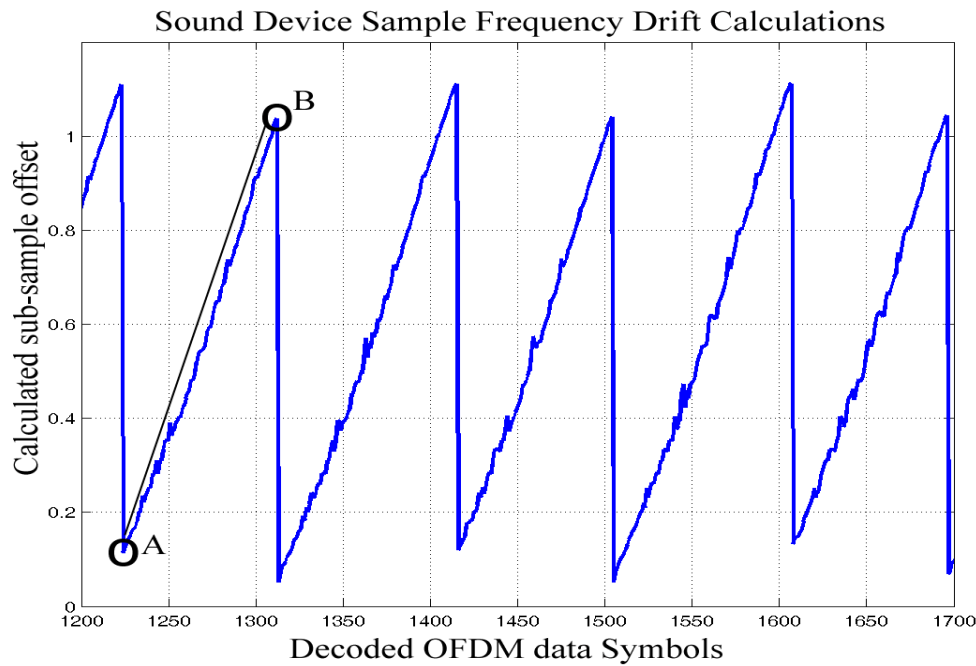


Figure 8.11: Calculated sub-sample offset and resulting sample frequency drift

This basically means that the decoder will advance 1 sample after every 95.32 OFDM data symbols is decoded. Since every OFDM symbol, together with its cyclic prefix / guard interval is exactly 80 samples in duration. The Gradient can be converted into a sampling frequency drift,

$$\text{Sample Frequency Drift} = \frac{1e6}{80 \times \text{Grad}(A, B)} = 131.14 \text{ PPM.} \quad (8.6)$$

The sample frequency drift is thus calculated as 131.14 PPM. This further confirms that sampling frequency drift is a real issue, especially with such relatively simple hardware as a personal computer sound device.

8.2.3.3 AWGN performance Test

In this test, the five test images were encoded into IEEE 802.11a OFDM test data packets. The encoded signals were subjected to different SNR of AWGN and transmitted over the communications channel. The receiving computer recorded the

transmission and gave it to the IEEE 802.11a OFDM decoder to decode. The reference carrier phase compensation was enabled. The parameters for this test are shown in Table 8.13.

Parameter	Value
Test type and number:	Initial Sound Device Transmission Test 1
Test data images:	Test Image1, Test Image2, Test Image 3, Test Image 4, Test Image 5
Test data set:	5 encodes of each Test Data Image at 10 different SNR levels. Total of 250 tests.
Total amount of digital bits transferred:	250 x 393216 = 98 304 000 bits = 93.75 Megabits
Reference carrier phase compensation:	Enabled
Data influenced by AWGN?	Yes, from 0dB to 10dB, in 1dB increments
Data influenced by sampling frequency drift?	No

Table 8.13: Initial Sound Device Transmission Test parameters.

The results for this test are shown in the Figure 8.12.

The results show that the initial sound device AWGN test results performed very close to the simulated sampling frequency drift results as determined in simulation test 7. This is a good sign, and shows that the simulations are in fact accurate representations of the real life implementation of the system. It should be noted that the OFDM decoder sometimes had trouble decoding some of the 0dB test images as the signal was too poor and the decoder could not successfully synchronise to the IEEE 802.11a OFDM packets.

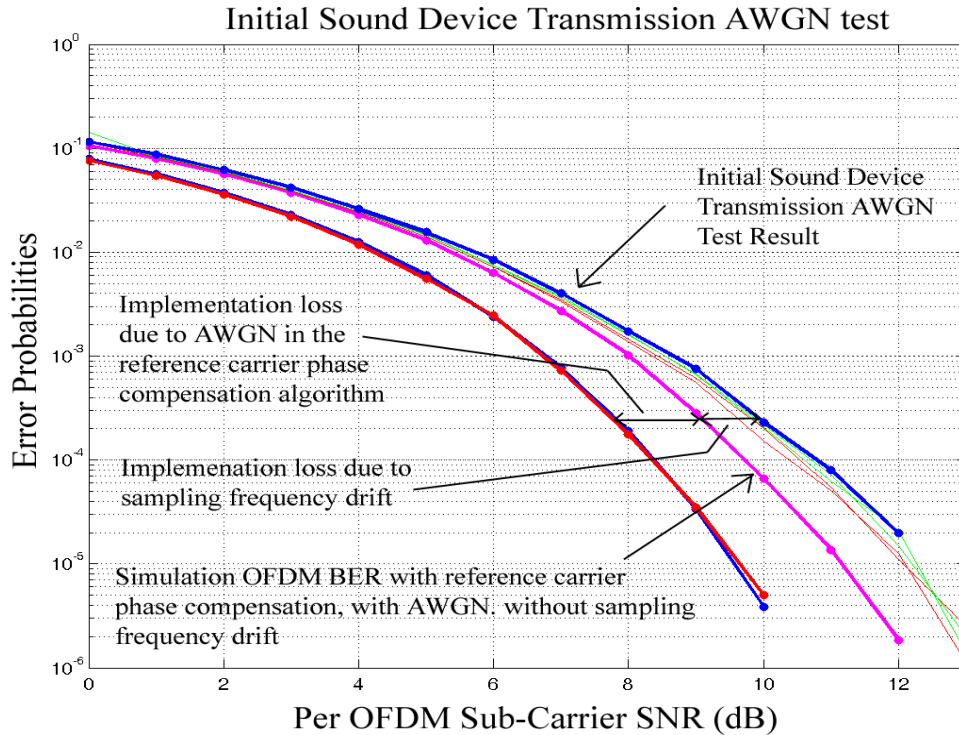


Figure 8.12: Initial Sound Device Transmission AWGN Test, BER vs. SNR.

8.2.3.4 Initial Sound Device Transmission Test Conclusions

From the initial sound device transmission tests, we can conclude that:

- Communication channel transfer function and impulse response is a real issue. Channel estimation is needed for modulation techniques that depend on signal amplitudes.
- Sampling frequency drift is also a real issue, even when using relatively simple and slow sampling devices like computer sound cards. This further emphasises the need for reference carrier phase compensation and continuous OFDM symbol re-synchronisation algorithms, even if they do introduce an implementation loss.
- The results from the initial sound device AWGN transmission tests are very close to the results from the final simulation test. This means that the simulations are in fact accurate representations of the real life implementation of the system.

8.2.4 Real-time Buffered Transmission Tests

Real-time buffered transmissions tests are the final data performance tests. The transmitter encodes test data into an IEEE 802.11a OFDM data packet and transmits it over the communications channel. The receiver stores the incoming signal in a receiving buffer, and then attempts to decode the IEEE 802.11a OFDM data packet in the buffer. The transmission signal is influenced by different levels of AWGN to determine how well the transceiver system works.

8.2.4.1 The Test Data Set

The real-time buffered transmission tests use different test data than the previous simulation and initial sound device transmission tests. The new test data packets contain 6144 bytes (49152 bits) of data. Using the current OFDM encoding parameters, these smaller data packets will occupy only 512 IEEE 802.11a OFDM data symbols instead of the 4096 that the test images occupy. This reduces the transmission time to approximately 1 second and helps speed up the performance tests. Please refer to Section 7.5.1.6 for more information. The three test data packets each have random bits of data.

8.2.4.2 AWGN Performance Tests

In this test the three test data packets were encoded into IEEE 802.11a OFDM test data packets. The encoded signals were subjected to different SNR of AWGN and transmitted over the communications channel. The receiving computer running the IEEE 802.11a encoding/decoding Demo GUI received the signals, buffered it and attempted to decode it. The reference carrier phase compensation algorithm was enabled. The parameters of this test are shown in Table 8.14.

Parameter	Value
Test type and number:	Real-time Buffered Transmission Test 1
Test data packets	Test Data Packet 1, 2 and 3
Test data set:	5 encodes of each Test Data packet at 10 different SNR levels. Total of 150 tests.
Total amount of digital bits transferred:	150 x 49152 = 7 372 800 bits = 7.035 Megabits
Reference carrier phase compensation	Enabled
Data influenced by AWGN?	Yes, from 0dB to 10dB, in 1dB increments
Data influenced by sampling frequency drift?	No

Table 8.14: Real-time Buffered Transmission Test parameters.

The results for this test are shown in Figure 8.13.

