TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Implementation of 2x1 Alamouti Space-Time Coded OFDM System

Diploma Thesis

executed for the purpose of obtaining the academic degree graduate Engineering

submitted by:
Emra Tairi

Student ID
0527228

Institute of Telecommunication
Submitted to Vienna University of Technology

under the supervision of:
Univ.Prof. Dipl.-Ing. Dr.techn. Christoph F. Mecklenbräuker
Dipl.-Ing. (FH) Georg Maier

submitted on:
12. 10. 2016

Erklärung:

Ich erkläre, dass die vorliegende Diplomarbeit/Masterarbeit von mir selbst verfasst wurde und ich keine anderen als die angeführten Behelfe verwendet bzw. mich auch sonst keiner unerlaubter Hilfe bedient habe.

Ich versichere, dass ich diese Diplomarbeit/Masterarbeit bisher weder im In- noch im Ausland (einer Beurteilerin/einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Weiters versichere ich, dass die von mir eingereichten Exemplare (ausgedruckt und elektronisch) identisch sind.

Datum: ..... Unterschrift: .....

Acknowledgements

This thesis has been written for the completion of the master study of Communications Engineering at the Technical University of Vienna.

First of all I would like to express my gratitude towards my University, Technical University of Vienna, for a wonderful experience during the past years.

I am deeply grateful to Prof. Christoph Mecklenbräuker for giving me the opportunity to work under his supervision and learn from his experience during this thesis. I enjoyed working with him. Also his good ideas and helpful suggestions are gratefully acknowledged. I am honored working with him.

Further, I wish to express my gratitude to my supervisor Georg Maier, who provided me constant support and guidance through the entire thesis. He has supported me with kindness, patience and friendly assistance throughout my thesis. His constant advice resulted in many useful suggestions that helped a lot to improve this master thesis with regard to both technical content and presentation.

I will never forget my colleagues and all the great moments we spent together throughout our studies at the University. I am deeply thankful to all my colleagues for always being there for me.

Big thanks to my family for patiently motivating me and helping me to put all my effort in this work. I can never thank them enough for their consistent encouragement, understanding, love, friendship and support.

Abstract

The rapid development of wireless communication enabled its use in many fields of technology. Also in vehicular environment to reduce the number of traffic accidents, to provide high data rate information as well as to enable entertainment and comfort applications for drivers and passengers wireless communication systems are used. Therefor IEEE 802.11p standard has been developed for dealing with vehicular communication, which is a protocol that adds wireless access in vehicular environment. It enables data exchange between the vehicles and between the vehicles and the roadside infrastructure. The standard specifies a Physical Layer (PHY), which is based on Orthogonal Frequency Division Multiplexing (OFDM). This thesis gives an overview of the IEEE 802.11p standard's Physical Layer (PHY) together with the methodology of the OFDM and the Alamouti Space-Time Code.

Especially a closer look has been achieved at the OFDM technology by investigating its major benefits and the main obstacles that come with it. The performance and reliability of OFDM system is calculated in Fixed-Point algorithm, analyzed through C code-based simulations and plotted on Matlab. Overall system is demonstrated through numerical simulations on PC and the performance is evaluated in AWGN channel. The results are plotted on graphs and also discussed and compared with the theoretical simulation results. The analyses highlight that the performance of OFDM system can be increased with 2x1 Alamouti Space-Time Code.

After a brief introduction, the entire communication system is implemented together with some design parameters. The first part of the thesis deals with the implementation of OFDM on the single-input single-output (SISO) system. The second major part analyses Alamouti Space-Time Coded OFDM on 2x1 (MISO, 2 transmitter and 1 receiver) system.

Kurzfassung

Die schnelle Entwicklung der drahtlosen Kommunikation ermöglicht in vielen Bereichen der Technik eingesetzt werden. Auch in Fahrzeugumgebung die Anzahl der Verkehrsunfälle zu reduzieren, die hohe Datenrate an Informationen zu ermöglichen als auch Unterhaltung und Komfortanwendungen für Fahrer und Passagiere zu ermöglichen, werden drahtlose Kommunikationssysteme verwendet. IEEE 802.11p Standard ist für den Umgang mit Fahrzeug-Kommunikation entwickelt worden. Es ist ein Protokoll, welches drahtlosen Zugang in Fahrzeugumgebung hinzufügt. IEEE 802.11p Standard ermöglicht den Datenaustausch zwischen Fahrzeugen und zwischen Fahrzeugen und der Straßeninfrastruktur. IEEE 802.11p Standard spezifiziert physikalischen Schicht (PHY) basierend auf das Orthogonales Frequenzmultiplexverfahren (OFDM).

Diese Diplomarbeit gibt einen Überblick über den physikalischen Schicht (PHY) des IEEE 802.11p Standard zusammen mit der Methodik des OFDM und des Alamouti Raum-Zeit-Kode. Besonders einen genaueren Blick auf die OFDM-Technologie durch die Untersuchung ihrer großen Vorteile und die wichtigsten Hindernisse erreicht wird. Die Leistung und Zuverlässigkeit von OFDM-System wird durch Fixpunkt Algorithmus berechnet, durch C-Kode-basierte Simulationen analysiert und im Matlab aufgetragen. Das Gesamtsystem wird durch numerische Simulationen auf PC demonstriert und die Leistung des Systems wird in AWGN-Kanal ausgewertet. Die Ergebnisse werden in Diagrammen aufgetragen und mit den theoretischen Simulationsergebnissen verglichen und diskutiert. In dieser Diplomarbeit zeigen die Analysen, dass die Leistung des OFDM-Systems mit 2x1 Alamouti Raum-Zeit-Kode erhöht werden kann.

Nach einer kurzen Einführung, beschreibt diese Diplomarbeit die Umsetzung des gesamten Kommunikationssystems zusammen mit Fehlerwahrscheinlichkeit und einige Design-Parameter. Der erste Teil der Diplomarbeit befasst die Umsetzung der OFDM auf dem SISO-System. Der zweite wesentliche Teil dieser Diplomarbeit ist Alamouti Raum-Zeit-Kodierte OFDM auf 2x1 (MISO, 2 Sender und 1 Empfänger) System.

Acronyms and Abbreviations

AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
CP	Cyclic Prefix
DAB	Digital Audio Broadcasting
DSL	Digital Subscriber Line
DSP	Digital Signal Processing
DFT	Discrete Fourier Transformation
DVB	Digital Video Broadcasting
FEC	Forward Error Correction
FFT	Fast Fourier Transformation
FWL	Fractional Word Length
ICI	Inter Carrier Interface
IDE	Integrated Development Environment
IDFT	Inverse Discrete Fourier Transformation
IEEE	Institute of Electrical and Electronics Engineers
IFFT	Inverse Fast Fourier Transformation
ISI	Inter Symbol Interface
ITS	Intelligent Transport Systems
IWL	Length of Words Integer Part
3G	Third Generation
GI	Guard Interval
LAN	Local Area Network
LFRS	Linear Feedback Shift Register
LMSE	Least Mean Square Estimation
LS	Least Square
LTE	Long Term Evolution
MAC	Medium Access Control
MISO	Multiple-Input Single-Output
MIMO	Multiple-Input Multiple-Output
MRRC	Maximum Ratio Receiver Combining
MPDU	Medium Access Control Protocol Data Unit
OFDM	Orthogonal Frequency Division Multiplexing
OSI	Open System Interconnection
PARP	Peak To Average Power Ratio

PHY	Physical Layer
PLCP	Physical Layer Convergence Protocol
PMD	Physical Medium Dependent
PPDU	Physical Protocol Data Unit
PSDU	PLCP Service Data Unit
QPSK	Quadrature Phase Shift Keying
SISO	Single-Input Single-Output
SNR	Signal-to-Noise-Ratio
STC	Space-Time Coding
STBC	Space-Time Block Code
STTC	Space-Time Trellis Code
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VA	Viterbi Algorithm
WAVE	Wireless Access in Vehicular Environments
WL	Total Length of the Word
WLAN	Wireless Local Area Network
WIFI	Wireless Fidelity
WIMAX	Worldwide Interoperability for Microware Access
WMAN	Wireless Metropolitan Area Network
ZF	Zero-Forcing

Keywords

Alamouti Space-Time Code
AWGN Channel
Channel Estimator
Convolutional Encoder
Cyclic Prefix
Error Ratio
Fast Fourier Transform
Fixed-Point Algorithm
Fixed-Point FFT
IEEE 802.11p
Interleaver
Multiple-Input Multiple-Output
Orthogonal Frequency Division Multiplexing
OFDM Symbol Assembler
PPDU Frame
Scrambler
Subcarrier
Twiddle Factor
Viterbi Decoder
Zero-Forcing Equalizer

Contents

1 Introduction

The increasing number of vehicles leads to more traffic jams and a higher risk of road accidents. Some technologies like seat belts, airbags, anti-lock braking systems, radar, cameras and sensors have been implemented in vehicles in order to reduce traffic jams and road accidents [1]. Especially while talking about car communication there are Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) wireless communication applications, which are used to provide real time communication network for drivers and for higher level traffic management systems. In V2V communication vehicles establish connections between each other. In V2I some infrastructure is present from which the vehicles retrieve information.

Applications of vehicular communications are traffic safety applications, traffic efficiency applications and commercial applications, which are used to avoid accidents. Traffic efficiency applications are speed limit information, route guidance, green light optimal speed information and variable traffic light phases. The commercial applications are local advertising, Internet access, chasing vehicles, parking management and point of interest notification. And lastly the traffic safety applications are collision avoidance, wrong-way driving warnings, and hazardous location notification. Safety applications require reliability and strict short predictable deadlines for the delay, while commercial applications require high data rates.

OFDM, with its high rate transmission capability and high bandwidth efficiency, is applied in vehicular communication systems. OFDM signals are generally created by the inverse fast Fourier transform (IFFT) on the transmitter site and demodulated by the fast Fourier transform (FFT) on the receiver site.

This thesis examines the effect of OFDM as a modulation technique for IEEE 802.11p standard. The main goal of this thesis is to develop, learn, and understand an OFDM 802.11p PHY layer baseband implementation. The most important building blocks in the development of this thesis are OFDM modulation with Fast Fourier Transform (FFT), and Alamouti Space-Time Code (STC). Entire communication system is numerically calculated with Fixed-Point algorithm, which enables better understanding about the usage of Fixed-Point algorithms.

This chapter provides a theoretical background about standardization, basics of OFDM, and Space-Time Codes together with multiple-input multiple-output (MIMO) systems.

1.1 Basics of the OFDM Transmission Technique

OFDM is a modulation technique that places the information on multiple carrier frequencies. As shown in the Fig. 1 it is a multicarrier modulation technique and is used for high-speed digital communications.

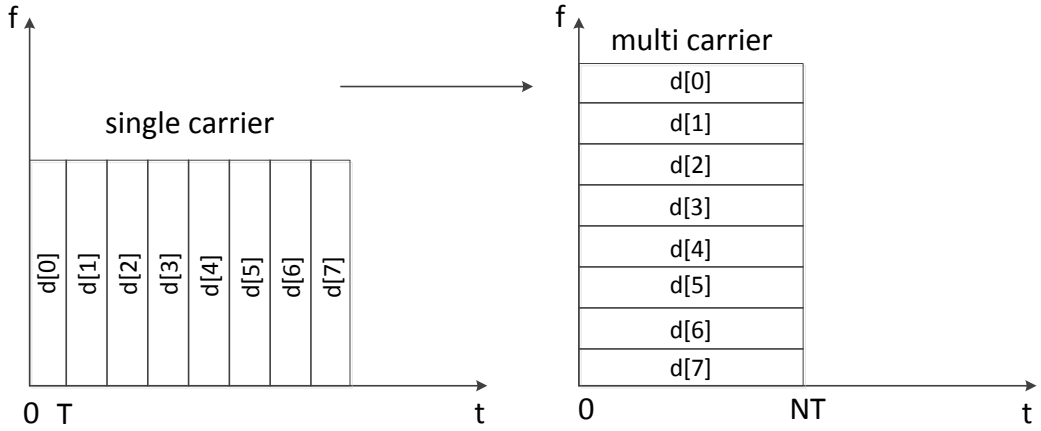

Fig. 1: Difference between single-carrier systems

OFDM performs excellently over frequency selective channels. The signal itself is split into independent channels, each using a fraction of the available bandwidth. Each of independent channels are called subcarriers, where all together form the OFDM carrier. Transport data is modulated in the amplitude and phase of the signal. *N* parallel streams are transmitted by modulating *N* distinct carriers. The symbol duration is enlarged *N* times and the bandwidth consumption of each symbol is reduced by the same factor *N*. The overall data rate and bandwidth consumption is kept constant through parallel transmission over *N* independent subcarriers. Although the spectra of different modulated carriers overlap, the subcarriers will not interfere with each other. Due to orthogonality a tight spacing of the subcarriers becomes possible enabling an efficient use of the available bandwidth *B*. In the Fig. 2 OFDM transmission technique in the frequency domain is shown.

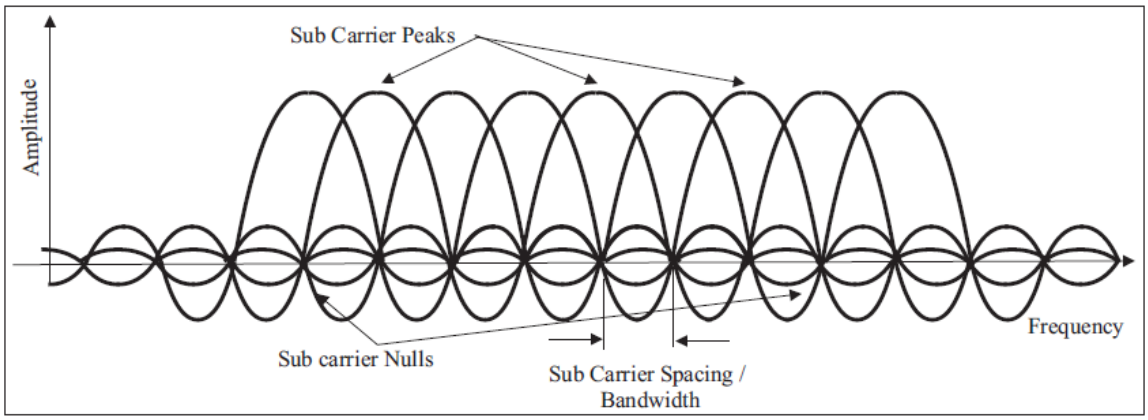

Fig. 2: Orthogonal basis function in an OFDM

N adjacent and orthogonal subcarriers are spaced by the frequency distance Δf on the frequency axis. Orthogonality between all subcarrier signals exists, if the subcarrier distance and the symbol duration are chosen such that $T_s = \frac{1}{\Delta f}$. Since the system bandwidth *B* is subdivided into *N* narrowband sub-channels, the OFDM symbol duration T_s is *N* times larger than in the case of an alternative sub-channel transmission system covering the same bandwidth *B* [2].

Typically the number of subcarriers is chosen in such a way that the symbol duration T_s is sufficiently large compared to the maximum multi-path delay τ_{max} of the radio channel. Additionally, delay dispersion in frequency-selective channels exists, which leads to appreciable errors even when $TF \geq 1$, and also leads to a loss of orthogonality between the subcarriers and thus the ICI (inter carrier interference) [2]. In order to maintain orthogonality a special type of guard interval called a cyclic prefix is added. This is done by taking the *M* last symbols of the frame and putting a copy of them at the beginning of the frame. This also makes the output from the IFFT periodic. Cyclic prefix serves as a buffer between consecutive OFDM frames. It is also mentioned in section 2.1.1.7.

OFDM symbol duration with cyclic prefix is $T_{s'} = T_s + T_{CP}$, where during $-T_{CP} < t < 0$ a copy of the last part of the symbol is transmitted. At the transmission, data is transformed into time-domain using IFFT. At the receiver the received time-domain signal is transformed back to the frequency domain using FFT. The number of points of the IFFT and FFT is usually of radix two. The 802.11p standard uses a 64-point FFT. The discrete-time representation of the signal after IFFT is given by Equation (1.1)

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k) e^{i2\pi\frac{k}{N}n} \quad (1.1)$$

where *N* is the total number of subcarriers and $n \in [0, N-1)$. At the receiver the data is recovered by performing FFT on the received signal, which is given by Equation (1.2)

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-i2\pi\frac{n}{N}k} \quad (1.2)$$

where $k \in [0, N-1)$.

The received signal is represented by the convolution of the transmitted time signal with the channel impulse response *h(t)* and an Additive White Gaussian Noise term given by Equation (1.3).

$$r_n(t) = s_n(t) * h_n(t) + n_n(t) \quad (1.3)$$

After demodulation (FFT) cyclic convolution corresponds to multiplicative input-output relation given by Equation (1.4).

$$r_k[n] = H_k \cdot s_k[n] + n_k[n]. \quad (1.4)$$

This shows that there is no interference between the sub channels and no inter-channel interference.