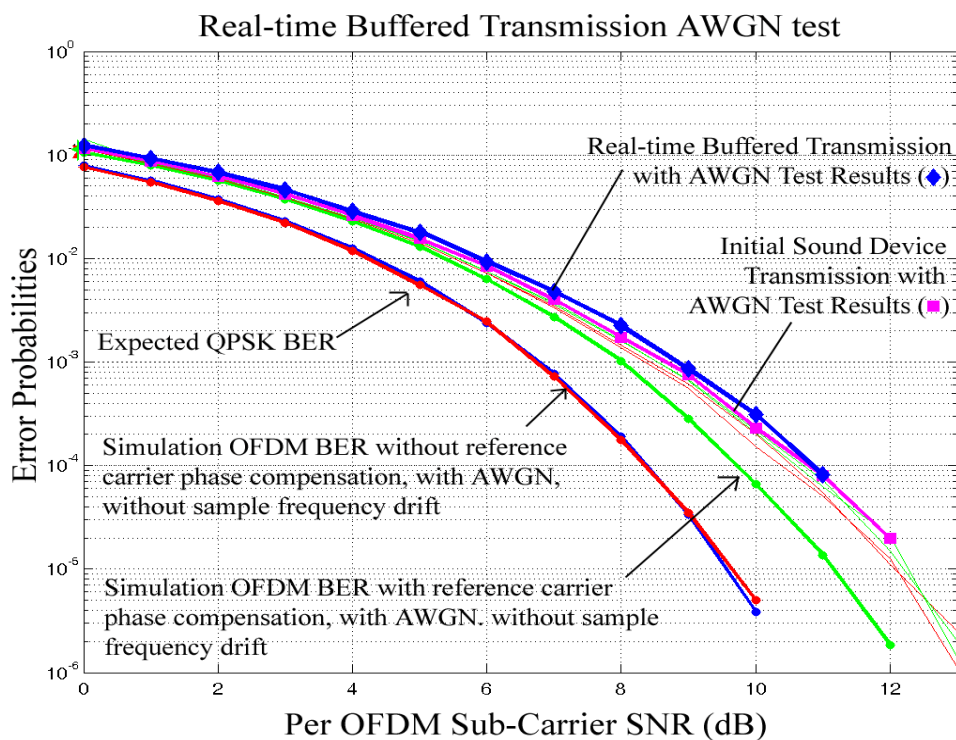


Figure 8.13: Real-time Buffered Transmission AWGN Test, BER vs. SNR.

The results show that the real-time buffered transmission AWGN test performed very close to the initial sound device transmission AWGN test as well as the simulated sampling frequency drift results as determined in simulation test 7. This means that the IEEE 802.11 transceiver works and that the sound card drivers interfaced successfully with the IEEE 802.11a decoding software.

8.2.4.3 Final IEEE 802.11a OFDM decoder performance Graph

In Section 8.2.2.2 we determined that there is in actual fact a 17.16dB difference between the AWGN BER graphs for QPSK modulated signals and OFDM modulated signals using QPSK as sub-carrier modulation. Finally it is possible to display the IEEE 802.11a OFDM performance graph when the signals are influenced by AWGN and referenced to the entire OFDM signal and not to a OFDM sub-carrier.

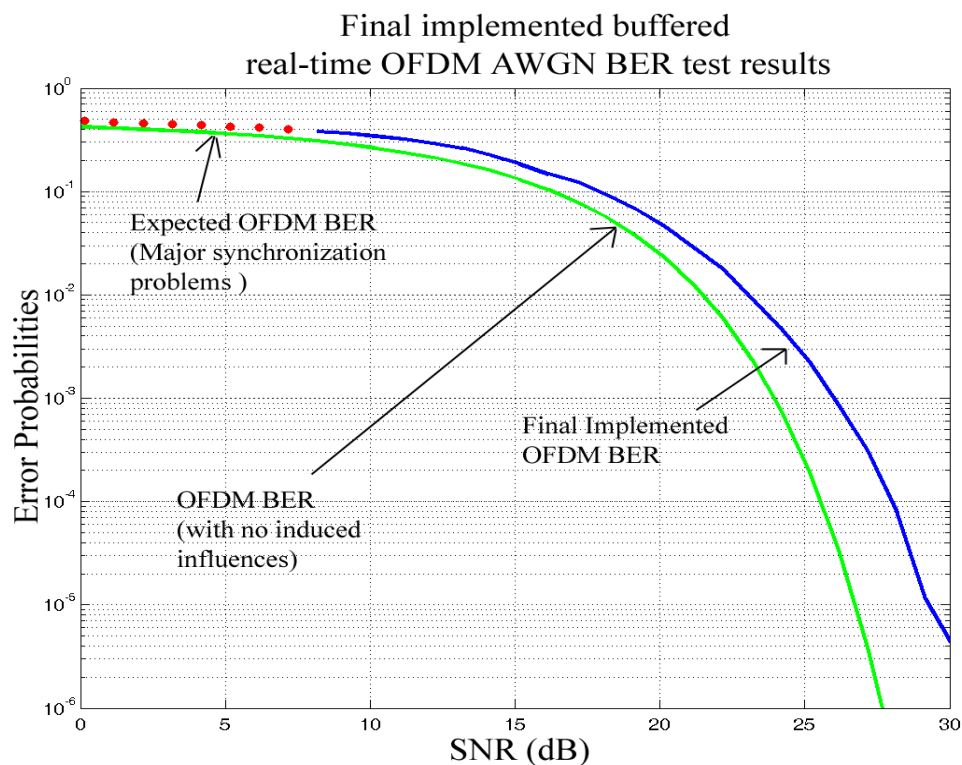


Figure 8.14: Final Real-time Buffered Transmission AWGN Test, BER vs. Complete OFDM Signal SNR.

8.2.4.4 Real-time Buffered Transmission Test Conclusions

From the real-time buffered transmission tests, we can conclude that:

- The real-time buffered transmission tests work.
- That the sound card interface to the IEEE 802.11a OFDM decoder works.
- The results from this test are very close to what we expected with the initial sound device transmission AWGN tests and the final simulations tests.

8.3 Speed Performance Tests and Results

Speed performance tests determine the IEEE 802.11a OFDM packet encoding and decoding times on the two different hardware platforms. These tests play a role in determining the different buffer lengths as well whether the current software will be able to encode and decode the full bandwidth 20 MHz IEEE 802.11a OFDM data packets.

8.3.1. Encoding Speed Tests

Encoding speed tests measure the time it takes the software to encode IEEE 802.11a data packets when it is given the encoding data. The test images as well as the test data packets were encoded and timed. These tests are performed on both the transmitter and receiver hardware platforms as described in Chapter 7. The parameters from this test are shown in Table 8.15.

Parameter	Value
Test type and number:	Speed Test 01 - Encoding
Test data packets:	Test Data Packet 1,2 and 3 Test Image 1,2 and 3
Test data set:	5 encodes of each test file. Total 30 tests.
Total amount of digital bits encoded:	$15 \times 49152 + 15 \times 393216$ bits = 6.328 Megabits

Table 8.15: Speed Test 1 parameters.

The results from the encoding speed test 1 is shown in Table 8.16.

The Encoding Times	Hardware Platform1	Hardware Platform2
Average Test Image Encoding Time (encoding time per bit)	486 ms (1.236 us)	1045 ms (2.657 us)
Average Test Data Packet Encoding Time (encoding time per bit)	66 ms (1.342 us)	136 ms (2.767 us)

Table 8.16: Speed Test 1 results

8.3.2 Decoding Speed Tests

Decoding speed tests measure the time it takes the software to decode IEEE 802.11a data packets when the whole signal is given to the decoder. The test images as well as the test data packets were encoded and saved to file. These files were then given to the receiver to decode and the decoding process was timed. These tests are performed on both the transmitter and receiver hardware platforms as described in Chapter 7. The parameters for this test are shown in Table 8.17.

Parameter	Value
Test type and number:	Speed Test 02 - Decoding
Test data packets:	Test Data Packet 1,2 and 3 Test Image 1,2 and 3
Test data set:	5 encodes of each test file. Total 30 tests.
Total amount of digital bits decoded:	15 x 49152 + 15 x 393216 bits = 6.328 Megabits
Reference carrier phase compensation	Both On and Off states tested
Data influenced by AWGN?	No
Data influenced by sampling frequency drift?	No

Table 8.17: Speed Test 2 parameters

The results from this test are shown in Table 8.18.

The Decoding Times (decoding time per bit)	Hardware Platform1	Hardware Platform2
Average Test Image Decoding Time with Reference Carrier Phase Compensation Enabled	1312.5 ms (3.34 us)	2380 ms (6.05 us)
Average Test Image Decoding Time with Reference Carrier Phase Compensation Disabled	770 ms (1.96 us)	1445 ms (3.68 us)
Average Test Data Packet Encoding Time with Reference Carrier Phase Compensation Enabled	198 ms (4.03 us)	313 ms (6.37 us)
Average Test Data Packet Encoding Time with Reference Carrier Phase Compensation Disabled	140 ms (2.85 us)	176 ms (3.58 us)

Table 8.18: Speed Test 2 results

8.3.3. Conclusions on Speed Performance Tests

By using Equations (6.2), (7.1) and (7.2) it is possible to calculate the test image and test data packet transmission times for both the full bandwidth (20 MHz) and the low bandwidth (44.1 kHz) signals. The results are shown in Table 8.19.

Transmission Times (transmission time per bit)	Full Bandwidth (20 MHz) signal	Low Bandwidth (44.1 kHz) signal
Test Images	16.404 ms (41.72 ns)	7.44 ms (18.92 us)
Test Data Packets	2.068 ms (42.07 us)	0.94 us (19.12 us)

Table 8.19: IEEE 802.11a Transmission times for full and low bandwidth signals

According to queuing theory [13], a system will only be stable when its service time, in this case the length of time the system takes to decode the data, is shorter than the data arrival rate, which is the transmission time and the encoding time.

The average encoding time on hardware platform 1 was 1.29 us per bit but the average decoding time, with the reference carrier phase compensation enabled on hardware platform 1 was 3.69 us per bit. So the system takes longer to decode a signal than to encode it. The average transmission time for the full bandwidth signal is only 41.9 ns per bit and does not influence the arrival rate. The average transmission time for the low bandwidth signal is 19 us per bit which is slow enough to give the decoder chance to finish its work. The results from speed test 1 and 2 show that the current software will not be able to support full bandwidth (20 MHz) real-time IEEE 802.11a signals. The low bandwidth (44.1 kHz) IEEE 802.11a signals on the other hand has sufficiently large transmission time: the decoder has enough time to finish decoding and will thus work.

If we look at the results of the speed tests done on hardware platform 1 (the faster of the two hardware platforms) we see that the test image encoding time is 486ms and the test image decoding time is 1312.5ms. The full bandwidth (20MHz) transmission time of a test image (8.7) is 16.404ms. From these results we can conclude that:

- The decoding time is far longer than the combined encoding time and transmission time. This means that the receiver will not be able to stay ahead in decoding received data. The buffer will finally overflow because the system is unstable.
- For this transceiver system to work well and the available bandwidth to be used efficiently, both the encoding and decoding times must be made smaller than the transmission time.
- This IEEE 802.11a transceiver system will not be able to cope with 20 MHz IEEE 802.11a signals, and must be made faster.

The results from the IEEE 802.11a packet encoding and decoding speed tests and the tests done on hardware platform 2, all come to these same conclusions: The software is currently not fast enough to cope with full bandwidth (20MHz) IEEE 802.11a signals. Please refer to the final chapter about recommendations on speeding up the system.

8.4 Conclusions

The IEEE 802.11a based transceiver system was programmed in C++ and implemented on the hardware platform as shown in the previous chapter. The software was then put through a series of tests to determine how well it worked. A series of simulation tests showed that the results from the implemented decoder indeed followed the theoretically predicted values from Chapter 2. It was also shown that phase compensation due to sub-sample offset resulting from sampling frequency drift is essential. The implemented reference carrier phase compensation algorithm introduced a 1 – 1.2 dB implementation loss due to AWGN. The algorithm also showed a further 1dB implementation loss when decoding signals effected by sampling frequency drift. The initial sound device transmissions tests and buffered real-time test results was very close to the final simulated results which means the simulations are a very close approximation of the real-life system. Further encoding and decoding speed tests, have determined that although the current software works for the (low-speed) low-bandwidth version of the IEEE 802.11a signal, it will not be able to sustain continuous decoding of the full bandwidth (20 MHz) IEEE 802.11a signal. The decoder and encoder are just too slow. The following and final chapter in this thesis concludes everything and has some recommendations on upgrading the IEEE 802.11a transceiver.

Chapter 9

Conclusions

9.1 Introduction

This thesis describes the implementation of an OFDM transceiver system as part of a SDR environment. Orthogonal Frequency Division Multiplexing, a subset of Frequency Division Multiple Access, is a multi-carrier modulation technique that encodes digital data into closely packed orthogonal sub-carriers. Orthogonality ensures that there is no interference between the sub-carriers. The overview into digital modulation techniques shows us how digital data bits can be encoded onto sinusoidal carrier waves. These techniques form the foundation for multi-carrier modulation schemes such as OFDM and can be used as reference when comparing the performance results. OFDM mathematics is studied and expanded into processes where digital data can be encoded and decoded into and from OFDM data symbols in a digital domain. The transmission of OFDM symbol trains across communications channels is prone to many performance-influencing factors. These factors are identified and methods to overcome them are determined and analysed in detail. The implementation of the OFDM transceiver system is based on the IEEE 802.11a standard for encoding and decoding OFDM signals. The IEEE 802.11a standard is very robust in handling these performance influencing factors and is a very efficient OFDM implementation. The SDR implementation of the IEEE 802.11a based transceiver system is done in a programming language called C++. The sound cards in two personal computers (hardware platforms) are connected together using standard stereo copper wires. IEEE 802.11a OFDM data is then encoded and transmitted from the one computer and received and decoded by the other. The software is put through different simulation and transmission tests influenced by AWGN and sampling frequency drift to determine the system performance. The final real-time data performance tests had an approximate 2.2dB implementation loss but still performed as expected and very close to the final simulated test results.

9.2 The IEEE 802.11a Standard and OFDM Modulation

OFDM modulation is a very attractive digital modulation technique and has its fair share of advantages and disadvantages. Some of these advantages are:

- OFDM is scalable in frequency and data rates.
- High bandwidth efficiency.
- Does not require channel equalisation.
- Flexible and adaptive; bit loading of different sub-carriers.
- Good at mitigating the effect of narrowband interferences.
- Does not require a phase lock loop.

Some of the disadvantages of OFDM include:

- OFDM is very processor intensive.
- High PAPR problems and dynamic range issues.
- Very sensitive to sampling frequency drift and phase noise.
- Precise OFDM symbol synchronisation is very important.

The IEEE 802.11a OFDM standard was designed for environments such as offices where office furniture and other objects might cause large multipath delays. High data rate single-carrier modulation techniques have very short symbol periods that need considerable channel equalisation when influenced by large multipath delays. OFDM can obtain the same data rates but will still have relatively long symbol periods. The long OFDM symbol periods are influenced much less by multipath delays, especially with the addition of guard intervals. Office environments also generally have many electronic devices of which some might purposefully and others inadvertently transmit radio waves within the OFDM signal spectrum. This narrowband interference will interfere only with a few OFDM sub-carriers, leaving the rest untouched (due to the orthogonality). Influencing the wideband spectrum of high-data-rate single-carrier modulation signals will influence the whole signal, and ultimately cause continuous decoding errors for the duration of the interference. The advantages that IEEE 802.11a OFDM modulation have over other modulation techniques, especially in the

office environment, does however come at a cost. OFDM modulation is quite processor intensive. For each OFDM data symbol that is decoded a DFT or FFT transform is needed. OFDM is also very sensitive to sampling frequency drift, which causes long term de-synchronisation of the OFDM symbol train and phase errors in all sub-carriers. The IEEE 802.11a OFDM standard utilises so many techniques for combating all the different performance influencing factors that finally a sizable percentage of the available bandwidth and time is used to make sure the data quality is acceptable. The bandwidth efficiency can be calculated as the percentage of data sub-carriers over the total usable sub-carries,

$$n_f = \frac{N_{SD}}{N_{ST}} = \frac{48}{52} = 0.923 \text{ or } 92.3\%. \quad (9.1)$$

The time signal efficiency can be calculated as the actual data signal time over the total OFDM signal time,

$$n_t = \frac{T_{FFT}}{T_{SYM}} = \frac{3.2\mu s}{4.0\mu s} = 0.8 \text{ or } 80\%. \quad (9.2)$$

The total efficiency can be calculated as the product of the bandwidth and time signal efficiencies,

$$n = n_f * n_t = 0.8 * 0.923 = 0.738 \text{ or } 73.85\%. \quad (9.3)$$

This means that the final IEEE 802.11a OFDM system uses 26.15% of its available time and bandwidth resources just to make sure the decoded signal quality is acceptable. With the rising cost and scarcity of available bandwidth one can ask whether IEEE 802.11a should be used at all. On the one hand, IEEE 802.11a utilises a part of the frequency spectrum called the ISM band, which is an unlicensed band and free for all to use. By using a part of the frequency spectrum that is free one could argue that since no one is paying for it, it doesn't really matter if we waste a small part of it, if it would ensure a good performance. In such a case the use of IEEE 802.11a in office environments is a very good choice. On the other hand, it is not in the least surprising to find that the IEEE 802.11 standards now already include various

upgrades after the IEEE 802.11a standard, such as IEEE 802.11b, IEEE 802.11n and IEEE 802.11g.

9.3 Software Defined Radio

Software defined radio is a very interesting and noble idea for the future development of new radio communications devices as well as other types of electronic devices. It is a fact that with the advances in technology, processors are becoming smaller and more powerful. This thesis is not however involved in the development of SDR equipment. The software written for the purpose of this thesis was tested on a personal computer, which could be seen as a very powerful SDR device, but which is not the final intended implementation of this system. As to date the SDR group at the University of Stellenbosch's electronic engineering faculty is still relatively new, and a generic hardware platform is not yet ready for use.

9.4 Notes on the C++ software

The C++ software was written to be generic and modular, and to use the most common C++ functions and libraries so it would be compatible with most of the existing C++ compilers available for embedded systems. For this reason complex values used in the different functions were handled as two separate values, one for the real and the other the imaginary part of the complex value. In other cases, the magnitude and phase representation of the complex value were used. Furthermore, it seems that there is still a small problem with the Port Audio sound card interface software. It seems that there might still be an error in the recording and/or playback callback functions in the OFDM_audio library. During recording callbacks when the receiver is receiving and buffering the incoming OFDM transmission, some small buffer segments get lost or skipped. This causes catastrophic failure in the OFDM decoding algorithm, since it translates to synchronisation errors larger than what the decoder is set to handle. The IEEE 802.11a transceiver software works as it is, but might not be running optimally and a more experienced programmer might be able to streamline the software and even improve the data performance results.

9.5 SDR Performance Tests

The IEEE 802.11a OFDM transceiver system was tested to determine how well the software performed in different circumstances. Seven simulation tests tested the performance of the decoder, by letting it decode artificially influenced OFDM signals. It was found that the basic decoder with the reference carrier phase compensation algorithm disabled performed precisely as predicted in the presence of AWGN. The BER graph followed the theoretically predicted results very closely. The same test, but with the reference carrier phase compensation enabled, showed a 1 – 1.2dB implementation loss. This basically means that the noise on the reference carriers affects the phase compensation and causes extra decoding errors. In the simulation tests where the OFDM signal was influenced by sampling frequency drift, the performance results showed another 1dB implementation loss. What this means is that the reference carrier phase compensation algorithm was unable to completely mitigate the resulting phase shifts on the sub-carriers. Finally, the initial sound device transmission test results and the real-time buffered test results are so to say the same as the final simulation tests. This is very good since it means that our simulations are a very close approximation of the real-life system. We can now see from the different simulation tests which part of the transceiver caused the different implementation losses and could need some extra work. The final IEEE 802.11a OFDM transceiver performance graphs can be seen in Figures 8.13 and 8.14. The IEEE 802.11a transceiver system was further tested and put through a series of speed tests. These tests timed the IEEE 802.11a packet encoding and decoding processes. It was found that the current software and hardware platform would not be able to continuously receive and decode full bandwidth (20MHz) IEEE 802.11a packets as intended by the IEEE 802.11a specifications. The current software is just too slow.

9.6 Future Work and Research

Future work on the IEEE 802.11a OFDM software include:

- Programming and implementing a convolutional encoder and decoder.
- Expanding the sub-carrier modulation software to include BPSK, 16-QAM and 64-QAM encoders and decoders.
- Improving the reference carrier phase compensation algorithms for better noise performance and faster operation.
- Optimising and streamlining the existing algorithms.
- Converting the DFT and IDFT function to FFT and IFFT functions to make the system run faster.

Future work on the IEEE 802.11a OFDM hardware include:

- Using a high bandwidth ADC and DAC instead of the computer sound card to use the full IEEE 802.11a OFDM bandwidth.
- Replacing the stereo copper cables which act as the communications medium with a appropriate analogue front-end and antenna.
- Porting the IEEE 802.11a software to an embedded system and creating a complete IEEE 802.11a transceiver.

Additional tests to be done on the IEEE 802.11a OFDM system

- Testing the multipath effect on the IEEE 802.11a transceiver data performance after upgrading the hardware with high bandwidth ADC's DAC's and analogue front ends.
- Testing the data performance after the convolutional encoding FEC software is installed.