Advantages of the OFDM:

- OFDM transmits simple constellation at low symbol rate.
- OFDM systems provide high spectral efficiency.
- Each transmitted data stream occupies a very narrow frequency band.
- ISI is less of a problem with OFDM because low data rates are carried by each carrier.
- OFDM offers frequency diversity by spreading the carriers all over the used spectrum.
- Transmitter and receiver can be implemented using an IFFT and FFT respectively.
- If the channel state is known the transmitter can decide on the correct transmission parameters for each subcarrier, namely, coding rate and modulation order. Also different power levels per subcarrier can be assigned according to the channel quality.
- OFDM needs simple equalizer at the receiver due to the long symbol duration and the use of CP.

Disadvantages of the OFDM:

- OFDM is not power efficient, due to linearity power amplifiers are required.
- OFDM is sensitive to Doppler shift-frequency error offset
- OFDM is also sensitive to frequency time issues.
- Cyclic prefix lowers overall spectral efficiency
- High Peak to average power ratio (PAPR) of the transmitted signal.
- High time-frequency synchronization at the receiver.
- The choice of small values of GI can introduce ISI and ICI thereby destroying the orthogonality.
- The choice of small values of subcarrier bandwidth can cause carrier offset errors.
- Requires complex electronics, like DSP including FFT algorithms, to run the software.

1.2 MIMO Systems

The capacity of wireless communication systems is increased with multiple transmit and receive antennas (MIMO) technology. MIMO systems use multiple transmit and multiple receive antennas for a single user providing much higher spectral efficiency than SISO systems. MIMO systems reduce *BER* or increase the transmission quality. Multiple-antenna technique simply improves reliability and increases transmission data-rate without any increase of transmission bandwidth. Especially the use of OFDM combined with MIMO technology improves system performance remarkably.

Let us consider MIMO system with j transmit and i receive antennas. The block diagram is given in Fig 3.

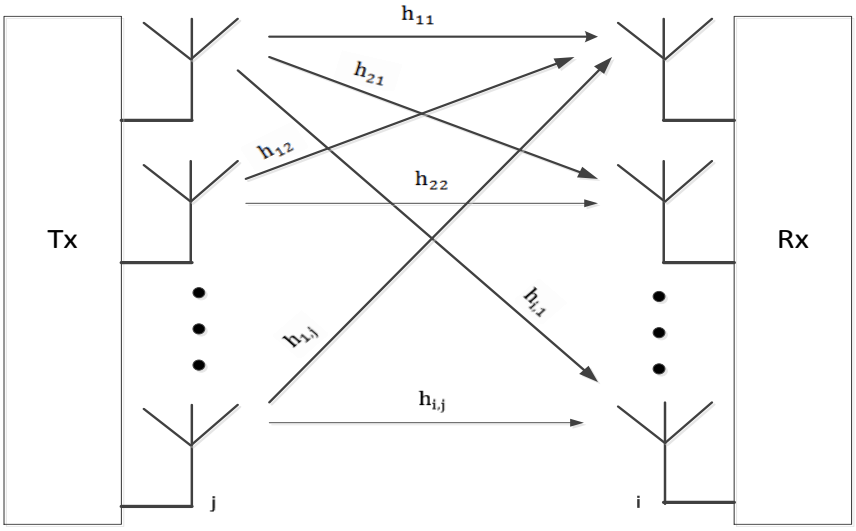

Fig. 3: MIMO System

The $h_{i,j}$ is a complex number corresponding to the channel gain between transmit antenna *j* and receive antenna *i*. If at a certain time instant complex signals s_j are transmitted then the receive signal can be expressed as in Equation (1.5)

$$r_i = \sum_{j=1}^{N} h_{i,j} \cdot s_j + n_i \quad (1.5)$$

where n_i is a noise term.

We can also combine all receive signals in an $i \times 1$ vector *r*, as shown in Equation (1.6)

$$r = H \cdot s + n \quad (1.6)$$

where *s* is the j × *1* transmit symbol vector, *n* is the $i \times 1$ additive noise vector and *H* is the $i \times j$ MIMO channel transfer matrix, given by Equation (1.7).

$$H = \begin{pmatrix} h_{11} & \cdots & h_{1j} \\ \vdots & \ddots & \vdots \\ h_{i1} & \cdots & h_{ij} \end{pmatrix}. \quad (1.7)$$

In this thesis a Multiple-Input Single-Output (MISO) system is used, because in a 2x1 Alamouti Space-Time Coded System two antennas are employed only at the transmitter side.

Due to diversity the different transmission channels carry independently fading copies of the same signal, so the correlation between signals on different diversity branches should be zero for maximum diversity, which means the higher the diversity gain, the lower the probability of error. There exists time, frequency, and space or antenna diversity. Antenna or space diversity is classified into receive diversity and transmit diversity. In this thesis the 2x1 Alamouti Space-Time coded OFDM system transmitter diversity is used. In transmit diversity information is processed at the transmitter and then spread across the multiple antennas, in our case across two antennas. In this thesis also time and frequency diversity is implemented. Time diversity is introduced by the convolutional encoder and the interleaver whereas frequency diversity is introduced by spreading the bits over different OFDM subcarriers.

1.3 Space-Time Coding

In the case of multiple antennas at the transmitter side, the implementation of Space-Time Codes (STC) is required. STC reduce the fading effect in the wireless channel and improves the BER performance in receiver. Through the space-time coding, the information is spread across space and time, meaning a set of symbols is encoded in another set of symbols suitable for the transmission on spatially separated streams. Space time coding introduces redundancy between signals transmitted from various antennas (space) at various symbol periods (time). The goals of STC are simple decoding, minimization of error probability and the maximization of the information rate. A space-time coding has low complexity and high diversity. Simply space-time codes are powerful techniques to achieve full spatial diversity with low decoding complexity. There exist two types of space-time codes, namely space-time trellis codes (STTC) and space-time block codes (STBC). STBCs are mostly used with an error correction code, whereas STTCs are an extension of the trellis codes in the case of multi-antenna transmit system and the error correction code is not needed. This thesis focuses on Alamouti Code, which is a special type of space-time block code (STBC). Alamouti introduces a very simple scheme allowing transmissions from two antennas with the same data rate as on a single antenna but increasing the diversity at the receiver from one to two in a flat-fading channel [3].

In the case of two transmit antennas, Alamouti code promises full diversity and full data rate, that means orthogonality between the signal vectors transmitted over the two transmit antennas exists. In the section 2.2 details and characteristics of the Alamouti space-time codes are explained.

1.4 Introduction to IEEE 802.11p

Wireless Access in Vehicular Environments (WAVE) defines a complete layered protocol stack, which is also shown in Fig. 4. The Institute of Electrical and Electronics Engineers (IEEE) develops IEEE 802.11p protocol. IEEE 802.11p is a standard for vehicular networks, which work with the cooperation of the IEEE 1609 standard family. The IEEE 802.11p standard specifies an OFDM PHY that provides high speed wireless data transmission for minimizing data errors. The IEEE 802.11p standard is mainly based on the Wireless Local Area Network (WLAN) IEEE 802.11a standard.

Communication architectures are usually based on the layered OSI (Open Systems Interconnection)-model, where each level provides certain functions. WAVE upper layers support functions like data transfer, resource management, system configuration, and notification. IEEE 802.11p standard provides specifications for the lower layers like Physical (PHY) and Medium Access Control (MAC) layer. The MAC layer enables participants of the vehicular network to establish a network or join a pre-existing network performing functions like fragmentation, packet retransmissions, and acknowledgements.

This thesis deals with the physical layer characteristics. PHY layer in IEEE 802.11 standard is split into two sublayers the PMD (Physical Medium Dependent) sublayer and the PLCP (Physical Layer Convergence Protocol) sublayer. The PLCP sublayer defines a method of mapping the MAC sublayer MPDU (MAC Protocol Data Units) into a framing format suitable to the medium. The PLCP prepares MPDUs for transmission when the MAC layer instructs, delivers incoming frames from the wireless medium to the MAC layer, and also minimizes the dependence of the MAC layer on the PMD sublayer. The PLCP sublayer adds a header containing parameters of the physical transmission like signal, service, length, CRC and preamble, which allows synchronization between transmitter and receiver, to each frame coming from the MAC layer. Under the direction of the PLCP, the PMD provides actual transmission and reception of PHY entities between two stations through the wireless medium [4]. PMD sublayer defines all modulation and coding types. IEEE 802.11p protocol uses OFDM signaling with 48 data subcarriers within a 10 MHz bandwidth. The OFDM frame, which is the resulting frame on the PHY Physical Medium Dependent (PMD) consists of 4 OFDM symbols for the PLCP preamble. These symbols are 2 for a short and 2 for a long preamble, 1 OFDM symbol for signaling, and a variable number of OFDM symbols for data. The packet length N_{MSDU} and data rate determine the number of data OFDM symbols. Tab. 1 shows all main values used in this thesis for the PHYs implementation.

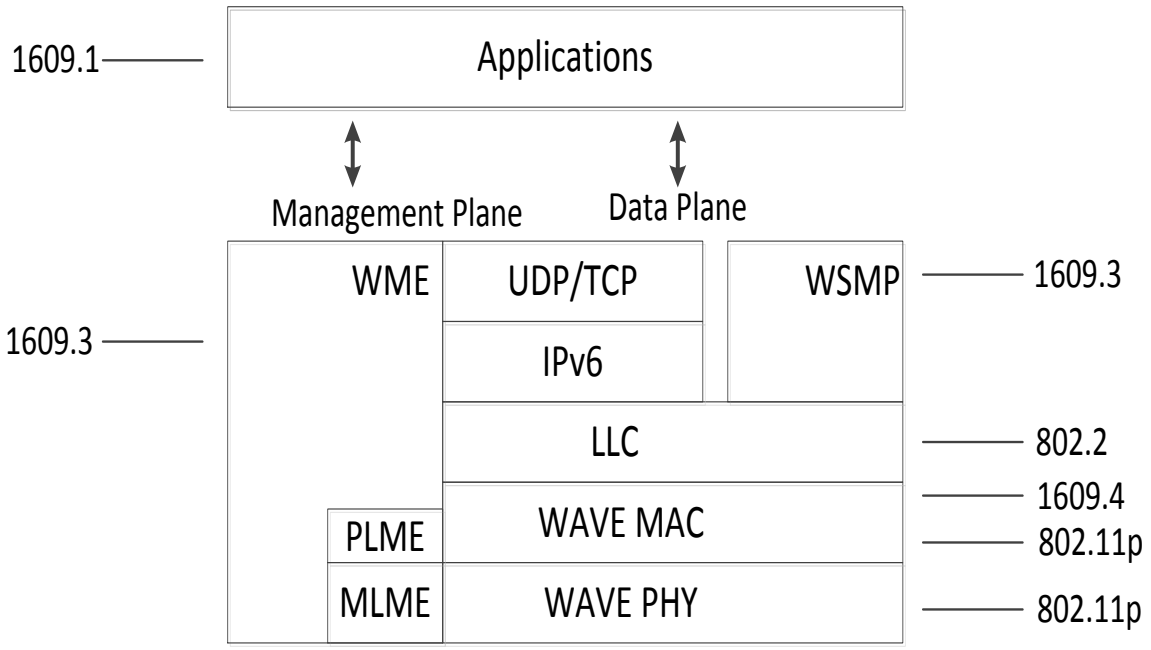

Fig. 4: Protocol Stack

Modulation	BPSK, QPSK
Data Rate (Mbits/s)	3, 6
Coded bits per subcarrier	1, 2
Data bits per OFDM Symbol	24, 48
Coded bits per OFDM Symbol	48, 96
Number of fft points	64
Number of sub carriers	52+DC
OFDM Symbol duration	8 µs
Cyclic prefix time (guard interval)	1.6 µs
FFT period	6.4 µs
Channel bandwidth	10 MHz
Subcarrier spacing	0.15625 MHz
Sampling period	100 ns
Alamouti space-time code rate	1/2

Tab. 1: Main system parameters in IEEE 802.11p standard

1.5 Scope

In this thesis OFDM system with 64 subcarriers is coded. OFDM system is also implemented in 16-bit fixed-point algorithm. At the transmitter bit streams are grouped and mapped into complex symbols. Channel is assumed to be AWGN channel. Convolutional encoder is applied to a block of data, resulting in the increases of transmission redundancy. At the receiver, Viterbi decoder is used in order to decode convolutional codes more efficiently. To improve the BER performance at the receiver also Equalizer and LMS Channel Estimation are used.

The goal of this thesis is the use of 2x1 (two transmit and one receive antenna) Alamouti Space-Time Coding and fixed-point algorithm. Especially Alamouti Coding increases diversity and enhances the transmission quality. This thesis will present the benefits of applying fixed-point algorithm and space-time coding schemes to vehicular communication-environments.

2. Implementation of Systems

In this chapter IEEE 802.11p PHY simulator is presented. For transmission an OFDM technique with 64 subcarriers is used. The transmitted signal is organized in frames. All submodules for 1x1 OFDM and 2x1 Alamouti Space-Time Coded OFDM communication systems are covered. Next chapter describes the Fixed-Point Algorithms, Fixed-Point FFT together with the important parts of the code. In chapter 4 the simulation results with plotted graphs are discussed. Conclusion, remarks and outlook are summed up in chapter 5.

2.1 1x1 OFDM System

Fig. 5 shows a block diagram of a point to point transmission system using OFDM, which is conceptually divided into three main modules: the transmitter, AWGN channel and the receiver. Each module is further split up into several submodules. This section gives a conceptual view of how each submodule is defined in the IEEE 802.11p standard.

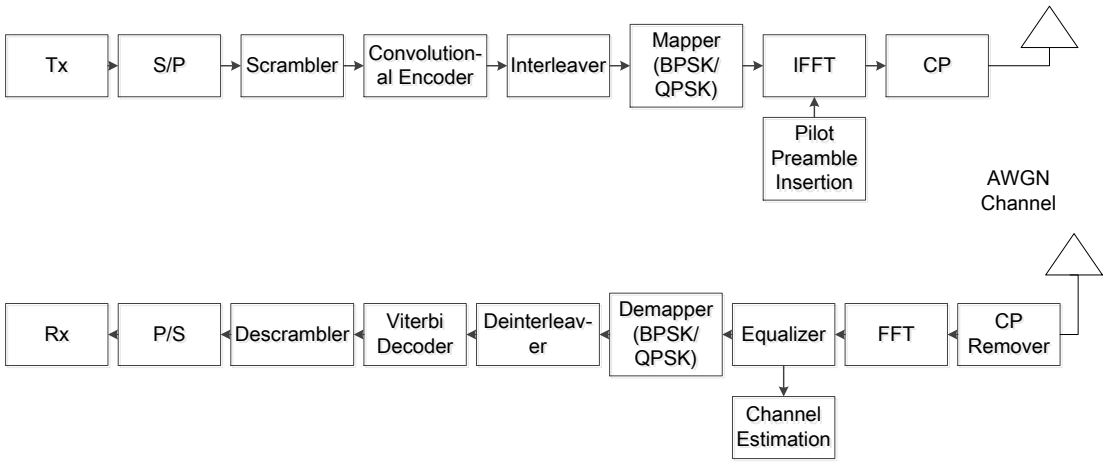

Fig. 5: Block diagram of a 1x1 OFDM System

2.1.1 Transmitter

Fig. 6 presents the main parts of the transmitter. First of all input data bits are randomly generated with equally likely ones and zeros. The input serial data bit stream is formatted into blocks of the size required for transmission. This data bit stream represents the information to be transmitted. The data is then transmitted in parallel by assigning each data word to one carrier in the transmission. The binary sequence from the source block is encoded. After the convolutional encoding some redundancy is introduced, which is used to combat the corruptive effect introduced by the channel. After convolutional encoder interleaving is used to map adjacent bits into non-adjacent OFDM subcarriers. Afterwards coded and interleaved data string is converted into a complex number

according to a certain signal constellation. This complex numbers are divided into groups of 48 and mapped to OFDM subcarriers. Next stage is assembling the symbols, which inserts 4 pilot subcarriers among 48 data subcarriers. Those 52 subcarriers together with the zero DC subcarrier are folded in 11 zero guard subcarriers to form an OFDM symbol. The comb pilot subcarriers and the two long training symbols are used to perform channel estimation. Short training sequences used for a coarse timing estimation do not appear on the figure because perfect time synchronization is assumed. The next block called IFFT (Inverse Fast Fourier Transformation) converts the complex-valued symbols on 64 subcarriers into a 64 tap complex discrete time signal for each OFDM symbol. This time domain signal consists of two long training sequences, signal field and data OFDM symbols. Signal field contains information about the modulation format, coding rate and frame length. Cyclic prefix, which is also called guard interval (GI) consists of the last 16 taps of the discrete time signal of the OFDM symbol, which is copied and placed on the front of each OFDM symbol. In the last step, PPDU frame is created with appended OFDM data symbols. Next subsection begins with the building blocks of the transmitter.