9.7 Comparing against other existing OFDM systems.

Due to the lack of an appropriate RF front-end for transmitting and receiving the IEEE 802.11a OFDM packets over the air, we are unable to test the currently implemented IEEE 802.11a transceivers' multipath performance. Multipath fading is a definite performance-influencing factor for any RF transceiver system.

To get a general idea of how multipath affects RF transceiver systems, the results from this thesis were compared against simulated results in [26]. In [26] the performance of an OFDM modulated signal with QPSK sub-carriers as well as a single-carrier high-speed QPSK signal were simulated over different fading channels. In one of the simulations the two modulated signals were given extra error correction capability. The receiver had the ability of correcting 6 bits in each 64-bit packet. Their performances were simulated over a two-path Rayleigh fading channel. (Basically a Rayleigh fading channel is a non-line-of-sight channel. In this simulation the two Rayleigh channels had equal power). The BER performance of the two modulated signals, each with and without the error correcting ability, together with the results from our real-time buffered transmission test and simulation test 2 (OFDM QPSK with no influences) were compared. The resulting graph in Figure 9.1 shows the relative performances.

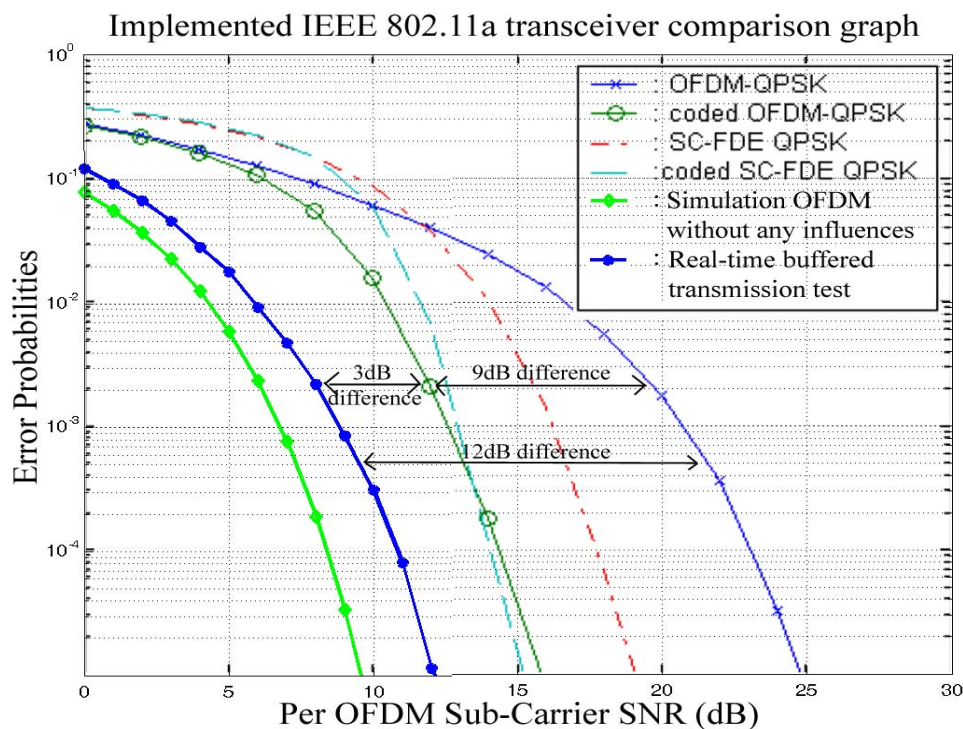


Figure 9.1: Implemented IEEE 802.11a transceiver comparison graph

The 12dB difference between our real-time buffered transmission graph and the uncoded OFDM-QPSK BER graph from [26] shows us just how devastating multipath effects can be.

Just as interesting is the ability of the error correction to correct faulty bits and to boost the data quality from a relative 12dB implementation loss to a mere 3dB. The error correction scheme thus introduced a 9dB coding gain, which is very impressive.

From these results it is clear that multipath and error correction both play big roles in influencing the performance of RF transmitted signals. They would thus undoubtedly also influence the performance of our implemented IEEE 802.11a transceiver once it is fully implemented.

9.8 Conclusions

The IEEE 802.11a OFDM transceiver system build for the purpose of this thesis works and has an approximate 2.2 dB data performance implementation loss.

9.9 Thanks

I would like to thank the University of Stellenbosch, the Electrical and Electronic Engineering department, the Digital Signal Processing group, the Software Defined Radio group, and everyone involved, for giving me the opportunity to work on this exciting project.

Appendix A

Useful Mathematical Proofs

A.1 Useful Trigonometry Functions:

$$\sin(a + b) = \sin(a) \cos(b) + \cos(a) \sin(b) \quad (\text{A.1})$$

$$\sin(a - b) = \sin(a) \cos(b) - \cos(a) \sin(b) \quad (\text{A.2})$$

$$\cos(a + b) = \cos(a) \cos(b) - \sin(a) \sin(b) \quad (\text{A.3})$$

$$\cos(a - b) = \cos(a) \cos(b) + \sin(a) \sin(b) \quad (\text{A.4})$$

$$\sin^2(x) = (0.5)(1 - \cos(2x)) \quad (\text{A.5})$$

$$\cos^2(x) = (0.5)(1 + \cos(2x)) \quad (\text{A.6})$$

A.2 Finding The Zero-points of a Sinc Waveform

A sinc can be expressed as

$$\text{sinc}(ft) = \left(\frac{\sin(\pi ft)}{\pi ft} \right). \quad (\text{A.7})$$

The sinc zeroes are found where $\sin(\pi ft) = 0$, and we know that a sine wave is equal to zero every 180 degrees or π radians,

$$\sin(\phi) = 0 \quad \text{where } \phi = m\pi \quad ; \quad m \in [-\infty, \dots, 0, \dots, \infty]. \quad (\text{A.8})$$

If we now substitute the parameters for the sine wave from (A.7) into Equation (A.8) we get

$$\sin(\pi ft) = 0 \quad \text{where } \pi ft = m\pi \quad ; \quad m \in [-\infty, \dots, 0, \dots, \infty]. \quad (\text{A.9})$$

This means that the sine wave from Equation (A.7) has zeroes at the following locations,

$$\sin(\pi ft) = 0 \quad \text{where } f = \frac{m}{t} \quad ; \quad m \in [-\infty, \dots, 0, \dots, \infty]. \quad (\text{A.10})$$

This means that the sinc waveform (A.7) has zeroes at the same locations as (A.10) so that

$$\text{sinc}(\pi ft) = 0 \quad \text{where } f = \frac{m}{t} \quad ; \quad m \in [-\infty, \dots, 0, \dots, \infty]. \quad (\text{A.11})$$

By choosing $m=0$, t and f becomes zero and (A.11) produces an undefined and unusable result

$$\text{sinc}(0) = \left(\frac{\sin(0)}{0} \right) = \frac{0}{0} \quad (\text{undefined}). \quad (\text{A.12})$$

(A.12) is solved by applying L'Hospital's rule to (A.7),

$$\text{sinc}(0) = \lim_{t \rightarrow 0} \left(\frac{\frac{d}{dt} \sin(\pi ft)}{\frac{d}{dt} \pi ft} \right), \quad (\text{A.13})$$

which reduces to

$$\text{sinc}(0) = \lim_{t \rightarrow 0} (\cos(\pi ft)) = 1. \quad (\text{A.14})$$

The value of the sinc function for an integer-valued argument can finally be expressed as

$$\text{sinc}(ft) = \left\{ \begin{array}{l} 0; \quad \text{where } f = \frac{m}{t} \quad ; \quad m \in [1, 2, \dots, \infty] \\ 1; \quad \text{where } f = 0 \end{array} \right\}. \quad (\text{A.15})$$

A.3 RMS of a sine wave

If a sinusoidal signal is given by $y(t)$ so that

$$y(t) = A * \cos(2\pi ft + \phi). \quad (\text{A.16})$$

The Root-Mean-Square (RMS) of (A.16) can be calculated as

$$y_{\text{RMS}}(t) = \sqrt{\left(\frac{1}{T} \int_0^T y^2(t) dt \right)}. \quad (\text{A.17})$$

Substituting the sinusoidal signal (A.16) into Equation (A.17) gives

$$y_{\text{RMS}}(t) = \sqrt{\left(\frac{1}{T} \int_0^T A^2 \cos^2(2\pi ft + \phi) dt \right)}. \quad (\text{A.18})$$

Substituting the trigonometry function (A.9) into Equation (A.18) yields

$$y_{\text{RMS}}(t) = \sqrt{\left(\frac{A^2}{2T} \int_0^T 1 + \cos(4\pi ft + \phi) dt \right)}. \quad (\text{A.19})$$

Equation (A.19) can then be reduced to

$$y_{\text{RMS}}(t) = \sqrt{\left(\frac{A^2}{2T} \int_0^T 1 dt + \frac{A^2}{2T} \int_0^T \cos(4\pi ft + \phi) dt \right)}. \quad (\text{A.20})$$

Equation (A.20) can further be reduced to

$$y_{\text{RMS}}(t) = \sqrt{\left(\frac{A^2}{2} + \frac{A^2}{2T} \frac{[\sin(4\pi ft + \phi)]_0^T}{4\pi f} \right)}. \quad (\text{A.21})$$

This gives

$$y_{\text{RMS}}(t) = \sqrt{\left(\frac{A^2}{2} + \frac{A^2 \sin(4\pi f\Gamma + \phi)}{4\pi f} \right)}. \quad (\text{A.22})$$

The integral of the sine wave is zero for all multiples of 2π , thus (A.22) will become

$$y_{\text{RMS}}(t) = \sqrt{\left(\frac{A^2}{2} \right)} = \frac{A}{\sqrt{2}}. \quad (\text{A.23})$$

Appendix B

Digital Modulation Schemes

B.1 Binary Phase Shift Keying (BPSK) modulation

A BPSK encoder takes one digital data bit (b_0) and converts it to one of 2 possible complex values represented by its I (real) and Q (imaginary) channel value.