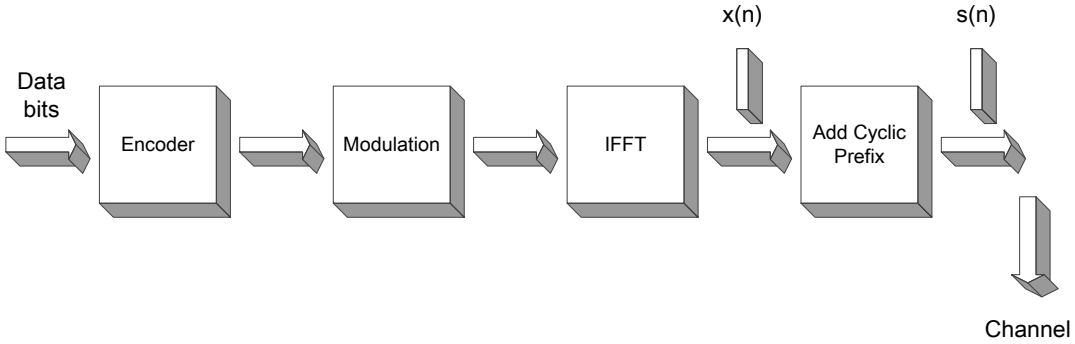

Fig. 6: Main parts of the transmitter

2.1.1.1 Source

Transmit signal is represented by a random binary sequence. Output values are binary and equally likely. All values which are greater than one half are mapped to “1” and smaller than one half to “0”. These random values represent the PSDU, the payload of the PPDU. The number of generated random bits N depends on coding rate *R*, number of data subcarriers N_{SD}, number of OFDM data symbols per frame N_{sym}, as well as on the number of bits corresponding to one constellation point N_{BPSC} which is given by the Equation (2.1):

$N = N_{SD} \cdot N_{sym} \cdot R \cdot N_{BPSC}$ [5]. (2.1)

2.1.1.2 Scrambler

Bitstream may contain long sequences of 1s or 0s, causing an energy flowing in the transmitter and receiver. To make sure that there is no long run of 1s or 0s in the bitstream and to disperse the energy across all of its bandwidth some randomization is needed. A scrambler also known as a randomizer is used to convert an input string into random output string at the same length. Scramblers are usually defined on linear feedback shift registers (LFSRs). The scrambler implemented in this thesis consists of 7 shift registers and 2 XORs as shown in the Fig. 7. The generator polynomial *S(x)* used by the frame synchronous scrambler is given by Equation (2.2):

$S(x) = x^7 + x^4 + 1$ [6]. (2.2)

which repeatedly generates a 127-bit sequence for a given pseudo-random initial state. Each incoming data bit is XORed with the current bit in the 127-bit sequence creating the output of scrambler.

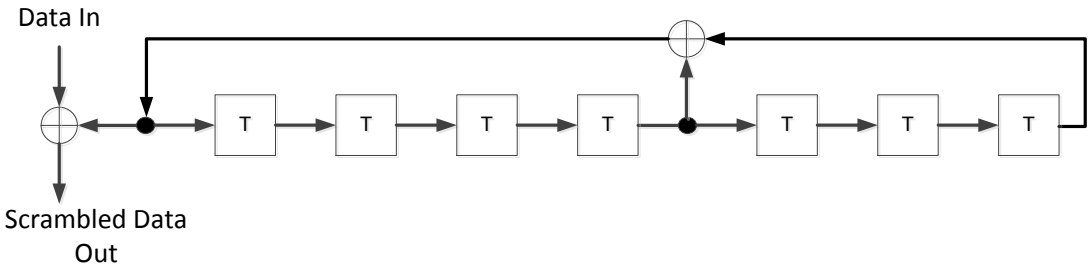

Fig. 7: Data Scrambler [6]

2.1.1.3 Encoder

The scrambled data passes through a convolutional encoder. In the convolutional encoder data sequence is first divided into long blocks and then encoded, because of that convolutional encoder requires very little buffering and storage hardware. Convolutional encoder adds correlation to the input data sequence by using delay elements and modulo adders, which can be implemented with a feed forward shift register and XOR gates. A convolutional encoder is described by three parameters (*n, k, m*) where *n* is the number of output bits, *k* is the number of input bits and *m* is the number of shift register stages of the encoder. The convolutional encoder accepts *k*-bit blocks of information sequence and produces an encoded sequence of *n*-bit blocks called codewords. However, each encoded block also depends on *M* previous blocks. As also shown in the Fig. 8 output of convolutional encoder is generated with one present input bit and six previous input bits. Constraint length *K* represents the number of bits in the encoder memory that affect the generation of the *n* output bits. The error correction capacity is also related with constraint length.

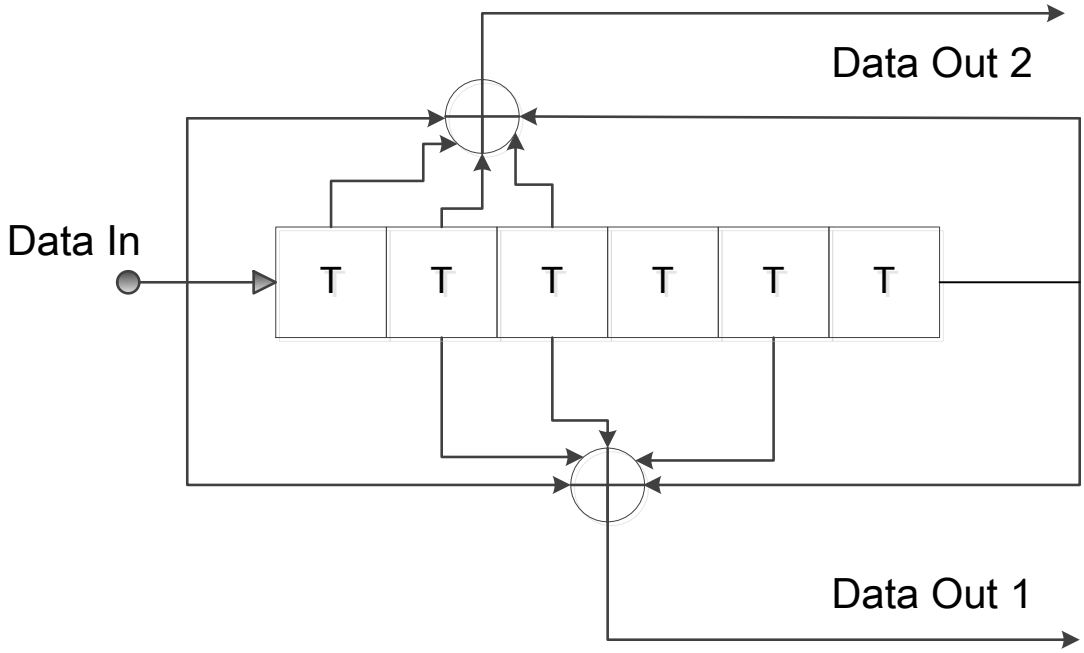

Fig. 8: Convolutional Encoder (K=7, r=1/2)

Code rate is another important parameter, which measures the code efficiency and is defined as *r= k/n*. Code rate of value 1/2 means at each time index the encoder takes one input information bit and produces two output code bits. An encoder with *N* memory elements has 2^N possible states and a *K=N+1* constraint length. Longer constraint length leads to more powerful code, more coding gain, more complex decoder and also more decoding delay. In this thesis IEEE 802.11p is implemented, which has a maximum size codeword packets of 2^{15} bits and 1/2 convolutional encoder (n=2, k=1), which means for one bit entering the encoder there are 2^1=2 possible branches. If constraint length K=7, then the size of shift register would be 6, which results in 2^6=64 possible states. Fig. 9 shows the state diagram of K=7 and r=1/2 convolutional encoder.

Also the generator polynomials g_0= $[01011011]_2$ and g_1= $[01111001]_2$ are important for representing the connection of the shift register taps to the modulo-2 adders. Redundancy introduced by the convolutional encoder is used for error correcting coding that allows the receiver to combat errors occurred by the channel.

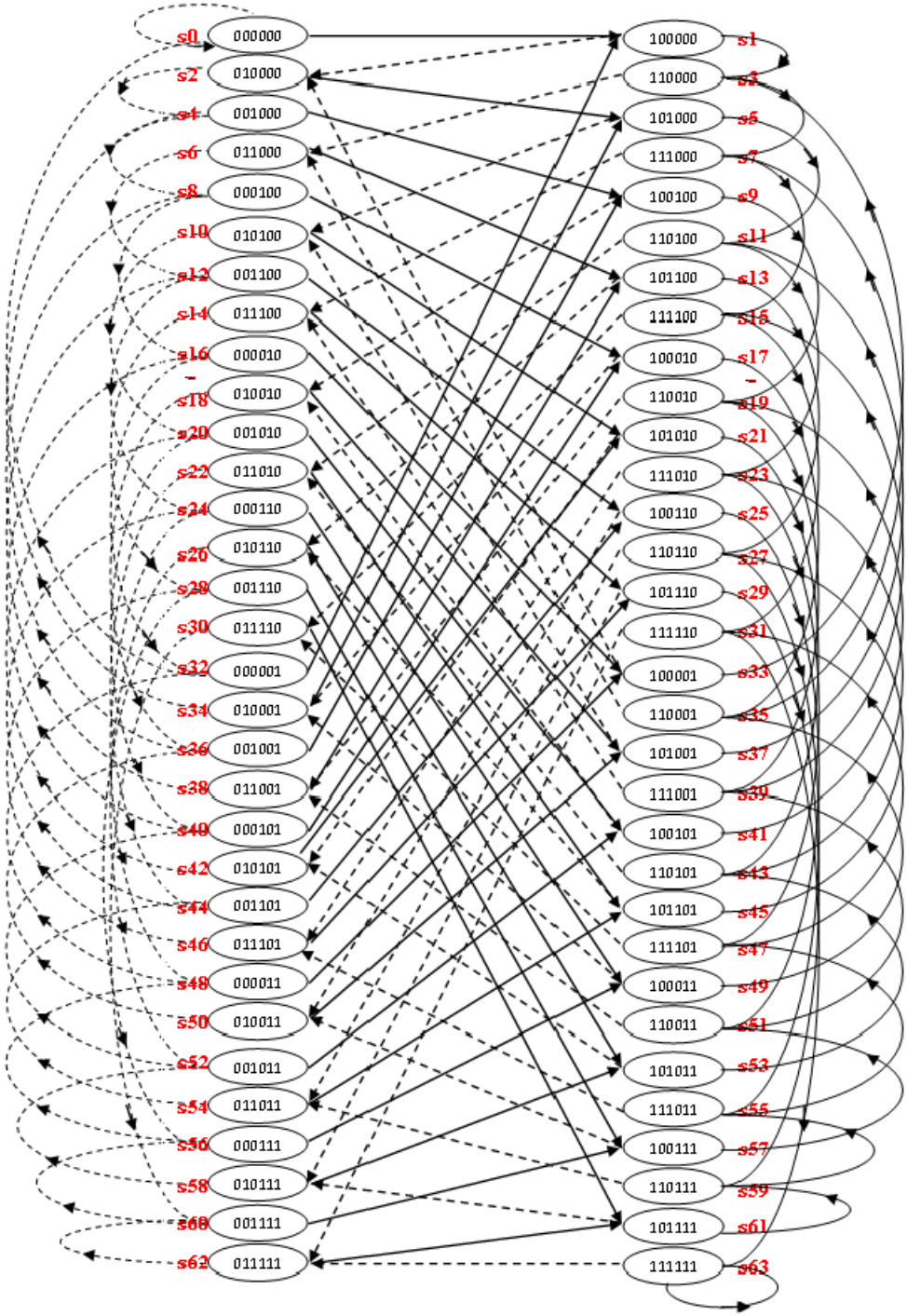

Fig. 9: State diagram of K=7, r=1/2 convolutional encoder [7]

2.1.1.4 Interleaver

The weakness of convolutional encoder is its susceptibility to burst error. This weakness can be avoided by using an interleaver, which is used for combating against burst error. Interleaver takes data packets, chops them up and then rearranges them. By this rearrangement the contiguous data is spaced further apart into a non-continuous stream. Interleaver aims to distribute transmitted bits in order to achieve desirable bit error distribution after demodulation. The kind of interleaving pattern depends on the channel characteristics. Interleaving reduces adjacent correlation in the bit stream. In the interleaver the encoded data bits are interleaved with the block size equal to the number of bits in a single OFDM symbol. In this thesis the interleaving depth, which defines the number of bits/bytes in each block of data, of *m*=16 is used.

The interleaver performs two consecutive permutations. The first one ensures that adjacent coded bits are mapped onto nonadjacent subcarriers. The second one ensures that adjacent coded bits are mapped alternately onto less and more significant bits of the constellation.

Let k=0,1,….N_{CBPS}-1 denote the index of the coded bit before interleaving, where N_{CBPS} gives number of coded bits in a single OFDM symbol, *i* the index after first permutation, *j* the index after second permutation. The first permutation works as shown in Equation (2.3):

$$i = (N_{CBPS}/16) \cdot (k \bmod 16) + \mathrm{floor}(k/16). \quad (2.3)$$

Function floor denotes the largest integer not exceeding the argument. The second permutation is defined as shown in Equation (2.4):

$$j = s \cdot \mathrm{floor}(i/s) + (i + N_{CBPS} - \mathrm{floor}(16 \cdot i/N_{CBPS})) \bmod s. \quad (2.4)$$

The value of *s* is determined by the number of coded bits per subcarrier N_{BPSC} given by Equation (2.5).

$$s = \max(N_{BPSC}/2, 1) \text{ [8]}. \quad (2.5)$$

At the receiver end, the interleaved data is arranged back into the original sequence by the deinterleaver.

2.1.1.5 Mapper

The mapper is used for achieving higher throughput where blocks of *l* consecutive bits b_j are combined into higher level "symbols" *a[k]* that can take on $M_a = 2^l$ different real values/level given by Equation (2.6).

$$a[k] \in A = \{a^{(1)}, a^{(2)}, \ldots, a^{(Ma)}\} \quad \text{with} \quad M_a = 2^l . \tag{2.6}$$

A is called the symbol alphabet. Since each symbol *a[k]* carries *l* bits, we can transmit these bits *l* times faster.

$M = M_a^K$ different signals *s(t)* are transmitted corresponding to a total number $L = ldM = ldM_a^K = KldM_a$ of transmitted bits [9]. The symbol rate is reduced by a factor of *l* as compared to the bit rate given by Equation (2.7):

$$R_s = \frac{1}{T_s} = \frac{1}{lT_b} = \frac{R_b}{l} = \frac{R_b}{ldM_a} \tag{2.7}$$

In practice the symbols *a[k]* are mapped to the subcarrier amplitude and phase. Those mapped symbols *a[k]* can be represented as points in the complex plane which is called signal constellation. The signal constellation design makes it possible to reduce the mean signal power without reducing the minimum distance between symbols. Gray codes are binary words, where neighboring symbols differ by exactly one bit, which are mostly used in Gaussian channels. In this thesis the signal constellations like BPSK and QPSK are defined according to Gray-coding constellation mappings and are illustrated in Fig. 10.

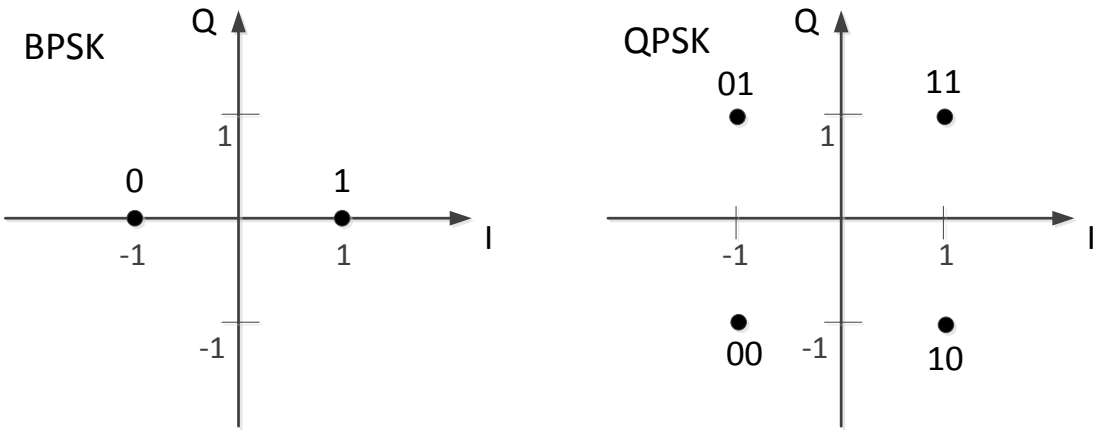

Fig. 10: Constellation Diagrams

In this implementation first the bit stream coming from the interleaver is divided into groups of N_{CBPS}=48 bits for BPSK and N_{CBPS}=96 bits for QPSK. For each group bits are mapped into constellation symbols of N_{BPSC}=1 bit for BPSK modulation and N_{BPSC}=2 bits for QPSK modulation.