Input Bit (b_0)	I-Out	Q-Out
0	-1	0
1	1	0

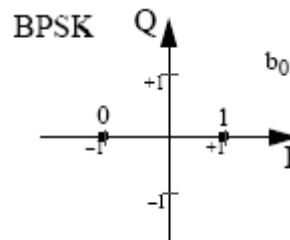


Table B.1: BPSK encoding table

Figure B.1: BPSK constellation bit encoding

B.2 Quadrature Phase Shift Keying (QPSK) encoding

A QPSK encoder takes two digital data bits (b_0b_1) and converts it to one of 4 possible complex values represented by its, I (real) and Q (imaginary) channel values.

Input Bit (b_0)	I-Out
0	-1
1	1

Input Bit (b_1)	Q-Out
0	-1
1	1

Table B.2: QPSK encoding table

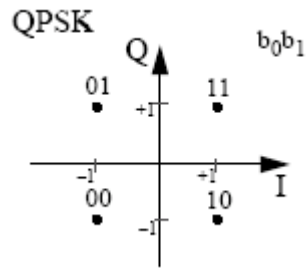


Figure B.2: QPSK constellation bit encoding

B.3 16-Quadrature Amplitude Modulation (16-QAM) encoding

A 16-QAM encoder takes four digital data bits ($b_0b_1b_2b_3$) and converts it to one of 16 possible complex values represented by their I (real) and Q (imaginary) channel values.

Input Bit (b_0b_1)	I-Out
00	-3
01	-1
10	1
11	3

Input Bit (b_2b_3)	Q-Out
00	-3
01	-1
10	1
11	3

Table B.3: 16-QAM encoding table

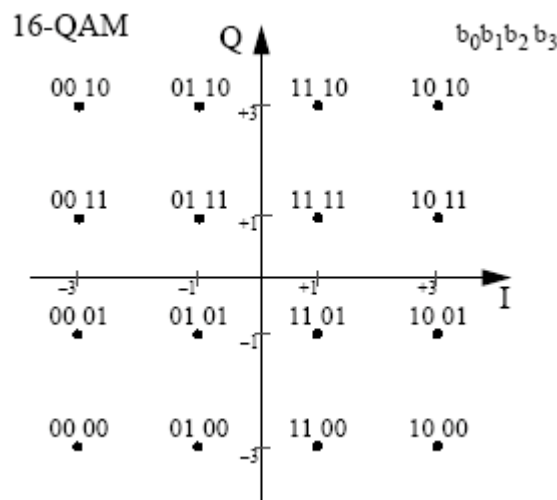


Figure B.3: 16-QAM constellation bit encoding

B.4 64-Quadrature Amplitude Modulation (QAM-64) encoding

A QAM-64 encoder takes six digital data bits ($b_0b_1b_2b_3b_4b_5$) and converts it to one of 64 possible complex values represented by their I (real) and Q (imaginary) channel values.

Input Bit ($b_0b_1 b_2$)	I-Out	Input Bit ($b_3b_4b_5$)	Q-Out
000	-7	000	-7
001	-5	001	-5
010	-3	010	-3
011	-1	011	-1
100	1	100	1
101	3	101	3
110	5	110	5
111	7	111	7

Table B.4: QAM-64 encoding table

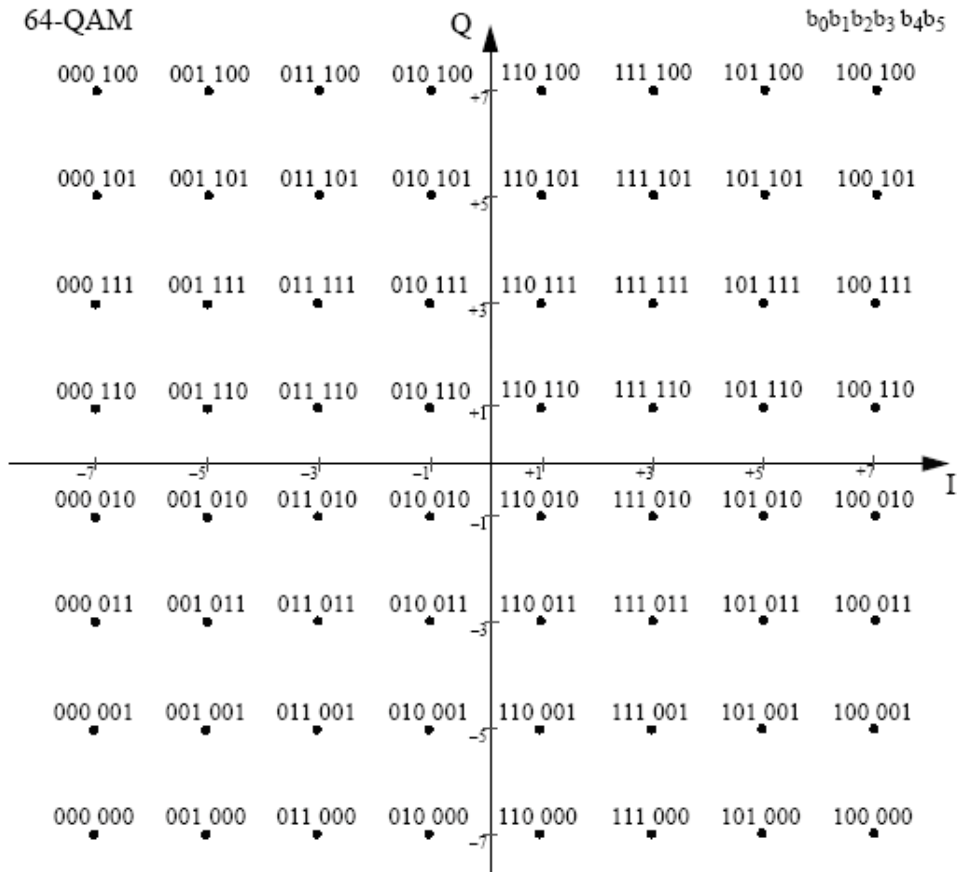


Figure B.4: 64-QAM constellation bit encoding

Appendix C

SDR OFDM Program Details

C.1 The OFDM Specification Structure Details

The OFDM specification structure buffers and variables are shown in Table C.1.

Structure Name		OFDM_Specification_struct
Variable Type	Variable Name	Variable Description
Integer	Fs	The Sampling Frequency value
Integer	FcMax	The Maximum Usable Frequency
Integer	Ntotal	Total Amount of Sub-carriers
Integer	Nst	Total Amount of used Sub-carriers
Integer	Nsd	Total Amount of Data Sub-carriers
Integer	Nsp	Total Amount of Reference Sub-carriers
Float	DeltaF	Sub-Carrier Frequency Spacing
Float	Tfft	Length of a pre-GI symbol (3.2us)
Float	Tgi	Guard interval length (0.8us)
Float	Tgi2	Guard interval of Long preambles length (1.6us)
Float	Tsignal	Length of the signal symbol
Float	Tlongpreamble	Length of the long preamble (8us)
Float	Tshortpreamble	Length of the short preamble (8us)
Integer	Nfft	Amount of samples in a OFDM symbol
Integer	Ngi	Amount of samples in a symbol guard interval
Integer	Ngi2	Amount of samples in a long preamble GI
Integer	Nsignal	Amount of samples in a OFDM symbol with its GI
Integer	Nlongpreamble	Amount of samples in long preamble
Integer	Nshortpreamble	Amount of samples in a short preamble
Integer	BitsPerSubCarrier	Amount of bits encoded in each OFDM sub-carrier

Structure Name		OFDM_Specification_struct
Variable Type	Variable Name	Variable Description
Integer	*pDataCarrier Positions	Pointer to an array that stores the positions of the data sub-carriers in an OFDM symbol
Integer	*pRefCarrier Positions	Pointer to an array that stores the positions of the reference sub-carriers in an OFDM symbol
Float	*pDataAndRef PhaseValues	Pointer to an array the stores the phase values of the data and reference sub-carriers
Float	*pDataAndRef MagnValues	Pointer to an array that stores the magnitude values of the data and reference sub-carriers
Float	*pDataAndRef RealValues	Pointer to an array that stores the real values of the data and reference sub-carriers
Float	*pDataAndRef ImagValues	Pointer to an array that stores the imaginary values of the data and reference sub-carriers
Float	Construction Magnitude	The standard magnitude value of all the data and reference sub-carriers
Float	*pRefRealValues	Pointer to an array that stores the real values of the reference sub-carriers
Float	*pRefImagValues	Pointer to an array that stores the imaginary values of the reference sub-carriers
Float	*pLongPreamble ImagPosValues	Pointer to an array that stores the imaginary positive frequency values for the long preamble
Float	*pLongPreamble ImagNegValues	Pointer to an array that stores the imaginary negative frequency values for the long preamble
Float	*pLongPreamble RealPosValues	Pointer to an array that stores the real positive frequency values for the long preamble
Float	*pLongPreamble RealNegValues	Pointer to an array that stores the real negative frequency values for the long preamble
Float	LongPreamble Multiplier	A long preamble amplitude multiplier (Debugging)

Structure Name		OFDM_Specification_struct
Variable Type	Variable Name	Variable Description
Float	MyLongPreambleMultiplier	Another long preamble amplitude multiplier (Debugging)
Float	*pLongPreambleRealTimeArray	Pointer to an array that stores the real time domain part of the long preamble
Float	*pLongPreambleImagTimeArray	Pointer to an array that stores the imaginary time domain part of the long preamble
Float	*pShortPreambleImagPosValues	Pointer to an array that stores the imaginary positive frequency values for the short preamble
Float	*pShortPreambleImagNegValues	Pointer to an array that stores the imaginary negative frequency values for the short preamble
Float	*pShortPreambleRealPosValues	Pointer to an array that stores the real positive frequency values for the short preamble
Float	*pShortPreambleRealNegValues	Pointer to an array that stores the real negative frequency values for the short preamble
Float	ShortPreambleMultiplier	A short preamble amplitude multiplier (Debugging)
Float	MyShortPreambleMultiplier	Another short preamble amplitude multiplier (Debugging)
Float	*pTempShortPreambleRealTimeArray	Pointer to an array that temporarily stores the real time domain part of the short preamble
Float	*pTempShortPreambleImagTimeArray	Pointer to an array that temporarily stores the imaginary time domain part of the short preamble
Float	*pShortPreambleRealTimeArray	Pointer to an array that stores the real time domain part of the short preamble
Float	*pShortPreambleImagTimeArray	Pointer to an array that stores the imaginary time domain part of the short preamble
Float	*pLongPreambleGIRealTimeArray	Pointer to an array that stores the guard interval of the real time domain part of the long preamble

Structure Name		OFDM_Specification_struct
Variable Type	Variable Name	Variable Description
Float	*pLongPreamble GIImagTimeArray	Pointer to an array that stores the guard interval of the imaginary time domain of the long preamble
Float	*pSignalRealTime Array	Pointer to an array that stores the real time domain part of the signal symbol
Float	*pSignalImagTime Array	Pointer to an array that stores the imaginary time domain part of the signal symbol
Float	*pSignalRealGITi meArray	Pointer to an array that stores the real time domain part of the signal symbol guard interval
Float	*pSignalImagGITi meArray	Pointer to an array that stores the imaginary time domain part of the signal symbol guard interval
Float	*pSignalwithGIRea lTimeArray	Pointer to an array that stores the real time domain part of the signal symbol with its guard interval
Float	*pSignalwithGIIma gTimeArray	Pointer to an array that stores the imaginary time domain of the signal symbol guard interval with its guard interval
Float	*pLongPreamble1 Magn;	Pointer to an array that stores the first received long preamble frequency spectrum magnitudes
Float	*pLongPreamble2 Magn;	Pointer to an array that stores the second received long preamble frequency spectrum magnitudes
Float	*pLongPreambleM eanMagn;	Pointer to an array that stores the mean received long preamble frequency spectrum magnitudes
Float	*pInverseChannelF ilter;	Pointer to an array that stores the inverse channel transfer function filter, estimated from the long preamble frequency spectrum magnitudes
Long double	**DFT_Wreal	Pointer to a matrix of real value twiddle factors for the DFT function
Long double	**DFT_Wimag	Pointer to a matrix of imaginary value twiddle factors for the DFT function

Structure Name		OFDM_Specification_struct
Variable Type	Variable Name	Variable Description
Long double	** IDFT_Wimag	Pointer to a matrix of imaginary value twiddle factors for the IDFT function
Float	*pRealTimeArray	Pointer to an array that stores the real time domain signal for use by the DFT/IDFT function
Float	*pImagTimeArray	Pointer to an array that stores the imaginary time domain signal for use by the DFT/IDFT function
Float	*pRealFreqArray	Pointer to an array that stores the real frequency spectrum for use by the DFT/IDFT function
Float	*pImagFreqArray	Pointer to an array of the imaginary frequency spectrum for use by the DFT/IDFT function
Float	*pMagnFreqArray	Pointer to an array of magnitude frequency spectrum for use by the DFT/IDFT function
Float	*pPhaseFreqArray	Pointer to an array of the phase frequency spectrum for use by the DFT/IDFT function
Int	Is_OFDM_Specs_Initiated	Equal 1 if the OFDM specification structure has been initiated

Table C.1: The OFDM Specification Structure Details

C. 2 The OFDM Transmitter Structure Details

The OFDM transmitter structure buffers and variables are shown in Table C.2.

Structure Name		OFDM_Transmitter_struct
Variable Type	Variable Name	Variable Description
Boolean	TransmitImage	Is true if we are going to transmit an image
Boolean	TransmitData	Is true if we are going to transmit a test data

Structure Name		OFDM_Transmitter_struct
Variable Type	Variable Name	Variable Description
Boolean	TransmitImageIs Loaded	Is true if the transmit image is already loaded
Boolean	TransitDataIs Loaded	Is true if the transmit test data packet is already loaded
Integer	TotalBytes	Stores the amount of bytes in the transmission
Integer	TotalBits	Stores the amount of bits in the transmission
Integer	TotalSymbols	Stores the amount of IEEE 802.11a OFDM data symbols in the transmission
Integer	TotalBitsPer Symbol	Stores the amount of bits we encode per OFDM data symbol
Byte	*pImageBytes	Pointer to an array that stores the transmission bytes
Byte	*PimageBits	Pointer to an array that stores the transmission bits
Integer	Entire_Transmission_Length	Stores the entire transmission sample length
Float	*pEntire_Transmission_I	Pointer to an array that stores the entire transmission real (I-channel) samples
Float	*pEntire_Transmission_Q	Pointer to an array that stores the entire transmission imaginary (Q-channel) samples
Float	*pSymbol_I	Pointer to an array that stores a temporary OFDM symbol (I-channel)
Float	*pSymbol_Q	Pointer to an array that stores a temporary OFDM symbol (Q-channel)
Ansi String	TransmitFilename	The Filename of the file that contains the data we want to transfer
Boolean	GotTransmit Filename	True if the GUI retrieved a valid filename
char	TransmitFilename Char[1024]	Array of chars that stores the filename

Structure Name		OFDM_Transmitter_struct
Variable Type	Variable Name	Variable Description
char	*prtTransmitFile CharStr	Pointer to the character string that stores the transmit filename
Float	SNR	The Signal-to-noise ratio of the AWGN to be added to the transmission
Float	Vratio	The linear ratio of signal to noise of the AWGN
Bool	Is_Noise_Added	True if AWGN has been added to the signal
Bool	Is_OFDM_Transmitter_Initiated	True if the OFDM transmitter structure has been initiated
Bool	Is_OFDM_Output_Data_Initiated	True if the transmission data has been loaded
Bool	Is_OFDM_Encoding_Complete	True if the OFDM encoding is complete

Table C.2: The OFDM Transmitter Structure Details

C. 3 The OFDM Receiver Structure Details

The OFDM receiver structure buffers and variables are shown in Table C.3.

Structure Name		OFDM_Receiver_struct
Variable Type	Variable Name	Variable Description
Boolean	TransmitImage	Is true if we are going to receive an image
Boolean	TransmitData	Is true if we are going to receive a test data
Integer	ReceiveBufferTime Length	The receiving buffer length in seconds
Integer	TotalBytes	The amount of bytes we expect to receive
Integer	TotalBits	The amount of bits we expect to receive

Structure Name		OFDM_Receiver_struct
Variable Type	Variable Name	Variable Description
Integer	TotalSymbols	The total amount of IEEE 802.11a OFDM data symbols to expect in the transmission
Integer	TotalBitsPer Symbol	The total amount of bits that can be encoded into one of our IEEE 802.11a OFDM data symbols
Integer	Entire_Transmission_Length	The Length in samples of the Entire transmission array
Float	*pEntire_Transmission_I	Pointer to an array that stores the samples of the receiving transmission (I-channel)
Float	*pEntire_Transmission_Q	Pointer to an array that stores the samples of the receiving transmission (Q-channel)
Integer	ET_Poll_Loc	The position in the transmission we are polling during the decoding cycle
Integer	KeepIndex	The Index number from which point we want to save the rest of the primary buffer at the next entire transmission array polling
Integer	PrimBufferSize	The length of the primary decoding buffer
Float	*pPrimBuffer_I	Pointer to the primary decoding array (I-channel)
Float	*pPrimBuffer_Q	Pointer to the primary decoding array (Q-channel)
Integer	AmountOfTimesIn State1	Stores the amount of times the decoding state machine was in state 1
Integer	AmountOfTimesIn State2	Stores the amount of times the decoding state machine was in state 2
Integer	AmountOfTimesIn State3	Stores the amount of times the decoding state machine was in state 3
Integer	AmountOfTimesIn State4	Stores the amount of times the decoding state machine was in state 4
Ansi String	ReceiveFilename	Stores the name of the file that contains the receiving signal

Structure Name		OFDM_Receiver_struct
Variable Type	Variable Name	Variable Description
Boolean	GotReceive Filename	True if the demo GUI successfully retrieved the filename of the receiving file
Char	ReceiveFilename Char[1024	Array of chars containing the receive filename
Char	*ptrReceiveFile CharStr	Pointer to the array that stores the filename of the receiving filename
Boolean	ReceiveFileIsA DataFile	True if the file we want to load is a data file
Boolean	ReceiveFileIsA WaveFil	True if the file we want to load is a wave file
Integer	State	The decoder state machine current state
Boolean	StopFileDecoder	True if we want to stop the decoder state machine
Integer	TimeInLastState	The amount of times spent in the previous state
Float	State1_Threshold	State 1 signal level threshold
Integer	Threshold_ Location	State 1 threshold exceeded locations
Float	*SP10_xcorr_ Results_I	Pointer to array that stores the result of the cross correlation of the receiving primary buffer and the 10 short preambles (I-channel)
Float	*SP10_xcorr_ Results_Q	Pointer to array that stores the result of the cross correlation of the receiving primary buffer and the 10 short preambles (Q-channel)
Float	*LP2_xcorr_ Results_I	Pointer to array that stores the result of the cross correlation of the receiving primary buffer and the 2 long preambles (I-channel)
Float	*LP2_xcorr_ Results_Q	Pointer to array that stores the result of the cross correlation of the receiving primary buffer and the 2 long preambles (Q-channel)

Structure Name		OFDM_Receiver_struct
Variable Type	Variable Name	Variable Description
Float	*SP10_LP2_xcorr_Results_I	Pointer to array that stores the result of the cross correlation of the receiving primary buffer and the 10 short and 2 long preambles (I-channel)
float	*SP10_LP2_xcorr_Results_Q	Pointer to array that stores the result of the cross correlation of the receiving primary buffer and the 10 short and 2 long preambles (Q-channel)
Integer	SP10_xcorr_Results_Length	Sample length of the SP10_xcorr_Results (I and Q –channel)
Integer	LP2_xcorr_Results_Length	Sample length of the LP2_xcorr_Results (I and Q –channel)
Integer	SP10_LP2_xcorr_Results_Length	Sample length of the SP10_LP2_xcorr_Results (I and Q –channel)
Float	*SP10_I	Pointer to array that stores 10 short preambles (I-channel)
Float	*SP10_Q	Pointer to array that stores 10 short preambles (Q-channel)
Float	*LP2_I	Pointer to array that stores 2 long preambles (I-channel)
Float	*LP2_Q	Pointer to array that stores 2 long preambles (Q-channel)
Float	*SP10_LP2_I	Pointer to array that stores 10 short and 2 long preambles (I-channel)
Float	*SP10_LP2_Q	Pointer to array that stores 10 short and 2 long preambles (Q-channel)
Float	SP10_I_xcorr_best_value	Stores the value of the peak in the 10 short preamble cross correlation result (I-channel)
Integer	SP10_I_xcorr_best_index	Stores the location of the peak in the 10 short preamble cross correlation result (I-channel)