To achieve equal average symbol power, the resulting 48 complex pairs are then normalized by multiplying the resulting value (S_I+jS_Q) in BPSK by 1 and in QPSK by a factor of $\frac{1}{\sqrt{2}}$.

2.1.1.6 OFDM Symbol Assembler

Input of the OFDM modulation is 48 complex numbers (one OFDM symbol) coming from the symbol mapping block. Then each complex number is mapped to one of 48 subcarriers represented as frequency offset indices. Also pilots are inserted in the four other subcarriers (-21, -7, 7, 21) giving the 52 total subcarriers per OFDM symbol. Pilots ensure robustness against frequency offsets and phase noise. By looking for pilot signals in the received signal all parameters regarding synchronization and equalization will be deduced. Then 52 complex pairs are padded to create 64 complex pairs. These remaining 12 subcarriers are zero-subcarriers. After modulation the OFDM data is divided into a groups of N_{SD} = 48 complex number with logical numbering 0 to 47. Logical subcarrier numbers are mapped into frequency offset index -26 to 26, while skipping pilot subcarriers and zero DC subcarrier. The subcarriers -32 to -27 and 28 to 32 are set to zero resulting in a guard band. Pilot subcarriers for the n^{th} OFDM symbol are produced by the multiplication between the values {1,1,1,-1} and the second element of sequence p_n. The sequence $p_n$controls the polarity of the pilot subcarriers, which is given by

p_n*={+1,+1,+1,+1,-1,-1,-1,+1,-1,-1,-1,-1,+1,+1,-1,+1,-1,-1,+1,+1,-1,+1,+1,+1,-1,+1,+1,+1,+1,+1,+1,-1,+1,+1,+1,-1,+1,+1,-1,-1,+1,+1,+1,-1,+1,-1,-1,-1,+1,-1,+1,-1,-1,+1,+1,+1,+1,-1,-1,+1,+1,-1,-1,+1,-1,+1,-1,+1,+1,-1,-1,-1,+1,+1,+1,-1,-1,-1,-1,+1,-1,-1,+1,-1,+1,+1,+1,+1,-1,+1,-1,+1,-1,+1,-1,-1,-1,-1,-1,+1,-1,+1,-1,+1,-1,+1,+1,+1,-1,-1,+1,-1,-1,-1,+1,+1,+1,-1,-1,-1,-1,-1,-1,-1}*

The sequence p_n is the cyclic extension of the 127-element sequence and can be generated by the scrambler when the all ones initial state is used, and by replacing all ones with negative ones and all zeros with ones [6].

2.1.1.7 IFFT

Fourier transform is efficient computational tool specifying given signal amplitude as a function of time or frequency. Each 48 complex sample coming from the mapper is associated with one OFDM symbol. For each complex sample also a subcarrier position in the OFDM symbol is associated. The first two OFDM symbols are called preamble and the last one postamble. After modulation different OFDM data subcarriers are transformed to a time domain signal by IFFT operation as shown in the Equation (2.8).

$$s_n = \frac{1}{N} \cdot \sum_{k=0}^{N-1} S_k \cdot e^{\frac{2\pi ikn}{N}}. \quad (2.8)$$

IFFT block converts the data from frequency-domain into a time-domain. *N* orthogonal sinusoids are basis functions for an IFFT. In this thesis IFFT is made up of *N*=64 samples. Input symbols behave like a complex weight for the corresponding sinusoidal basis functions. Through IFFT all 64 sinusoidals are added to make up a single OFDM symbol.

The coefficients from 1 to 26 are mapped to the same numbered IFFT inputs, the coefficients -26 to -1 are copied into IFFT inputs 38 to 63, while the rest of the inputs 27 to 37 and 0 (dc) input are set to zero by forming the 64-point IFFT as shown in the Fig. 11.

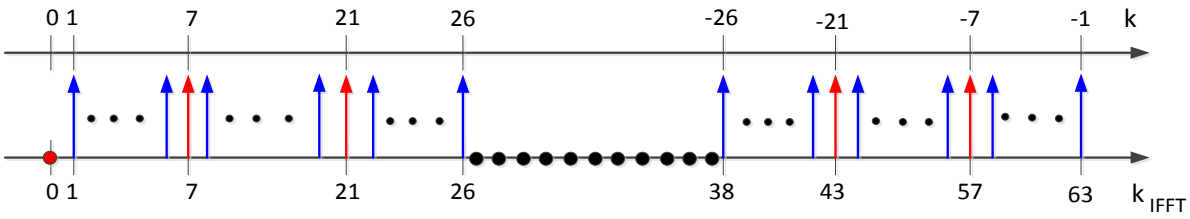

Fig. 11: 64 subcarriers 4 of them pilots [6]

The IFFT is useful for OFDM because it generates samples of a waveform with frequency components satisfying orthogonality conditions. IFFT/FFT operations ensure that subcarriers do not interfere with one another and can be brought closer. Orthogonality between the subcarriers allows them to overlap while avoiding crosstalks. Thus, a significant bandwidth savings can be achieved by using an orthogonal multicarrier technique.

To reduce the effect of the multipath propagation in the form of ISI some redundancy in the transmitted signal has to be added, which is called cyclic prefix (CP). Because of that, the output of IFFT is cyclically extended to the desired length. The cyclic prefix is added to avoid problems in the receiver caused by the mixing of subsequent symbols in the receiver. Cyclic prefix is applied to the signal in the time domain and is meant to create a kind of guard band at the beginning of each symbol. The cyclic prefix is formed by a cyclic continuation of the signal. As mentioned above cyclic prefix is a copy of Fourier transformed waveform corresponding to the last 16 samples of OFDM symbol. The complete time duration of OFDM symbol is given by Equation 2.9.

$$T_{sym} = T_{FFT} + T_{CP} = T_{FFT} + \frac{T_{FFT}}{4} = 8\ \mu s, \tag{2.9}$$

where T_{FFT}= 6.4 μs is IFFT/FFT period and T_{CP} is a duration of cyclic prefix. Both Fig. 12 and Fig. 13 illustrate OFDM frames separated by cyclic prefix.

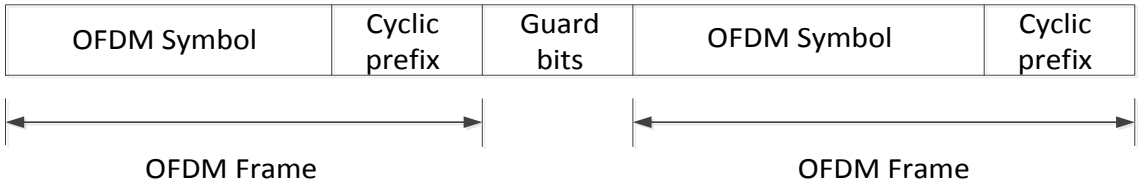

Fig. 12: OFDM frames separated by guard bits

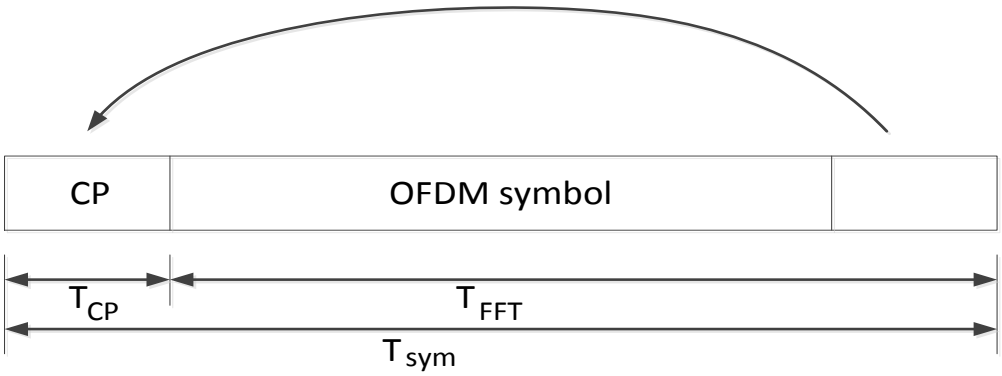

Fig. 13: Cyclic Prefix

With 64 point IFFT and 16 bits long cyclic prefix 80 samples are obtained at the output of the cyclic prefix block as shown in the Fig. 14. 16-tap long cyclic prefix is chosen to be longer than the maximal channel impulse response length. The OFDM symbols can interfere just when the transmission delay of the symbols is bigger than the cyclic prefix duration. The advantages of cyclic prefix are not without a cost. The transmitted energy required to transmit the signal increases with the length of the cyclic prefix. This SNR_{loss} after the insertion of *CP* is given by Equation (2.10):

$$SNR_{loss} = -10log_{10}\left(1-\frac{T_{CP}}{T_{sym}}\right). \quad (2.9)$$

Fig. 14: Cyclic Prefix Block

2.1.1.8 PPDU Frame

During transmission, an IEEE 802.11p physical layer data unit (PPDU) consists of a preamble, header and data. The PPDU includes the OFDM PLCP preamble, OFDM PLCP header, PSDU, tail and pad bits. The PLCP preamble field, which is used for synchronization and channel estimation, consists of ten short training symbols (t_0, t_1, t_2 to t_{10}) and two long training symbols (T_1 and T_2) as illustrated in both Figures 15 and 16. The long training sequence exhibits double duration and consists of two equal OFDM symbols and is used for channel estimation and fine frequency offset estimation. The PLCP preamble is followed by the signal field and data. SIGNAL field is encoded using BPSK with a coding rate of 1/2. However the first DATA symbol includes the 16 bit SERVICE field, where the first six bits are set to zero used to synchronize the descrambler at the receiver.

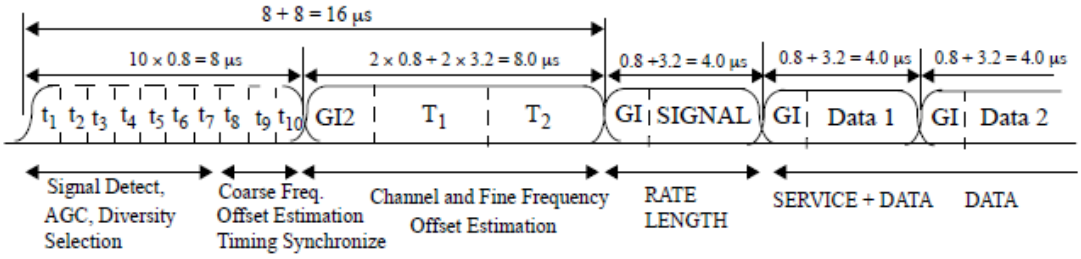

Fig. 15: OFDM Training Structure [5]

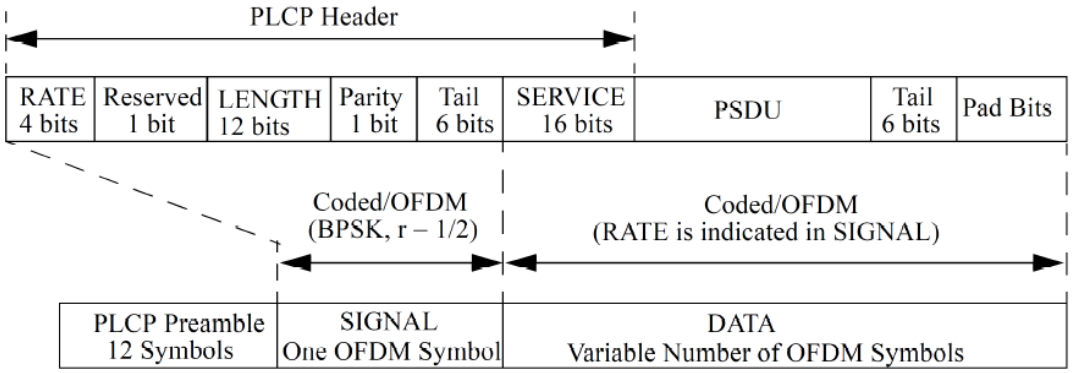

Fig. 16: PLCP Header [5]

A long OFDM training sequence is composed of 53 subcarriers including a zero value at DC, which are modulated by the fixed sequence L, given by:

$L_{-26,26}$ = {0,0,0,0,0,0,+1,+1,-1,-1,+1,+1,-1,+1,-1,+1,+1,+1,+1,+1,+1,-1,-1,+1,+1,-1,+1,-1,+1,+1,+1,+1,0,+1,-1,-1,+1,+1,-1,+1,-1,+1,-1,-1,-1,-1,-1,+1,+1,-1,-1,+1,-1,+1,-1,+1,+1,+1,+1,0,0,0,0,0}

The contents of the two long training symbols are equal, therefore averaging them can be used to improve the quality of the channel estimation. The two long training OFDM symbols, each of duration 6.4 µs with a cyclic prefix of duration 3.2 µs, which is doubled as compared with duration of cyclic prefix of data OFDM symbol, are implemented. The reason is that the cyclic prefix contributes to both long training symbols, instead of prepending each symbol with cyclic prefix of duration 1.6 µs, as is done for data symbols. In this thesis the final PPDU frame is assembled by simply appending long training sequence, signal field and data symbols one after another.

2.1.2 Channel

The channel is the physical media between the transmitter and the receiver. The transmitted signal undergoes channel distortion. AWGN channel attenuates the signal, adds Gaussian-distributed noise and causes phase rotation. Adding noise to transmitted signal involves generating Gaussian random numbers, scaling the numbers according to the desired E_s/N_0 (energy per symbol to noise density ratio) and adding the scaled Gaussian random numbers to the channel symbol values. Typically, noise will be assumed uncorrelated, zero-mean, white with power spectral density $N_0/2$ and Gaussian process. AWGN simply means ‘Additive’ added to any noise, ‘White’ uniform power across frequency band and ‘Gaussian’ normal distribution in the time domain. The output of the channel is the convolution of the channel impulse response with the transmitted signal and with the AWGN as illustrated in Fig. 17.

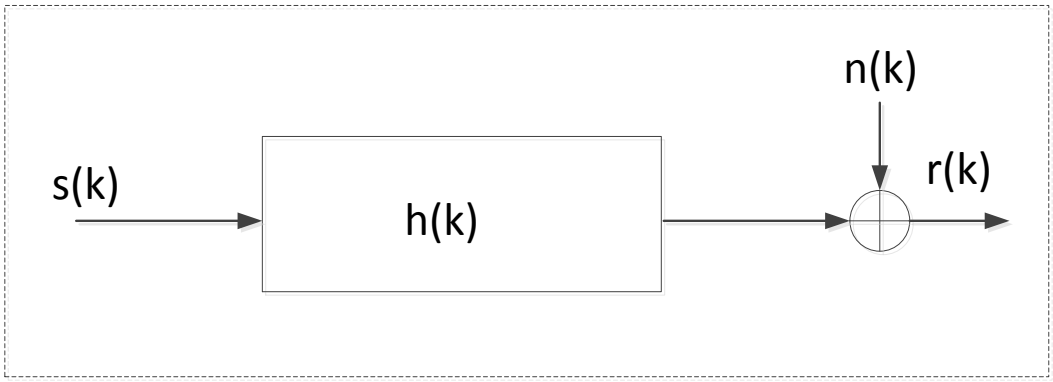

Fig. 17: Channel Model

The frequency response of the channel is estimated in the receiver.

By the OFDM transmission technique each subcarrier has an individual transfer factor H_k. Therefore, the subcarriers can have different bit error probabilities if the same modulation scheme is applied to all subcarriers.

If all subcarriers are transmitted with the same normalized transmit power, the subcarrier-specific signal-to-noise ratio (SNR_k) values are calculated in decibel as shown in Equation 2.11:

$$SNR_k = 10 \cdot log_{10} \frac{|H_k|}{\sigma^2}, \quad (2.11)$$

where σ^2 is the variance of the white Gaussian noise.

2.1.2.1 AWGN Channel

To generate random numbers with Gaussian distribution Central Limit Theorem ("law of large numbers") is used. According to the central limit theorem a sum of randoms will approach normal distribution. The law of large numbers shows that the sample mean converges to the distribution mean as the sample size increases. The random variable being observed, which is the sum or mean of many independent identically distributed random variables can be defined by Equation (2.12):

$$\text{Gauss} = \text{mean} + \sqrt{\sigma} \cdot x \quad (2.12)$$

where the variance σ is calculated as given by Equation (2.13):

$$\sigma = \frac{1}{10^{\frac{SNR}{10}}} \quad (2.13)$$

mean is set to zero and x is an independent, identically distributed random variable, which can be defined by Equation (2.14):

$$x = \sum_{i=0}^{NSUM} \frac{rand}{RAND\_MAX} \qquad (x_{new} = x_{old} - \frac{NSUM}{2}) \quad (2.14)$$

where NSUM=12 is used quantity of noise values.

The received signal can then be represented as Equation (2.15):

$$r[k] = h[k] * s[k] + n[k] \quad (2.15)$$

where *s[k]* is a transmit signal, *h[k]* is a channel impulse response and *n[k]* is a noise component.