Structure Name		OFDM_Receiver_struct
Variable Type	Variable Name	Variable Description
Float	SP10_Q_xcorr_best_value	Stores the value of the peak in the 10 short preamble cross correlation result (Q-channel)
Integer	SP10_Q_xcorr_best_index	Stores the location of the peak in the 10 short preamble cross correlation result (Q-channel)
Float	LP2_I_xcorr_best_value	Stores the value of the peak in the 2 long preamble cross correlation result (I-channel)
Integer	LP2_I_xcorr_best_index	Stores the location of the peak in the 2 long preamble cross correlation result (I-channel)
Float	LP2_Q_xcorr_best_value	Stores the value of the peak in the 2 long preamble cross correlation result (Q-channel)
Integer	LP2_Q_xcorr_best_index	Stores the location of the peak in the 2 long preamble cross correlation result (Q-channel)
Float	SP10_LP2_I_xcorr_best_value	Stores the value of the peak in the 10 short and 2 long preamble cross correlation result (I-channel)
Integer	SP10_LP2_I_xcorr_best_index	Stores the location of the peak in the 10 short and 2 long preamble cross correlation result (I-channel)
Float	SP10_LP2_Q_xcorr_best_value	Stores the value of the peak in the 10 short and 2 long preamble cross correlation result (Q-channel)
Integer	SP10_LP2_Q_xcorr_best_index	Stores the location of the peak in the 10 short and 2 long preamble cross correlation result (Q-channel)
Integer	SP_Pos	The location of the short preamble
Integer	LP1_Pos	The location of the first long preamble
Integer	LP2_Pos	The location of the second long preamble
Integer	Signal_Pos	The location of the signal symbol
Integer	NextDataSymbol_Pos	The location of the next data symbol
Integer	Data_Phases_In_S_Locs_Neg[24]	Negative frequency spectrum locations of the data phases in OFDM symbols

Structure Name		OFDM_Receiver_struct
Variable Type	Variable Name	Variable Description
Integer	Data_Phases_In_ S_Locs_Pos[24];	Positive frequency spectrum locations of the data phases in OFDM symbols
Integer	Data_Phases_In_ CPP_Locs_Neg[24]	Negative frequency spectrum locations in an C++ array of the data phases in OFDM symbols
Integer	Data_Phases_In_ CPP_Locs_Pos[24]	Positive frequency spectrum locations in an C++ array of the data phases in OFDM symbols
Byte	*ReceivedBytes	Pointer to an array of the decoded data bytes
Integer	NextReceived ByteIndex	Index to the location of the next data byte in the *ReceivedBytes array
Byte	*ReceivedBits	Pointer to an array of the decoded data bits
Integer	NextReceiveBit Index	Index to the location of the next data bit in the *ReceivedBites array
Float	*NoRecord	Pointer to a array that stores each sub-sample offset during the decoding process
Integer	SymbolsReceived	Stores the amount of decoded OFDM data symbols
Integer	DecodingComplete	Equals 1 if decoding is complete
Integer	*CarrierBitErrors;	Pointer to an array that stores the amount of bits errors in each sub-carriers
Integer	LastSymbol BeforeAdjustment	The Symbol number of the last time we did a sample shift adjustment
Integer	NoGradient	Estimated gradient of the sub-sample offset change
Boolean	GotNoGradient	True when a valid gradient has been estimated
Boolean	RefCarrierPhase Comp	True when Reference Carrier Phase Compensation has been enabled
Boolean	Is_OFDM_ Receiver_Initiated	True when the OFDM receiver structure has been initiated

Table C.3: The OFDM Receiver Structure Details

C. 4 The OFDM Comparer Structure Details

The OFDM comparer structure buffers and variables are shown in Table C.4.

Structure Name		OFDM_Comparer_struct
Variable Type	Variable Name	Variable Description
Char	*ptrCompare FileCharStr	Pointer to an array of chars that store the compare filename
Char	CompareFilename Char[1024];	Array of chars that store the compare filename
Ansi String	CompareFilename	The compare filename
Boolean	GotCompare Filename	True when the GUI successfully retrieved the compare filename
Boolean	CompareFileIs Image	True if the compare file is an image
Boolean	CompareFileIsData	True if the compare file is a data file
Boolean	CompareFile Loaded	True if the compare file has been loaded
Byte	*pCompareBytes	Pointer to an array that stores all the comparative bytes
Byte	*pCompareBits	Pointer to an array that stores all the comparative bits
Integer	TotalBits	Total amount of bits that was compared
Integer	TotalErrorBits	Total amount of incorrect compare bits
Float	BER	Bit Error Rate of received data

Table C.4: The OFDM Comparer Structure Details

C. 5 The OFDM Audio Structure Details

The OFDM audio structure buffers and variables are shown in Table C.5.

Structure Name		OFDM_Audio_struct
Variable Type	Variable Name	Variable Description
Boolean	AudioInited	True if sound device has been initiated
Boolean	AudioPortInit ErrorFound	True if AudioPort external library has an error
Integer	AudioPortDevices	Amount of available sound devices detected
Integer	SelectedAudioPort Device	Currently selected audio device
Const	PaDeviceInfo	[*AudioPortDeviceIndo] Selected device info linked to outside library
Boolean	Is_OFDM_Audio_ Initiated;	True if OFDM audio structure has been initiated
Boolean	Is_OFDM_Audio_ Device_Selected	True if a sound device has been selected
Boolean	Is_OFDM_Audio_ Ready_To_Start;	True is audio is ready!

Table C.5: The OFDM Audio Structure Details

C. 6 The Main OFDM Library Functions

The main OFDM library functions are shown in Table C.6.

Library Name	OFDM
Function Name	Function Description
OFDM_WF(.)	Writes a float array to file (debugging only)
OFDM_Initiate_Specifications()	Initiated OFDM specifications buffer
Clean_CleanAllTimeBuffers()	Clean the DFT/IDFT time buffers
Clean_CleanAllFreqBuffers()	Clean the DFT/IDFT frequency buffers
OFDM_Initiate_All()	Calls initiation sub-functions
DFT_IDFT()	The IDFT function
DFT_DFT()	The DFT function
FFT_Convert_RealAndImag_To_MagnAndPhase()	Convert the real and imaginary frequency spectrum into magnitude and phase values
FFT_Create_Twiddle_Factors()	Create the DFT/IDFT twiddle factors
OFDM_Create_Long_Preamble()	Creates the IEEE 802.11a long preamble
OFDM_Create_Long_Preamble_GI()	Creates the IEEE 802.11a long preamble guard interval
OFDM_Create_Short_Preamble()	Creates the IEEE 802.11a short preamble

Table C.6: The main OFDM library functions

C. 7 The OFDM Math Library Functions

The OFDM math library functions are shown in Table C.7.

Library Name	OFDM_math
Function Name	Function Description
Math_CrossCorrelation()	Calculates the cross correlation between two input float-type buffers

Library Name	OFDM_math
Function Name	Function Description
Math_MaxBufferValue()	Returns the maximum value inside a float-type array
Math_MaxBufferIndex()	Returns the index of the maximum value inside a float-type array
Math_MinBufferValue()	Returns the minimum value inside a float-type array
Math_MinBufferIndex()	Returns the index of the minimum value inside a float-type array
Math_RMSBufferValue()	Returns the Root-Mean-Square (RMS) value of a float-type array
Math_QPSK_PhaseTo2Bits()	Demodulates a QPSK phase into 2 bits
Math_QPSK_PhaseTo2Bits Array()	Demodulates an array of QPSK phases into a array of bits
Math_BitToInt()	Converts a 8-bit binary array into an integer value
Math_BitStreamToByteStream()	Converts a 8-bit binary array stream into a integer array
Math_QPSK_2BitsToPhase()	Modulates 2 bits into a QPSK phase
Math_QPSK_2BitToPhase Array()	Modulates an array of bits into an array pg QPSK phases
Math_IntToBit()	Converts a integer value into a 8-bit binary array
Math_IntToCustomBit()	Converts a integer value into a N-bit binary array
Math_Round()	Rounds a float value into an integer
Math_Reverse_Array()	Returns the inverse of the input float-type array
Conjugate_Array()	Returns the negative of the input float-type array
Math_FloatTo2Byte()	Convert a 16-bit float-type value into 2 bytes
Math_2ByteToFloat()	Convert 2 bytes into a 16-bit float-type value
Math_AbsFloat()	Returns the absolute of a float-type value
Math_Phase2Quad	Returns the quadrant the input phase is located in.
Math_AmountOfDifferences InArrays()	Returns the amount of differences between two Integer-type arrays
Math_DistanceToZero()	Unwraps and calculates the distance from the input phase to zero

Library Name	OFDM_math
Function Name	Function Description
Math_DistanceTo180()	Unwraps and calculates the distance from the input phase to 180 degrees

Table C.7: The OFDM Math library functions

C. 8 The OFDM Transmitter Library Functions

The OFDM transmitter library functions are shown in Table C.8.

Library Name	OFDM_transmitter
Function Name	Function Description
OFDM_Transmitter_Init_Image_Transmitter	Initiates the OFDM transmitter structure for the transmission of an test image file
OFDM_Transmitter_Init_Data_Transmitter	Initiates the OFDM transmitter structure for the transmission of a test data packet
OFDM_Transmitter_GetTransmitFilename	Retrieve the transmitter filename using a file dialog box
OFDM_Transmitter_Load_Transmit_File	Load the transmitter file
OFDM_Transmitter_Add_Preambles	Adds the IEEE 802.11a preambles to the transmission buffer
OFDM_Transmitter_DataPhases_To_Complete_Symbol	A function that converts data phases into complete IEEE 802.11a OFDM data symbols
OFDM_Transmitter_Add_Signal	Adds the IEEE 802.11a signal symbol to the transmission buffer
OFDM_Transmitter_Encode	Encodes the transmission data into IEEE 802.11a OFDM data symbols
OFDM_Transmitter_Add_Data	Adds the encoded OFDM data symbols to the transmission buffer

Library Name	OFDM_transmitter
Function Name	Function Description
OFDM_Transmitter_Save_Transmission_To_Custom_File	Saves the transmission buffer to a local file
OFDM_Transmitter_Add_Noise	Adds AWGN to the transmission buffer

Table C.8: The OFDM Transmitter library functions

C. 9 The OFDM Receiver Library Functions

The OFDM receiver library functions are shown in Table C.9.