In the developed model, white Gaussian noise is added by means of the standard AWGN channel block, which adds complex Gaussian noise and produces a complex output

signal, if the input to the block is complex. In this simulation the actual power of the noise in an AWGN channel is predefined by the quantities SNR and *Eb/N0*.

2.1.3 Receiver

Receiver captures the transmitted signal. Fig. 18 shows the main parts of the receiver. Receiver's procedure is just the inverse of the transmitter. As seen in the Fig. 5, the receiver is formed from the Serial-to-Parallel converter, FFT, equalizer, demapper, deinterleaver, decoder (Viterbi decoder), descrambler, and the received binary data stream. First step at the receiver is to parallelize the signal which is done by using the FFT. After FFT the OFDM signal is converted into time domain and the preamble training symbols, signal field and the pilots are removed. Pilot subcarriers and two long training symbols are used for channel estimation. The equalizer compensates the fading effects, which gives the estimate of the transmitted symbol. These symbol sequences are converted into the codeword in the demapper. The deinterleaver reverses the stages of interleaver performed in the transmitter. Viterbi decoder converts the codeword into the dataword. The decoded bits are descrambled to recover the original binary data stream. Finally the received binary data stream is compared with the transmitted binary data, in order to derive the overall error ratio. Submodules of the receiver are presented in the following parts of this chapter.

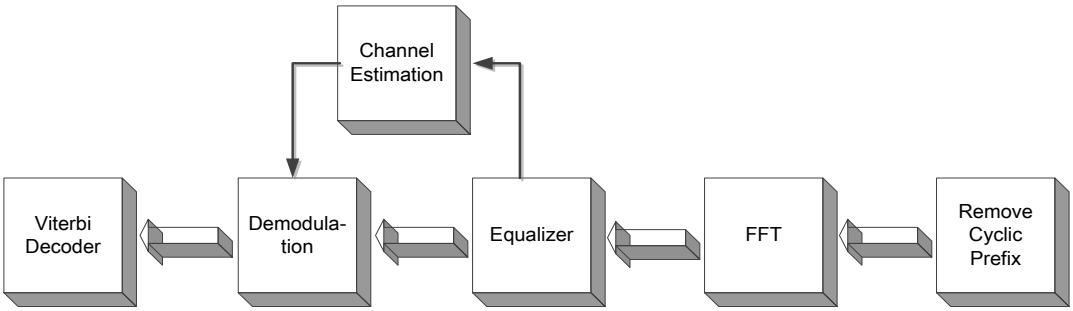

Fig. 18: Structure of the receiver

2.1.3.1 Demodulator

Each PPDU frame after the S/P submodule is divided into blocks. The first two blocks correspond to the ten short training symbols (not needed in our case), the third and the fourth blocks correspond to long training symbols, the fifth block is a signal field and the

rest are data symbols. First of all the cyclic prefix is removed by ignoring first 16 samples of each symbol. After removing the cyclic prefix from 80 samples 64 samples are obtained at the output of the cyclic prefix remover block. The FFT at the receiver side performs the inverse of the IFFT and is defined as in Equation (2.16):

$$S_k = \sum_{n=0}^{N-1} s_n \cdot e^{-\frac{2\pi i k n}{N}} \quad (2.16)$$

At the output of the FFT block subcarriers are rearranged by exchanging the left and right halves of the spectrum and moving the zero-frequency component to the middle. The ideal FFT output should be the original samples that were sent to the IFFT at the transmitter.

2.1.3.2 Channel Estimator

Channel adds some noise into the signal, therefore channel estimation is used to obtain the channel state information. A continuous estimate of the channel behavior is important to ensure reliable data transmission and reception, for this reason a known frame is sent, and the received frame is compared with the original frame for the estimation of channel response.

The channel estimation frame, which is a set of all the possible symbols assigned to each sub-channel, is another important frame that predicts the channel response at the receiver. Once the channel estimation frame is received, this information is used to generate a channel adjustment matrix. The channel adjustment matrix acts as a linear transform applied to all of the following frames to shift and scale the symbols as required to negate the combined effects of the channel.

Due to the cyclic prefix in the OFDM system we assume, that the OFDM symbols are ISI-free and orthogonal. Because of this the system can be described as a set of parallel Gaussian channels. We try to estimate the *K* complex-valued channel coefficient gains. As shown in the Fig. 19 the received data symbol r_k on each subcarrier *k* equals the data symbol s_k, which was transmitted on that sub-carrier, and multiplied with the corresponding frequency-domain channel coefficient H_k in addition to the transformed complex Gaussian random variable n_k given by Equation (2.17):

$$r_k[n] = H_k s_k[n] + n_k[n]. \quad (2.17)$$

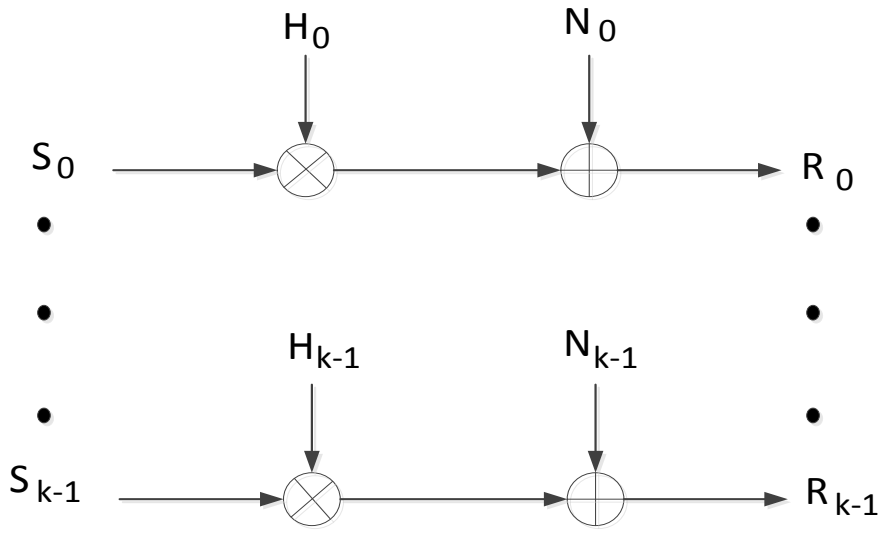

Fig. 19: OFDM System interpreted as pasallel Gaussian channels

First of all a raw channel is estimated at pilot subcarrier frequencies using known pilots. The first step in determining the least squares estimate is to extract the pilot symbols from their known location. As the values of pilot symbols are known, the channel response at these locations can be determined by using the LS estimate which is obtained by dividing the received pilot symbols by their expected values.

The estimation of channel coefficients is defined as in Equation (2.18):

$$\hat{H}_k[n] = \frac{r_k[n]}{s_k[n]} = H_k[n] + \frac{n_k[n]}{s_k[n]}. \quad (2.18)$$

The final channel estimation for the k^{th} subcarrier is then given by Equation (2.19):

$$\hat{H}_k = \frac{1}{2}(\hat{H}_k[1] + \hat{H}_k[2]) \quad (2.19)$$

where $\hat{H}_k[1]$ and $\hat{H}_k[2]$ are estimates of the first and second long training symbols.

2.1.3.3 Equalizer

After the channel coefficients are estimated, the equalizer utilizes the information about the channel to remove the distortion of the signal. The equalizer creates estimates of the transmitted binary data, which means the output of the equalizer should be a good approximation to current symbol. Equalization requires an element-wise multiplication of the FFT output by the inverse of the estimated channel transfer function. Zero-Forcing (ZF) strategy is one of the design strategies of the equalizer, which is used in this thesis.

In ZF equalizer multiplication by the complex conjugate of the channel estimate can do the equalization.

Orthogonality of individual subcarriers is assumed to be still present and multiplicative input-output relation of the channel holds. It is also assumed that all channel coefficients H_k are known. The coefficients of the Zero-Forcing equalizer are inverse-proportional to the channel coefficients given by $E_k = 1/H_k$. Hence, the output of the equalizer is given by transmitted samples plus a noise component given by Equation (2.20):

$$\hat{r_k} = E_k r_k[n] = \frac{1}{H_k}(H_k s_k[n] + n_k[n]) = s_k[n] + \hat{n_k}[n]. \quad (2.20)$$

Through Zero-Forcing (ZF)-design total absence of ISI is possible. But the great disadvantage of the ZF equalizer is its effect on the noise. Hence values of E_k near zero will result in significant noise amplification.

2.1.3.4 Demapper

Each received symbol sequence at the output of the Equalizer is demapped into the corresponding binary word, which is an inverse process of the mapper at transmitter. These detected symbols can be any complex number and can also be different from the transmitted symbols $\hat{a} \neq a$, where $\hat{a}$ is not from symbol alphabet $A = \{a^{(1)}, a^{(2)}, \ldots, a^{(Ma)}\}$, which means that symbol errors have occurred. The output of the demapping are detected coded bit sequences $\hat{c}_j$ with bit rate $R_b = lR_s$. If no symbol errors have occurred, the detected coded bit sequence equals the original coded bit sequence $\hat{c}_j = c_j$. In the case of symbol errors, some of the detected coded bits will be incorrect. Hard demapping and soft demapping are two types of detection methods. In this thesis hard demapping is used, which is based on the minimum Euclidian distance between the received symbols and the valid symbols given by Equation (2.21):

$$\hat{a}[n] = \arg \min_{a \in A} \left[q[k] - a \right] = \arg \min_{a \in A} \sqrt{\left(q_R[k] - a_R \right)^2 + \left(q_I[k] - a_I \right)^2} \quad (2.21)$$

2.1.3.5 Deinterleaver

The deinterleaver converts the interleaved hard-coded bits into the original code sequence. As a result of deinterleaver, correlated noise introduced in the transmission channel appears to be statistically independent at the receiver, allowing better error correction. Like interleaver also deinterleaver performs two consecutive permutations.

Let j=0, 1,….N_{CBPS}*-1* denote the index of the received bit before deinterleaving, *i* the index after first permutation and *k* the index after second permutation. The first permutation works as shown in Equation (2.22):

$$i = s \cdot floor(j/s) + j + (floor(16 \cdot j/N_{CBPS})) \bmod s. \quad (2.22)$$

As already mentioned function floor denotes the largest integer not exceeding the argument. The second permutation is defined as shown in Equation (2.23):

$$k = 16 \cdot i - (N_{CBPS} - 1) floor(16 \cdot i/N_{CBPS}). \quad (2.23)$$

Like at interleaver also at deinterleaver the value of *s* is determined by the number of coded bits per subcarrier N_{BPSC} and is defined in Equation (2.5).

2.1.3.6 Viterbi Decoder

Beside the channel estimator and the equalizer, the purpose of the encoder and decoder is to minimize the effect of channel noise. Viterbi decoder, which is a part of this thesis, was developed by Andrew J. Viterbi. Viterbi decoder together with the convolutional encoder is the most popular FEC (Forward error correction) technique that is particularly suited to the AWGN channel. FEC communication scheme adds redundancy to the transmitted data so that any errors introduced by the channel can be corrected at the receiver.

Viterbi decoding based on the maximum likelihood algorithm has a fixed decoding time, which is well suited to hardware implementations. At the Viterbi decoder, actual received encoded data plus the noise is compared with the encoded data sequence for each of the possible outputs of the convolutional encoder. In other words in Viterbi decoder, the deinterleaved coded bit sequence $\hat{c}$ is converted into the transmitted bit sequence $\hat{b}$, which means any error introduced by the channel is corrected. Fig. 20 shows the block diagram of Viterbi decoder. Maximization of the log-likelihood function represents the probability that the decoder output sequence matches the encoder input sequence. Viterbi decoder is simple to implement and offers large coding gain but it needs to process O(2^K) transitions each bit time, meaning the time complexity is exponential in constraint length K.

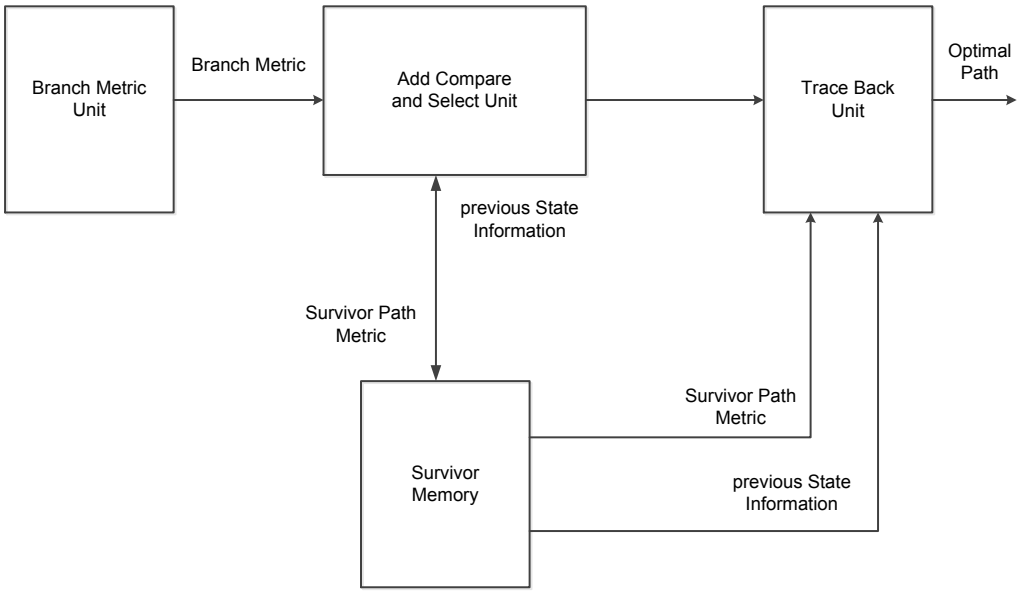

Fig. 20: Block diagramm of Viterbi decoder

The Viterbi decoder receives a pair of channel symbols and determines a distance metric between the received channel symbol pair and the possible channel symbol pairs. The mapping of an existing encoder state to a new state based on the input bit value is illustrated by a trellis diagram, which is a time indexed version of a state diagram. Trellis diagram consists of nodes and branches. Each column has the set of states, each state in a column is connected to two states in the next column. States of the convolutional encoder are values of the encoder's register. State transitions $\Psi_i \rightarrow \Psi_{i+1}$ represent branches in the trellis diagram.

If there are no errors the path in the trellis would match up with the received sequence. Finding the most likely transmitted sequence means having the minimum BER. Because of that Viterbi decoder is used to capture the errors occurred through the states of the trellis and to navigate the trellis without actually materializing the entire trellis. The error probability depends on the constraint length (larger K better error correction), the number of generators (larger this number, lower the rate and the better the error correction) and the amount of noise.

Fig. 21 gives an example of a trellis diagram. Let the state at time *i* be defined as the current content of this encoder's register, so the collection of the outputs of the *J* delay elements at time *k* is given by Equation (2.24):

Ψ_i *= (s[k-1], s[k-2], …, s[k-J]).* (2.24)

Since each sequence can take *M* different values, there are M^J different states. Different states Ψ_i for some time index *I* are represented by the nodes of the trellis diagram. Viterbi decoder finds the most likely transition and ignores the other transitions. In this thesis Viterbi decoder with *J=6* delay elements, *M=2* different values and $M^J=2^6=$ 64 different

states is implemented. Meaning BMU calculates sixty four set of hamming distances and each set consists of two values.

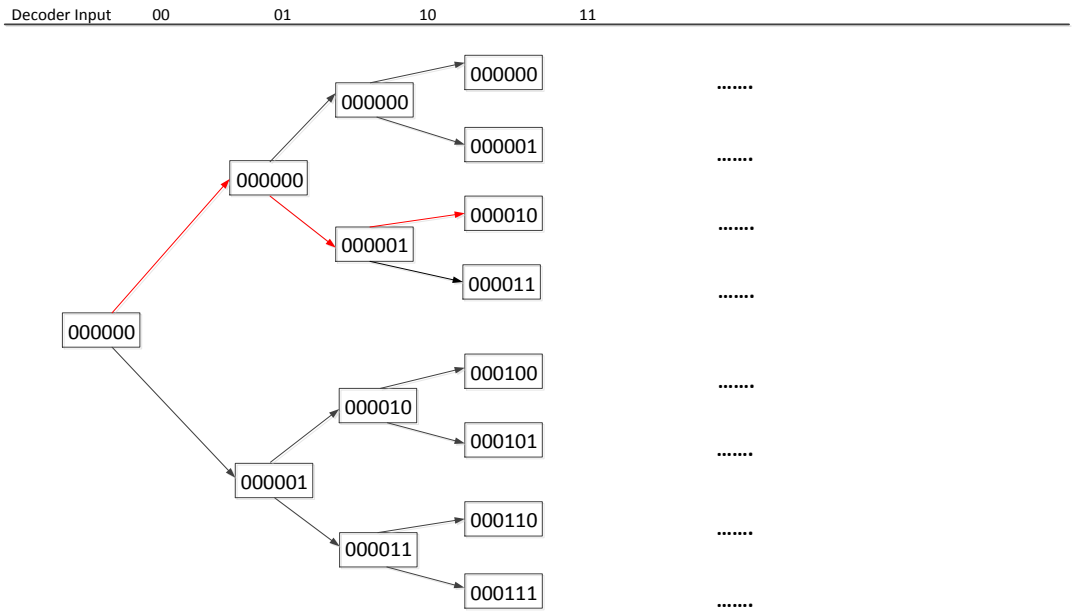

Fig. 21: First three levels of the decoder

As already mentioned branch metric (BM) and the path metric (PM) are two metrics used in Viterbi algorithm. Branch metric is a measure of the distance between what was transmitted and what was received. Hard decision decoder and soft decision decoder are two main methods of branch metric calculation depending on the decoder type. By the "soft decision" decoder method a branch metric is calculated using the Euclidean distance.

In the "hard decision" decoder a branch metric is calculated using the Hamming distance between the received pair of bits and the "ideal" pair, where the result can only be zero, one, or two. In this thesis a hard decision decoder is used, where the branch metric is calculated as given by Equation (2.25):

$$d_k^2[i] = |r[i] - s_k[i]|^2. \quad (2.25)$$

Patch metric is a value associated with a state in the trellis. The path metric is calculated with the "Add-Compare-Select" procedure as also shown in Fig. 22, where at a new state branch metric is added to the path metric from the old state, then the sums for paths arriving at the new state are compared and the path with smallest value, called survivor path, is selected. Path metric is represented by Equation (2.26):

$$d_k^2[i] = \sum_{l=1}^{i} |r[l] - s_k[l]|^2. \quad (2.26)$$

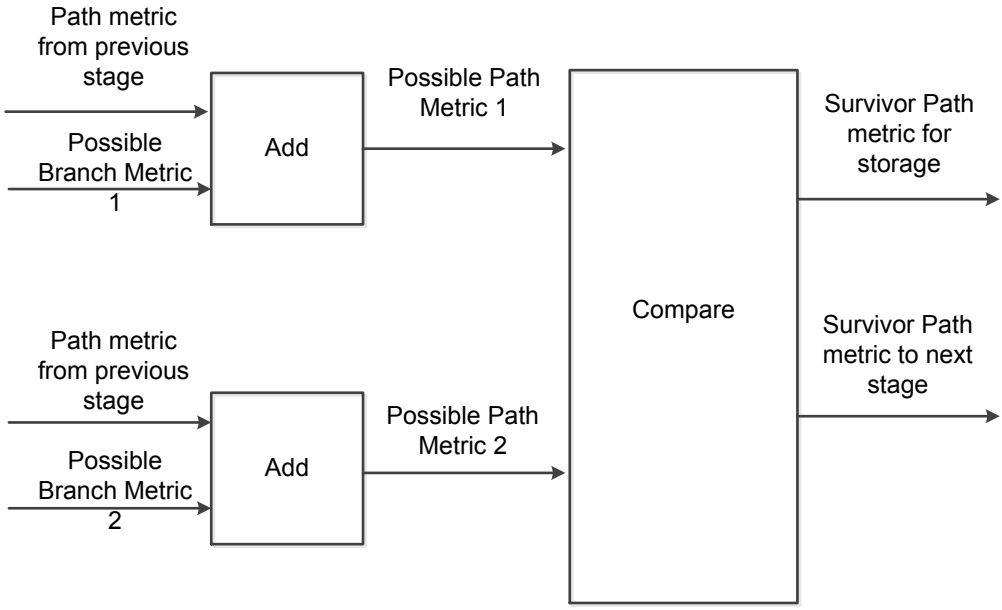

Fig. 22: Block diagram of Add Compare and Select Unit

At the current time point *i,* M^J survivor paths have to be stored. It has been proved that all survivor paths merge after decoding a sufficiently large block of data. In particular, decoding depth *D* is a very important parameter. A decoding depth should be considerably large for quality decoding. Increasing *D* decreases the probability of a decoding error, but also increases latency [10]. Any deeper traceback increases decoding delay and decoder memory requirements. In this thesis traceback depth of 5xK is used, which is very sufficient for Viterbi decoding.

2.1.3.7 Descrambler

The descrambler block descrambles input data sequence. Descrambler is the inverse of the scrambler block at the transmitter. At each time step, the input causes the contents of the registers to shift sequentially.

2.2 2x1 OFDM System

Until now, OFDM system based on IFFT and FFT with one transmit and one receive antenna signaling over the channel was described. However, OFDM-based system is a special case of multiuser systems. The goal of this subsection is the development of a model for the data transmission in an OFDM-based system over a MISO channel.

MISO system provides a better performance in two different respects, the diversity and the multiplexing. Receiver antenna diversity at the remote units needs to remain relatively simple, inexpensive and small. Therefore, for commercial reasons, transmit diversity schemes are becoming increasingly popular as they promise high data rate transmission. The diversity gain improves the quality of the communication link and is obtained through the use of Space Time Coding (STC). To obtain space diversity gain, multiple transmit antennas and at least one receive antenna must be used.

By the use of OFDM in MISO systems, the bit stream is distributed onto many parallel subcarriers with a low data rate. The difference of the 2x1 OFDM system from the 1x1 OFDM system is that MISO, diversity and space-time coding rules are used. In this thesis, MISO is implemented using Alamouti algorithm. ST-OFDM is based on the transmit diversity scheme, where the transmitter is formed from two transmit antennas and the signals are combined through Alamouti Space-Time codes. Block diagram of such communication system is illustrated in the Fig. 23.

In this subsection, an OFDM system equipped with N_{FFT} subcarriers (*K* data subcarriers, remaining N_{FFT}-*K* subcarriers are zero), N_T= 2 transmit and N_R=1 receive antennas will be discussed. It can be seen that channel bandwidth *B* is decomposed into N_{FFT} orthogonal MIMO channels, each with bandwidth B/N_{FFT}, which in our case corresponds to two transmit channels. Also, different channel gains exist. The transmit function depends on the space and time components. Using Alamouti-Coded OFDM system with 64 subcarriers can turn a frequency-selective MISO channel into a set of parallel frequency-flat MISO channels.

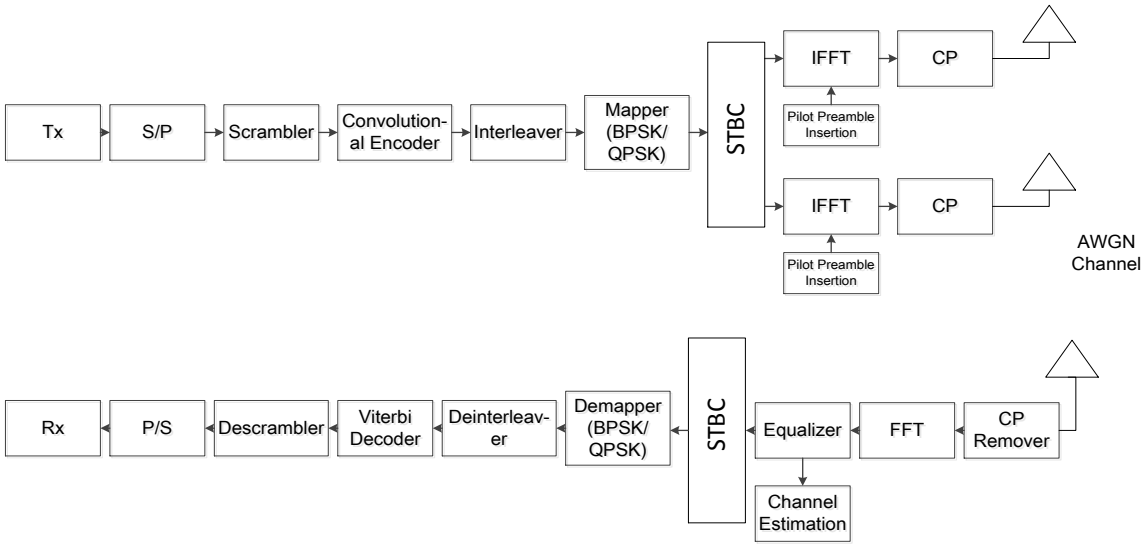

Fig. 23: Block Diagram of 2x1 Alamouti Space-Time Coded OFDM System

2.2.1 Alamouti Coding

As mentioned above second main part of this thesis investigates the application of the Alamouti Space-Time coded system. Alamouti Code was introduced by Siavash M. Alamouti in 1998. Alamouti Code is the first STBC that provides full diversity at full data rate for two transmit antennas. Alamouti code improves the quality of reception with low complexity at the receiver. Two transmit and one receive antenna provides the same diversity order as Maximum Ratio Receiver Combining (MRRC) with one transmit and two receive antennas. However, due to the fact that no channel state information is available at the transmitter, there is approximately 3 dB offset between 1x2 MRRC and 2x1 Alamouti STBC BER versus SNR curves. Particularly Alamouti scheme with two transmit antennas is optimum in both the capacity and the diversity. Also, the Alamouti scheme does not require channel state information at the transmitter, which requires a low complexity maximum-likelihood decoding algorithm. To decouple the transmitted signals from different antennas, the channel between individual transmit and receive antenna pairs requires to remain constant during two consecutive OFDM symbol periods. Figures 24 and 25 illustrate how the Alamouti encoding procedure is performed.

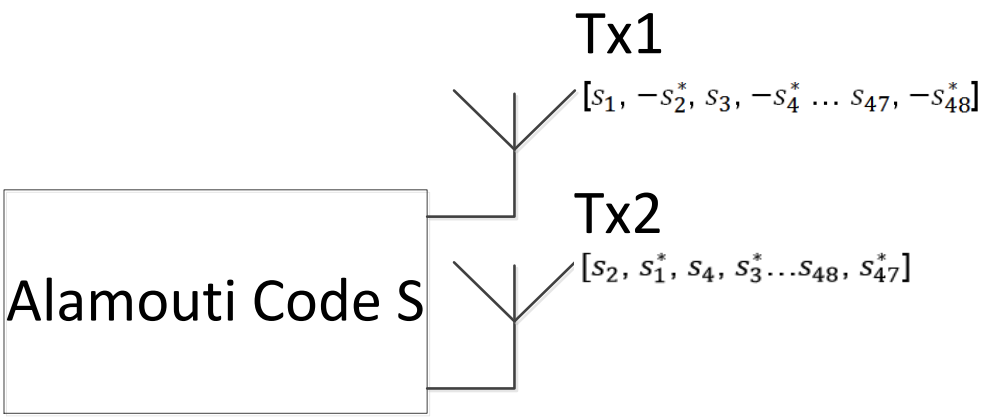

Fig. 24: Alamouti scheme with two transmit antenna

The two SISO channels from the two transmit antennas are assumed to be both time- and frequency selective. The space-time block encoder generates a pair of complex symbols $\{s_1, s_2\}$, which implies that transmitter with two antennas using the Alamouti STBC requires two signaling intervals. In general, transmission matrix of the space-time block code is linear combination of the transmitted symbols and their conjugates. During the first interval, s_1 and s_2 are transmitted from the first and second antenna respectively and during the second, $-s_2^*$ and s_1^* are transmitted from the first and second antenna respectively. In other words, the information symbols are transformed into the space-time code matrix given by Equation (2.27):

$$S = \begin{bmatrix} s_1 & s_2 \\ -s_2^* & s_1^* \end{bmatrix} \quad (2.27)$$

where the first row represents the first transmission period and the second row the second transmission period. Encoding is performed in both time and space domain. As shown in the equations, two consecutive symbols are spatially encoded into four symbols. One of the main properties from Alamouti codeword matrices is the orthogonality, regardless of the underlying modulation scheme [11]. This property enables the receiver to detect s_1 and s_2 using a simple linear signal processing operation. In this thesis Alamouti Space-Time Coding is performed to every symbol in 48-bit data blocks.

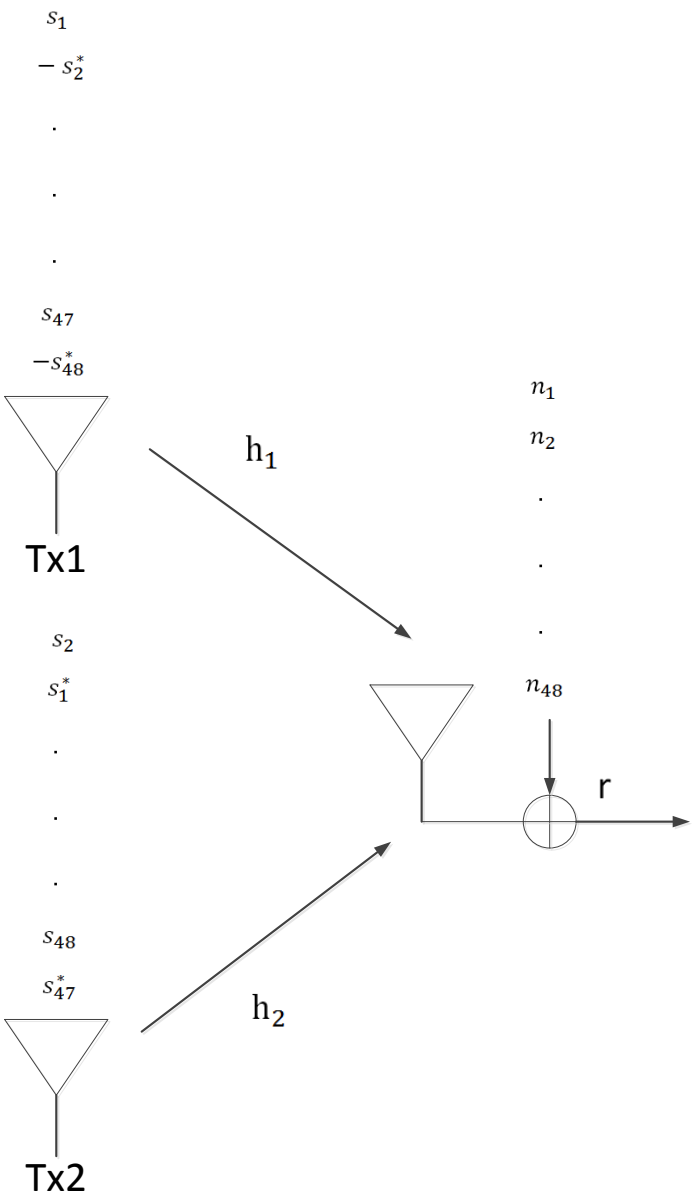

Fig. 25: Alamouti scheme

The observations during the forty-eight signaling interval at a receiver with a single antenna are given by Equation (2.28):

$$\begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{47} \\ r_{48} \end{bmatrix} = \begin{bmatrix} h_1 & h_2 \\ h_2^* & -h_1^* \\ \cdots & \cdots \\ h_{47} & h_{48} \\ h_{48}^* & -h_{47}^* \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ \cdots \\ s_{47} \\ s_{48} \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \\ \cdots \\ n_{47} \\ n_{48} \end{bmatrix} = r = H_{eff} \cdot s + n \tag{2.28}$$

where n is white Gaussian and zero-mean noise and H_{eff} is an effective channel matrix. In the Equation 2.28, a mathematical decoding description is presented. The channel gains are assumed to stay constant during each consecutive symbol interval. Alamouti code transforms a MISO (2x1) channel into an effective MIMO (2x2) channel with transfer function H_{eff}. H_{eff}, which is orthogonal, namely H_{eff}*H_{eff}= $||H||^2 I$, where $||H||^2$= $|h_1|^2$+ $|h_2|^2$ is the squared norm of the underlying MISO channel *H= [*h_1, h_2*]* [12]. The output signal then is given by Equation (2.29):

$r = H_{eff}^* \cdot r = H_{eff}^* \cdot (H_{eff} \cdot s + n) = ||H||^2 \cdot s + \hat{n}$ (2.29)

From the result it is seen that the maximum-ratio combiner output for a two-input single-output channel with Alamouti transmit diversity coding is statistically identical to the maximum-ratio combiner output for a single-input two-output channel with receive diversity and no space- time coding. The Alamouti scheme leads to order-two diversity, without an array at the receiver, without channel knowledge at the transmitter and with only simple linear combining and slicing at the receiver. Alamouti's technique with two transmit antennas can easily be extended to an arbitrary amount of receive antennas to obtain a diversity order of $2M_R^2$.