Library Name	OFDM_receiver
Function Name	Function Description
Receiver_WB	Writes a Byte-type buffer to file (debugging)
Receiver_WF	Writes a float-type buffer to file (debugging)
OFDM_Receiver_UpdateImage	Updates the received and decoded image on the GUI
OFDM_Receiver_Init_Setup_Variables	Receiver structure initialisations: Setup all the variables used in the receiver
OFDM_Receiver_Init_Create_Buffers	Receiver structure initialisations: Create all the buffers used in the receiver
OFDM_Receiver_Init_Clear_Buffers	Receiver structure initialisations: Clear all the buffers used in the receiver
OFDM_Receiver_Init_Fill_Buffers	Receiver structure initialisations: Fill all the buffers used in the receiver
OFDM_Receiver_Init_Image_Receiver	Receiver structure initialisations: Initialise the receiver to receive test images
OFDM_Receiver_Init_Data_Receiver	Receiver structure initialisations: Initialise the receiver to receive test data packets

Library Name	OFDM_receiver
Function Name	Function Description
OFDM_Receiver_State3	OFDM decoder state machine state 3 (Information extraction from long preamble and signal symbol)
OFDM_Receiver_State2	OFDM decoder state machine state 2 (Initial OFDM symbol synchronisation)
OFDM_Receiver_State1	OFDM decoder state machine state 1 (Signal threshold detector)
OFDM_Receiver_Create_Freq_Shifts	Create phase shifts from different sub-sample offsets
OFDM_Receiver_Add_Freq_Shifts	Add the received reference carriers to the phase shifts
OFDM_Receiver_Reference_Freq_Shifts	Reference all the sub-sample influenced phases to zero degrees
OFDM_Receiver_Calc_Smallest_Distances	Determine which sub-sample influenced phase performs the best and returns the result
OFDM_Receiver_Calculate_SubSample_Offset	Container function which calculates the best sub-sample offset for a given reference carriers and sub-sample offset range
OFDM_Receiver_State4	OFDM decoder state machine state 4 (Data Symbol Decoding)
OFDM_Receiver_Decode_Buffer	OFDM decoder state machine
OFDM_Receiver_GetReceiveFilename	Function which returns the receiver file filename using a file dialog box in the demo GUI
OFDM_Receiver_LoadTransmission_From_CustomFile	Loads the transmission data from a custom data file
OFDM_Receiver_LoadTransmissionSize_From_CustomFile	Loads the transmission data size from a custom data file

Library Name	OFDM_receiver
Function Name	Function Description
OFDM_Receiver_LoadTransmission_From_WaveFile	Loads the transmission data from a wave file
OFDM_Receiver_LoadTransmissionSize_From_WaveFile	Loads the transmission data size from a wave file
OFDM_Receiver_StartDecode_From_File	Container function, starts the OFDM decoding after data has been loaded from file
OFDM_Receiver_StartDecode_From_SoundBuffer	Container function, starts the OFDM decoding after data has been received from the sound device

Table C.9: The OFDM Receiver library functions

C.10 The OFDM Comparer Library Functions

The OFDM comparer library functions are shown in Table C.10.

Library Name	OFDM_comparer
Function Name	Function Description
OFDM_Comparer_GetCompareFilename	Retrieves the comparative filename using a file dialog box in the demo GUI
OFDM_Comparer_LoadData_File	Loads the comparative data if it's a test data packet
OFDM_Comparer_LoadCompareFile	Loads the comparative data if it's a test image
OFDM_Comparer_Compare	Compare the received data to the comparative data

Table C.10: The OFDM Comparer library functions

C.11 The OFDM Audio Library Functions

The OFDM audio library functions are shown in Table C.11.

Library Name	OFDM_audio
Function Name	Function Description
Audio_Initiate_Audio	Initiates the OFDM audio structure and audio device
Audio_Change_Audio_Device_With_Combobox	Change the currently selected audio device using the combo box on the GUI
Audio_Data_Transmission_Callback	Callback function which transmits signals over the sound device
Audio_Play_Transmission	Container function which plays back transmissions over the sound device
Audio_Play_Calibration	Transmits full volume sine wave to calibrate the volume controls
Audio_Data_Record_Callback	Callback function which records signals from the sound device
Audio_Record_Calibration	Records the calibration signals and calls the evaluations function
Audio_Record_Calibration_Evaluation	Evaluates the received calibration signal
Audio_Record_Transmission	Records incoming data transmissions

Table C.11: The OFDM Audio library functions

Appendix D

SDR OFDM Included CD

D.1 SDR OFDM Included CD

The CD included with this thesis contains the following:

- The complete IEEE 802.11a transceiver C++ source code.
- The complete IEEE 802.11a GUI Demo program.
- Some Matlab OFDM debugging programs.
- All the test images used in the performance testing.
- All the test data packets used in the performance testing.
- This thesis.
- The IEEE 802.11a specifications.
- Other useful OFDM related documentation.

Bibliography

- [1] IEEE Std 802.11a-1999 (supplement to IEEE Std 802.11a-1999, “Part 11: Wireless LAN medium access control (MAC) and physical layer PHY specifications: High speed physical layer in 5 GHz band,” temp. rep., IEEE, Sept. 1999.
- [2] Jung-yeol Oh, Jae-sang Cha, Seong-kweon Kim and Myoung-seob Lim. “Implementation of Orthogonal Frequency Division Multiplexing Modem Using Radix-N Pipeline Fast Fourier Transform (FFT) Processor,” *Jpn. J. Appl. Phys.* Vol.42 (2003) pp.1-6 Part 1, No 4B, April 2003.
- [3] Guillermo Acosta. “OFDM Simulations Using Matlab,” Smart Antenna Research Laboratory report, August 2000.
- [4] Yun Chiu, Dejan Markovic, Haiyun Tang and Ning Zhang. “OFDM Receiver Design,” EE225C, Final Report, 12 December 2000
- [5] Dr. Jean. Armstrong. “OFDM – Orthogonal Frequency Division Multiplexing,” presentation notes, Department of Electronic Engineering, La Trobe University.
- [6] Greg DesBrisay. “Basics of Orthogonal Frequency Division Multiplexing,” presentation notes, Cisco Systems, Inc., 2000
- [7] V. Moeyaert, P. Megret, J. Friodure, L. Robette and M. Blondel. “Analytical formulation of the error probability of a QPSK transmission impaired by the joint action of gaussian and impulse noises,” Service d’Electromagnetisme & Telecommunications, Faculte Polytechnique de Mons, MULTITEL,
- [8] John. G. Proakis, Masoud Salehi. “Communications Systems Engineering,” Prentice Hall, 1994
- [9] Seymour Stein, J. Jay. Jones. “Modern communications principles, with application to digital signalling,” McGraw-Hill, 1967
- [10] Peyton Z. Peebles, JR. “Probability, Random Variables and Random Signal Principles,” Third Edition, McGraw-Hill International Editions, 1993
- [11] R. E. Ziemer, W.H. Tranter. “Principles of Communications, Systems, Modulation, and Noise,” Fourth Edition, John Wiley & Sons, Inc., 1995

Bibliography

- [12] J.J. van de Beek, P. Odling, S.K. Wilson, P.O. Borjesson. "Orthogonal Frequency-Division Multiplexing (OFDM)," The International Union of Radio Science (URSI), Lulea University of Technology, 2002
- [13] Dr. R. Wolhuter. Class notes from the Telecommunications 823 postgraduate course, Department of Electrical and Electronic Engineering, University of Stellenbosch.
- [14] Edward Snyder. "Convolutional Encoding," Apogee Labs, Inc.
- [15] Haiyun Tang, Kan Y. Lau, and Robert W. Brodersen. "Synchronisation Schemes for Packet OFDM Systems," Berkeley Wireless Research Centre
- [16] V. S. Abhayawardhana, I. J. Wassell. "Residual Frequency Offset Correction for Coherently Modulated OFDM Systems in Wireless Communication," Laboratory for Communications Engineering, Department of Engineering, University of Cambridge
- [17] Jan-Jaap van de Beek, Per Ola Borjesson, Marie-Laure Boucheret, Daniel Landstrom, Julia Martinez Arenas, Per Odling, Christer Ostberg, Mattias Wahlqvist and Sarah Kate Wilson. "A Time and Frequency Synchronisation Scheme for Multiuser OFDM," IEEE Journal on Selected Areas in Communications, Vol. 17, No. 11, November 1999
- [18] Suhas N. Diggavi. "On achievable performance of spatial diversity fading channels," AT&T Shannon Laboratories, New Jersey, USA
- [19] A.D.S Jayalath and C. Tellambura. "Peak-to-Average Power ratio of IEEE 802.11a PHY layer Signals," School of Computer Science and Software Engineering, Monash University.
- [20] Curt Schurgers and Mani. B. Srivastava. "A Systematic Approach to Peak-Average Power Ratio in OFDM," Electrical Engineering Department, University of California at Los Angeles.
- [21] Lawrey, E., Kikkert, C.J., "Peak to average power ratio reduction of OFDM signals using peak reduction carriers," Fifth International Symposium on Signal Processing and its Applications, ISSPA, August 1999

Bibliography

- [22] Yngvar Larsen, Geert Leus and Georgios B. Giannakis. “Reduction of Peak-To-Average Power Ratio In Block Differential OFDM Systems,” Dept. of Pysics, University of Tromso, Norway.
- [23] Sinem Coleri, Mustafa Ergen, Anuj Puri and Ahmad Bahai. “A Study of Channel Estimation in OFDM Systems,” University of California, Berkeley.
- [24] L. Vandendorpe, B. Devillers, J. Duplicy, J. Louveaux. “ OFDM: Orthogonal Frequency Division Multiplexing,” Communications and Remote Sensing Lab, Universite Catholique de Louvain, Belgium.
- [25] Markku Juntti and Juha Ylitalo. “Spatial Multiplexing,” Tutorial – Mimo Communications with Applications to (B)3G and 4G Systems, University of OULU.
- [26] Marius Oltean, Andy Vesa, Eugen Marza. “Performance Evaluation of Single-Carrier Broadband Transmission with Frequency Domain Equalizers,” Transactions of Electronics and Communications, Buletinul Științific al Universității “Politehnica” din Timișoara, Tom 49(63), Fascicola 1-2, 2004.