2.2.2 Channel Estimation of Alamouti 2x1

In this subsection, the basic estimation algorithm in an OFDM MISO system is represented. Time orthogonal method with four linear independent equations and four unknown variables are used. It is assumed that transmitted and received preamble symbols, *s* and *r* are known at the receiver. Also the channel matrix *H* is assumed to stay constant during the transmission, which is given by Equation (2.30).

$$R_k = H_k S_k + N_k$$

$$\begin{pmatrix} r_{1,1}^k & r_{1,2}^k \\ r_{2,1}^k & r_{2,2}^k \end{pmatrix} = \begin{pmatrix} h_{1,1}^k & h_{1,2}^k \\ h_{2,1}^k & h_{2,2}^k \end{pmatrix} \begin{pmatrix} s_{1,1}^k & s_{1,2}^k \\ s_{2,1}^k & s_{2,2}^k \end{pmatrix} + \begin{pmatrix} n_{1,1}^k & n_{1,2}^k \\ n_{2,1}^k & n_{2,2}^k \end{pmatrix} \quad (2.30)$$

Where $r_{i,l}^k$ is the received signal, $h_{i,j}^k$ are the channel gains, $s_{j,l}^k$ the transmit signal and the indices *i, j, l* represent the number of receive antenna, number of transmit antenna and OFDM symbol interval respectively. The transmit signal matrix S_k is always $c\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$ with *c*= 1 or *c*= -1, which depends on subcarrier index *k*. The second method, which is used in this thesis is the time multiplexed MISO channel estimation. It is suboptimal for signal amplification issues where the transmit matrix S_k is always *cI* (identity matrix), which leads to the following channel estimation given by Equation (2.31):

$$\hat{\mathrm{H}}_k = R_k S_k^{-1} = H_k + N_k S_k^{-1} \ , \tag{2.31}$$

where both equations are valid for the case of 2x1 channel estimation [12].

3. Fixed-Point Algorithm and Code

This chapter is composed of two main parts. The first part describes the fixed-point algorithm and the second part gives an understanding about the development of the code. The first part begins with the theoretical explanation of the fixed-point algorithm and fixed-point FFT, in the second part, explanation of the functions written in C programming language are given. In the next chapter the results from the code which are simulated in PC are presented.

3.1 Algorithmic Explanations

The main goal of this thesis is to present an entire 1x1 and 2x1 OFDM System in fixed-point algorithm. The meaning and the way of fixed-point programming in the C programming language is described below.

3.1.1 Fixed-Point Representation

The arithmetic units in digital signal processing hardware usually employ fixed-point computations. The reason is that fixed-point operations are fast, memory and bus sizes are small and use less area and energy. C and other high level programs usually allocate 16 bits to store integers. A K-bit fixed-point number can be represented as an integer or as a fraction. Integer representation may be unsigned or signed. In unsigned case number can take any integer value from 0 to 65535 and on signed integer case two's complement is used to include negative numbers having the range from -32768 to 32767. Integer fixed-point representation is difficult to use due to possible overflows. Also fractional fixed-point representation may be unsigned fraction and signed fraction, in unsigned fraction the 65536 levels are spread uniformly between 0 and 1 but the signed fraction format allows negative numbers equally spaced between –1 and 1. Fractional fixed-point representation, which is also called Q-format, represents numbers between -1.0 and $1\text{-}2^{-(N-1)}$, when N is a number of bits. In fractional fixed-point representation multiplying a fraction by a fraction always results in a fraction, which will not produce an overflow [13].

In this thesis signed integer fixed-point representation is used.

Signed integer number is shown in the Fig. 26. N-bit fixed-point, two`s complement number is shown by Equation (3.1):

$$X = -b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + \ldots + b_0 2^0 \quad (3.1)$$

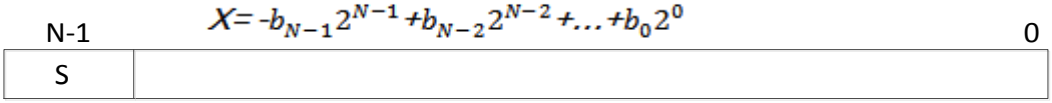

Fig. 26: Signed integer fixed-point representation of a number

For the conversion of the integers into a fixed-point format, first the range of number should be found, than the suitable fixed-point version of the number can be obtained.

General characteristics of the fixed-point implementation are:

1) The processing time is reduced.
2) Is adopted for smaller area and lower energy consumption.
3) May suffer from some loss on the error-rate performance, which needs to be investigated.
4) The processing time is drastically reduced compared with the floating-point counterpart, running at a higher clock frequency.

The fixed-point arithmetic operations are addition, subtraction, multiplication and division. The shifting operation is used for all the arithmetic operations. Shifting is used to displace the position of the binary point. It is equivalent to multiplication or division by a power of two. In order to add two fixed-point numbers, first they must be shifted to have the same binary point position and then can be added with each other. The result is a fixed-point number with a length of *N+1* bits, where *N* represents the number of bits in each number. In multiplication the bits are unsigned and the number of bits of the result is equal to the sum of the numbers of the bits of the multipliers [14]. After the multiplication, if in the result extra signed bits exist, then they should be avoided by shifting the bits to the left. The shifting positions are equal to the number of extra signed bits. More difficult problem is division because hardware divider does not exist. For division reciprocal of the divisor is calculated.

3.1.2 Fixed-Point FFT and Twiddle Factor

FFT is one of the most important tools used to reduce computations of DFT from $O(N^2)$ to *NlogN*, where *N* denotes the length of the input vector. FFT breaks up the original N point samples into two (N/2) sequences, where one is formed from the even-numbered points and the other from the odd-numbered points of the original DFT. The development of FFT algorithm because of complex twiddle factors result in complex variables. Therefore FFT algorithm is designed to perform complex additions and multiplications. However, the input sequence consists of real numbers. For the calculation of an FFT with fixed-point arithmetic the input data is scaled to prevent overflow and to maintain accuracy. Also new conditions on the magnitudes of the input components to avoid overflow during the computation of the FFT are derived. Butterfly-based architecture, recursive algorithm-based architecture, multiplier-accumulator-based structure and ROM operation-based structure are four possible categories used for the FFT/DFT computation. The radix-2 butterfly is the simplest FFT algorithm, which greatly reduces the total computational cost by recursively splitting the DFT as shown in the Fig. 27. The radix-2 FFT of a complex sequence $\{s_n\}$ of length *N* is the complex sequence $\{S_k\}$ of length *N*, given by Equation (3.2):

$$S[k] = \sum_{n=0}^{N-1} s[n] \cdot e^{\frac{-i2\pi kn}{N}}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} s[2n] \cdot e^{\frac{-i2\pi kn}{(\frac{N}{2})}} + W_N^{kn} \sum_{n=0}^{\frac{N}{2}-1} s[2n+1] \cdot e^{\frac{-i2\pi kn}{(\frac{N}{2})}} \quad (3.2)$$

As shown by the Equation 3.2 one radix-2 FFT begins by calculating N/2 2-point DFT. In the next stage these are combined to form N/4 4-point DFT and it continues until a single N-point DFT is produced. Fast Fourier transform with basic radix-2 algorithm, can be characterized in terms of a basic computational unit known as the butterfly. These butterflies process a pair of complex data values to produce a new pair of complex data values which can occupy the same storage locations as the input data. The butterfly is the FFT algorithm represented as a diagram and its operations are complex addition and multiplication by a complex exponential. For radix-2 algorithm, there are log_2^N stages in the computation. At each of the log_2^N stages of the FFT, *N/2* butterflies are computed. For 64-point DFT 6 stages are needed. At each stage, *N* complex values are used to compute *N* new complex values. It is important to guarantee that the data to be transformed has a range which avoids overflow problems. The entire data array is scaled (shift the data to accomplish multiplication by power of two) with the same scale factor to bring the largest value to near full scale.

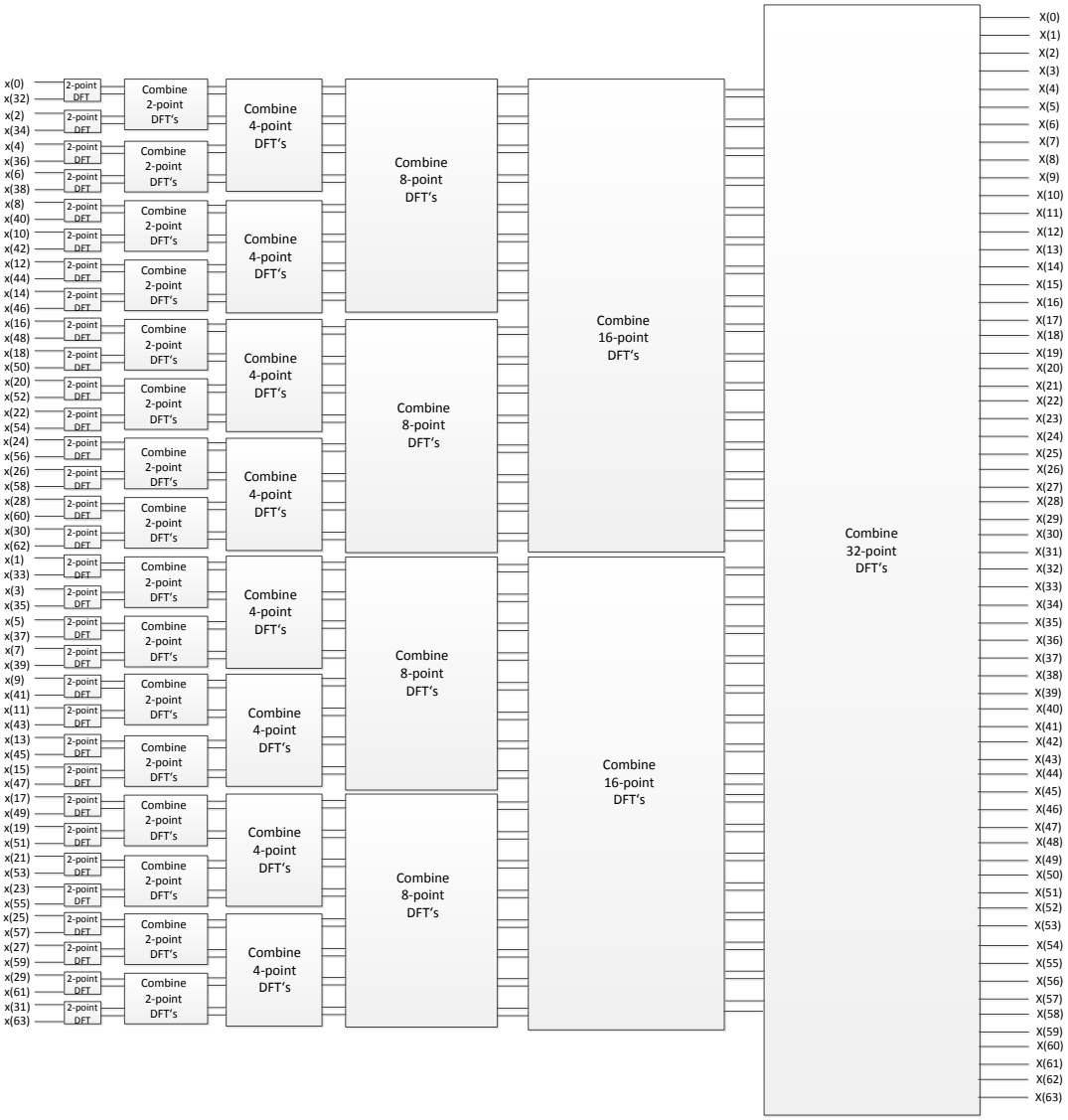

Fig. 27: 6 stages in the computation of the N=64-point DFT

W is the twiddle factor matrix with the elements defined by the complex exponential as in Equation (3.3):

$$W_N^{kn} = exp(\frac{-2j\pi nk}{N}) = cos(\frac{2\pi nk}{N}) - j \cdot sin(\frac{2\pi nk}{N}). \quad (3.3)$$

Twiddle means “Turn around with fingers”, shown also in Fig. 28 it is the factor of rotation.

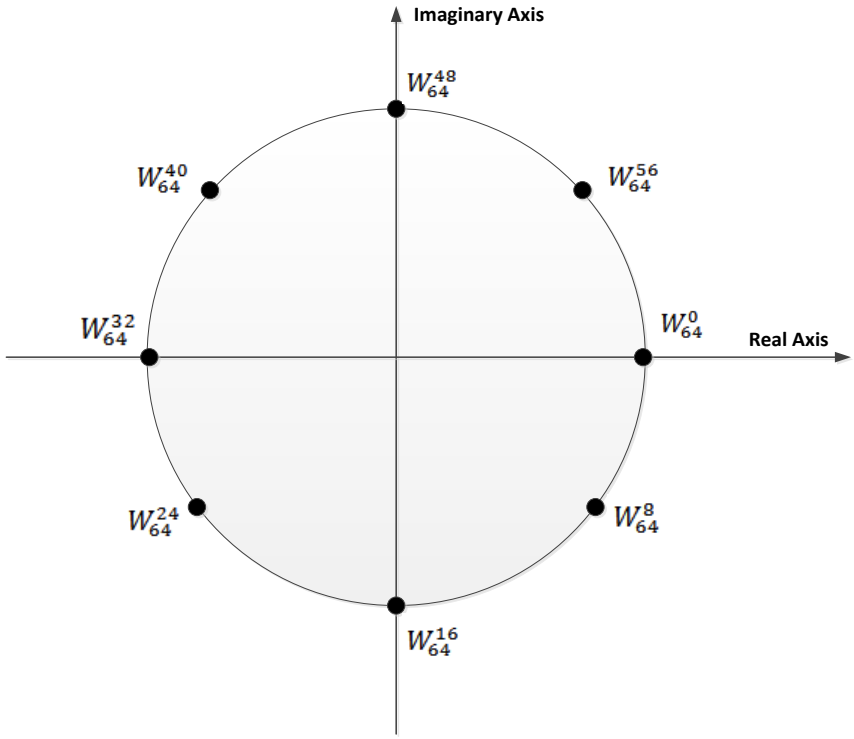

Fig. 28: Twiddle factors for 64-point FFT

In this thesis for the fixed-point FFT first of all the values are scaled between the signed short range of 32767 to -32768. The scaled values that serve as input to the FFT are labeled as Fix Real and Fix Imag. Then the twiddle values are calculated using the equations above. The calculated twiddle factors are scaled to fixed-point values and multiplied by input values to give an output values.

3.1.3 Treatment of End Effect using Zero Padding

Zero padding is also important method during simulations. Because the input signal must be periodic and the duration of the response has to be the same as the period of the data. For the first condition often the data has to be padded with the number of zeros on one end, equal to the maximum positive duration or maximum negative duration of the response function. These zeros protect the output channels from wrap-around pollution. If the response is much shorter than the length of the data set, it can be extended to the same length by padding it with zeros.

3.2 Explanation of the Functions from the Code

Hardware implementation for a SISO and MISO Fixed-Point OFDM system utilizing Alamouti Space-Time Coding is written in C source. The main point of the code is to use a fixed-point algorithm. First block of the transmitter module is a data generator. It randomly produces the payload data bits with equally likely zeros and ones. This generated amount of data is an input parameter to the simulator. The next block, called the scrambler is used to randomize the bit pattern because often the payload bits consist of long runs of zeros or ones. In this thesis the data generator is already random so a scrambler is not really needed, a code can also be implemented without it. The next block of the transmitter module is the convolutional encoder. The data passes through the convolutional encoder, which introduces redundancy in the stream to combat corruptive channel influence. Encoder in this thesis has a code rate of 1/2 and constraint length of 7. After the encoding, the interleaver combats against temporally long channel fades resulting in burst errors. The interleaver simply rearranges the bit streams, which is then reversed at the receiver node.

In order to obtain higher spectral efficiency the output of the interleaver is mapped onto a higher order of modulation symbols, which is carried out by the mapper. In this thesis the supported modulation alphabets for IEEE 802.11p are BPSK and QPSK. Due to the use of Gray coded modulation schemes, the consecutive symbol points differ by only one bit, so a symbol error will most likely result in only one bit error.

Transmission is organized in frames. Frame structure consists of data symbols and preamble. In other words the payload bit size together with the modulation and coding scheme determine the OFDM frame size.

Until the mapping part, the MISO simulation is equal to the SISO simulation. In the MISO simulator space-time encoding process is accomplished before the OFDM symbol assembler. Alamouti Space-Time encoding is represented in Equation (3.4):

$$S = \begin{bmatrix} s_1 & s_2 \\ -s_2^* & s_1^* \\ \cdots & \cdots \\ \cdots & \cdots \\ s_{47} & s_{48} \\ -s_{48}^* & s_{47}^* \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \\ \cdots & \cdots \\ \cdots & \cdots \\ c_{47,1} & c_{47,2} \\ c_{48,1} & c_{48,2} \end{bmatrix}. \tag{3.4}$$

As shown on the Tab. 2 the Alamouti coding scheme is applied to each OFDM symbol in each time interval.

	Time intervals					
	t	t+T	t+2T	t+3T	…	t+47T
Tx_1	s_1	$-s_2^*$	s_3	$-s_4^*$	…	$-s_{48}^*$
Tx_2	s_2	s_1^*	s_4	s_3^*	…	s_{47}^*

Tab. 2: Coding process of Alamouti scheme

48 time intervals exists because OFDM symbol comprises 48-bit data blocks. The symbol stream obtained from the mapper is grouped into blocks of two consecutive symbols. The elements of every block with index *b* are labeled s_1 and s_2. All these blocks are encoded by the Alamouti encoding procedure ending up in four code symbols $c_{i,l}^{b}$ which are s_1, s_2, $-s_2^*$ and s, where the indices *i* and *l* represent the transmit antenna number and the OFDM symbol number modulo 2, respectively.

After encoding every block of the consecutive symbols, the encoded symbols c are arranged at the various positions in the time-frequency grid. After rearrangement the comb pilot subcarriers and preamble are added. Then, encoded and assembled data is modulated by an IFFT transformation with N_{FFT} = 64 and N_{GI} = 16.

The core functions of this code are IFFT and FFT. In these two functions we can see the effect of fixed-point numbers. IFFT and FFT are not simply presented through their sine and cosine functions. They are presented through the twiddle factor W_N^{kn}.

For 64 subcarriers in OFDM we get a 64x64 matrix of twiddle factor and the output of the IFFT can be expressed as in Equation (3.5).

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_{63} \end{bmatrix} = \frac{1}{N} \cdot \begin{bmatrix} W^0 & W^0 & W^0 & \cdots & W^0 \\ W^0 & W^1 & W^2 & \cdots & W^{63} \\ W^0 & W^2 & W^4 & \cdots & W^{62} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ W^0 & W^{63} & W^{62} & \cdots & W^1 \end{bmatrix}^* \cdot \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ \vdots \\ S_{63} \end{bmatrix} \tag{3.5}$$

The values of the inverse twiddle factor matrix are converted to the fixed-point values, where 1 is equal to 32767 and -1 to -32768. All the inverse twiddle factors are represented with limited number of bits and the loss due to the inexact coefficients is called coefficient quantization error.

Code works with 16 bit long variables. If the output values of IFFT lie outside the values 32767↔ -32768, the bits have to be shifted in order not to lose too much information.

Afterwards, the OFDM signals pass through the AWGN channel, which introduces white Gaussian noise. As mentioned above just the values in the AWGN channel are not fixed-point. The input output relation of a 2x1 MISO channel model is given by Equation (3.6).

$$y[n] = h_1[n,m]s_1[n] + h_2[n,m]s_2[n] + n[n]. \tag{3.6}$$

AWGN system model is represented with Gaussian distributed random variables with impulse response *H*=1 and *N* additive white Gaussian noise. The model is presented in a per-subcarrier manner. Since during the entire transmission the impulse responses for every link branch remain constant the time dependence vanishes as given by Equation (3.7).

$$Y = HS + N \tag{3.7}$$

Since the transmitted signal is in general a complex valued sequence, a complex valued AWGN component has to be added. The elements of *N* are derived from the noise component and added to the discrete time domain signal on receive antenna.

FFT at the receiver does not change the statistical properties of the noise component, the elements of *N* are complex valued, stationary, zero mean, circularly symmetric, Gaussian-distributed random processes with independent real and imaginary parts, so the variance remains equal to the noise power. This complex-valued output of the AWGN channel is converted into fixed-point values again.

The first submodule at the receiver is the demodulator. At the demodulator, first the cyclic prefix is removed and the remaining 64 samples of the OFDM symbol are directed to an FFT block, which converts the data into a time-frequency grid. At the FFT block the twiddle factor is multiplied with the input of the FFT as shown in the Equation (3.8).

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ \vdots \\ S_{63} \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & \cdots & W^0 \\ W^0 & W^1 & W^2 & \cdots & W^{63} \\ W^0 & W^2 & W^4 & \cdots & W^{62} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ W^0 & W^{63} & W^{62} & \cdots & W^1 \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_{63} \end{bmatrix} \tag{3.8}$$

Also here after the multiplication in case of overflow the bits are shifted. It is the opposite of the IFFT function.

The demodulated symbols are then directed to the channel estimator. After channel estimation equalization follows. Alamouti STC decoding is implemented with the ZF Equalizer. ZF Equalizer provides the unbiased estimates $X_k^{\wedge}=[s_{1,k}^{\wedge} \quad s_{2,k}^{\wedge}]$ and is given by Equation (3.9).

$$X_k^{\wedge} = (H_k^H H_k)^{-1} H_k^H Y_k \tag{3.9}$$

Consequently the estimates $s_k^{\wedge}$ and $s_{k+1}^{\wedge}$ for the transmitted signal s_k and s_{k+1} are obtained as given by Equation (3.10).

$$s_k^{\wedge} = \frac{1}{H_{1,k}H_{1,k+1}^* + H_{2,k}H_{2,k+1}^*}(H_{1,k+1}^* y_k + H_{2,k} y_{k+1}^*)$$

$$s_{k+1}^{\wedge} = \frac{1}{H_{1,k}H_{1,k+1}^* + H_{2,k}H_{2,k+1}^*}(H_{2,k+1}^* y_k - H_{1,k} y_{k+1}^*) \tag{3.10}$$

Hard demapping, based on the minimum Euclidean distance from every distinct receive soft symbol to the set of transmitted symbols, is processed by the slicer given by Equation (3.11).

$$\hat{A} = arg\ min\ ||y\text{-}a|| = arg\ min \sqrt{(y_I - a_I)^2 + \left(y_Q - a_Q\right)^2} \quad (3.11)$$

In the demapper the estimated hard symbol stream is converted into an estimated bit stream. Afterwards in the deinterleaver the rearrangement procedure accomplished by the interleaver is inverted.

The next and the most important submodule of the receiver is the Viterbi decoder. Viterbi decoder is based on a Maximum Likelihood decoding selecting the minimum distance between the received signal and the codeword. Viterbi decoder calculates set of hamming distance. Afterwards cumulative hamming distance until the last branch is added to hamming distance of the new branches by add-compare-select module. After adding operation each current state gets two new cumulative hamming distances, now add-compare-select module compares the size of the two cumulative distances and selects the smaller one as a survivor. The smaller cumulative hamming distance becomes the benchmark for the next computation. Survivor paths of all the states are stored. In last step, survivor paths are stored at each stage. When there are no more encoded bits to process the detection of a node having minimum path metric is done by comparing all the cumulative hamming distances at the last stage.

In this thesis for 48 data bits 24 time points in trellis diagram exists. Current state and the hamming distance are stored for Viterbi algorithm. The path with lowest Hamming distance minimizes the total number of bit errors and is most likely when the BER is low. Because of that the path with lowest hamming distance is codeword, which means the corresponding input number are read.

The payload data bits ŝ are estimated at the receiver through the decision rule with r representing the receive bit sequence and s the transmitted bit sequence given by Equation (3.12).

$$\hat{s} = max\ Pr(y|s) \quad (3.12)$$

4. Simulation Results

This chapter presents the graphical results from the code providing the simulation results obtained from the IEEE 802.11p PHY simulator. The system depends on the certain user-defined parameters that strictly determine the performance of the physical layer. Received bits are compared with transmitted bits and if they are different an error is recorded. After the counted errors BER is calculated by dividing the number of errors by the total number of counted bits. In system performance analyses both uncoded and decoded BER are calculated. The uncoded BER is the error ratio obtained by comparing the encoded bits at the output of the encoder with the undecoded bits at the input of the Viterbi decoder. The decoded BER is the error ratio obtained by comparing the payload data bits at the transmitter with the decoded bits at the output of the Viterbi decoder at the receiver. Code performance is analyzed in terms of decoded/encoded BER as a function of Eb/N0. Eb represents the average energy transmitted per information bit and N0 represents the signal sided power spectral density of a channel. The power profile is normalized so that the overall average channel impulse response energy is equal to 1. Noise at the receiver is assumed to be additive white complex Gaussian noise. The concept of Eb/N0 is to find an analytical relationship between the SNR of the coded bits of a data frame and the frame error probability. SNR is derived from the simulator input parameter Eb/N0 in a linear form, which is used to prescribe the noise power at the receiver. The x-axis measure Eb/N0 and y-axis measure BER. The channel is modeled as an AWGN channel. The channel SNR is swept from 0 to 20 dB. General input parameters are transmission regime, frame length (in bytes), bit energy to noise power spectral density ratio (Eb/N0), AWGN channel and the types of the channel estimator, equalizer, demapper and decoder. The simulation results are visualized on BER-curves versus Eb/N0. Especially the most important point for fair comparison is that the total transmit energy must be equal for SISO and Alamouti. Therefore the mean transmit power in the simulator on SISO and Alamouti is 1. The simulation results visualized by the curves give an idea how to design the hardware for vehicular communications.

4.1 Error Ratio Calculation

As mentioned above the main performance measure used to assess the efficiency is the bit error probability. In particular, using the receive energy per bit, $\mathrm{E_b}$ corresponding to each of the information bits contained in the modulated data symbol is determined and using the two-sided power spectral density $N_0/2$ of the noise, the corresponding bit error probability P_b is simulated for various values of E_b/N_0. Error rate calculation block compares input data from a transmitter with input data from a receiver. It calculates the error rate as a running statistic by dividing the total number of unequal pairs of data elements by the total number of input data elements from one source [15]. If the inputs are bits, the block computes the bit error rate and if the inputs are symbols then the symbol error rate is calculated. Also for an exact calculation of error rate the T_x and R_x signals must share the same sampling rate. In this thesis, simulator output provides information of transmission reliability in terms of Bit Error Rate (BER) versus E_b/N_0.

4.2 Graphical results over AWGN channel

In the following diagrams performance of SISO, fixed-point SISO, 2x1 Alamouti and fixed-point 2x1 Alamouti schemes is compared. Simulations are carried out with a simple AWGN channel model, which helps to analyze key transmission settings. In the Fig. 29 SISO with fixed-point SISO is compared. Simulation is done for uncoded BER with BPSK mapper, code rate ½, frame length 2000 byte, and 500 frames. Fixed-point OFDM is simulated for word length of 16 bit. From the figure it is seen that fixed-point SISO performs with 2 dB difference.

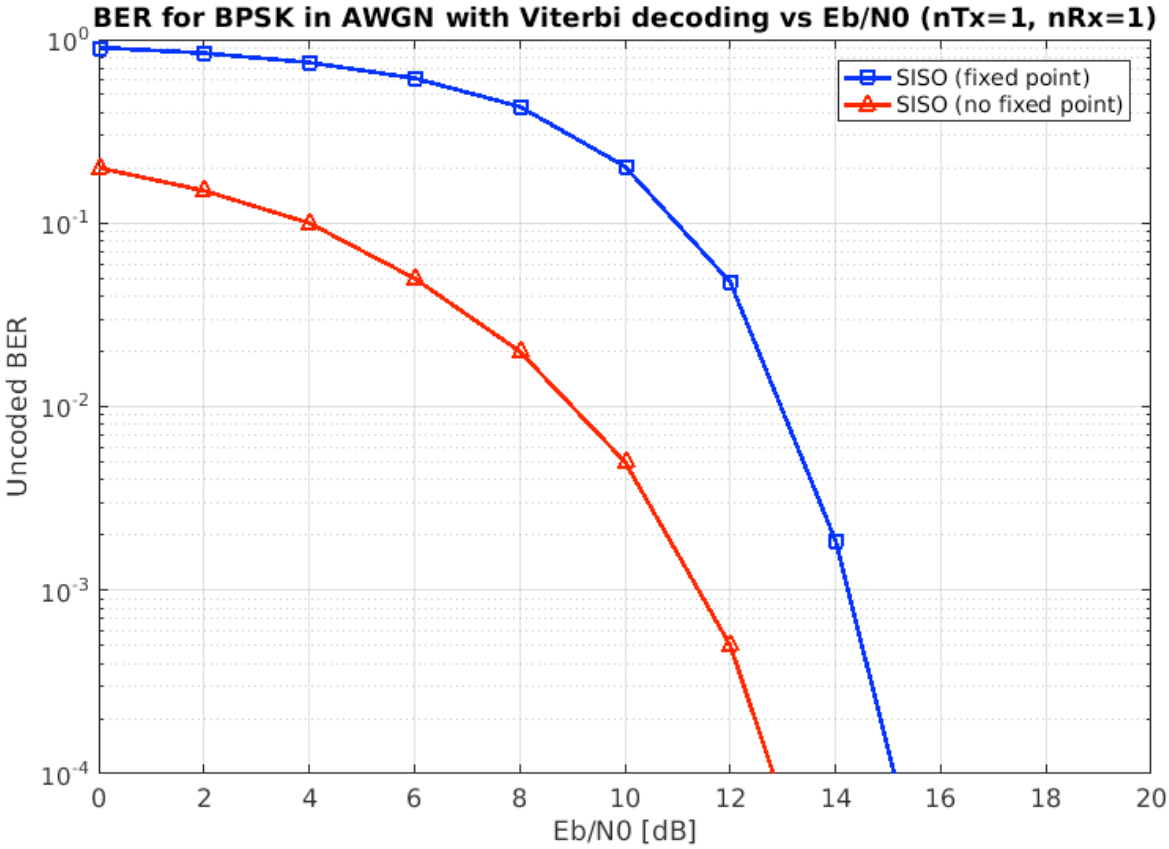

Fig. 29: Simulation Results for SISO and Fixed-Point SISO

Next simulation model was developed to quantify the performance of the Alamouti STBC. Here the channel is also modeled as an AWGN channel. The total transmit power is equally divided by the number of transmit antennas. In the Fig. 30 simulation is done also for uncoded BER with BPSK mapper, code rate ½, frame length 2000 byte and 500 frames. Fixed-point Alamouti is also simulated for word length of 16 bit. From Fig. 30 can be seen that the BER of the fixed-point Alamouti is also higher.

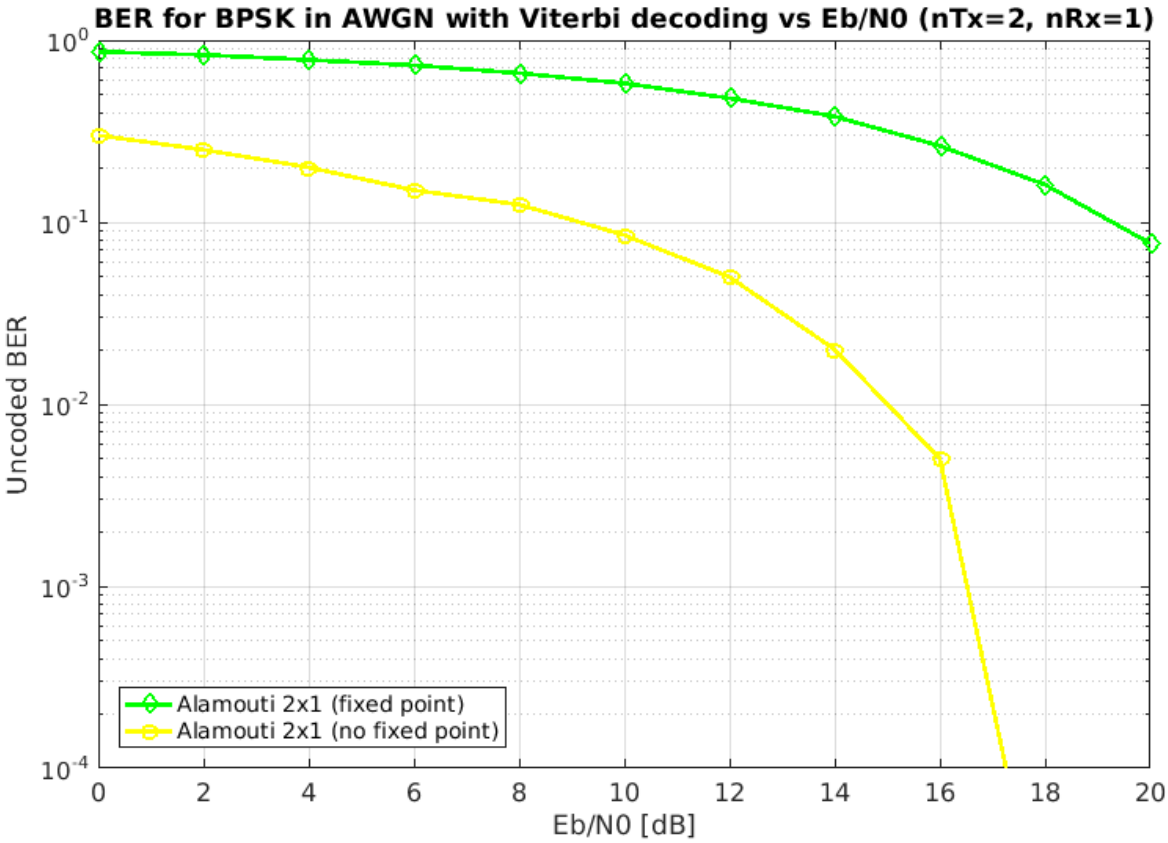

Fig. 30: Simulation Results for 2x1 Alamouti and Fixed-Point 2x1 Alamouti

From both figures (Fig. 29 and Fig. 30) can be concluded that the use of fixed-point arithmetic increases the BER.

Next two figures compare the performances of fixed-point SISO with the fixed-point 2x1 Alamouti. Fig. 31 presents uncoded BER and Fig. 32 presents decoded BER. Both curves fixed-point SISO and fixed-point 2x1 Alamouti reach almost the small BER values when the E_b/N_0 is low. In higher E_b/N_0 values the fixed-point 2x1 Alamouti scheme achieved higher BER than fixed-point SISO.

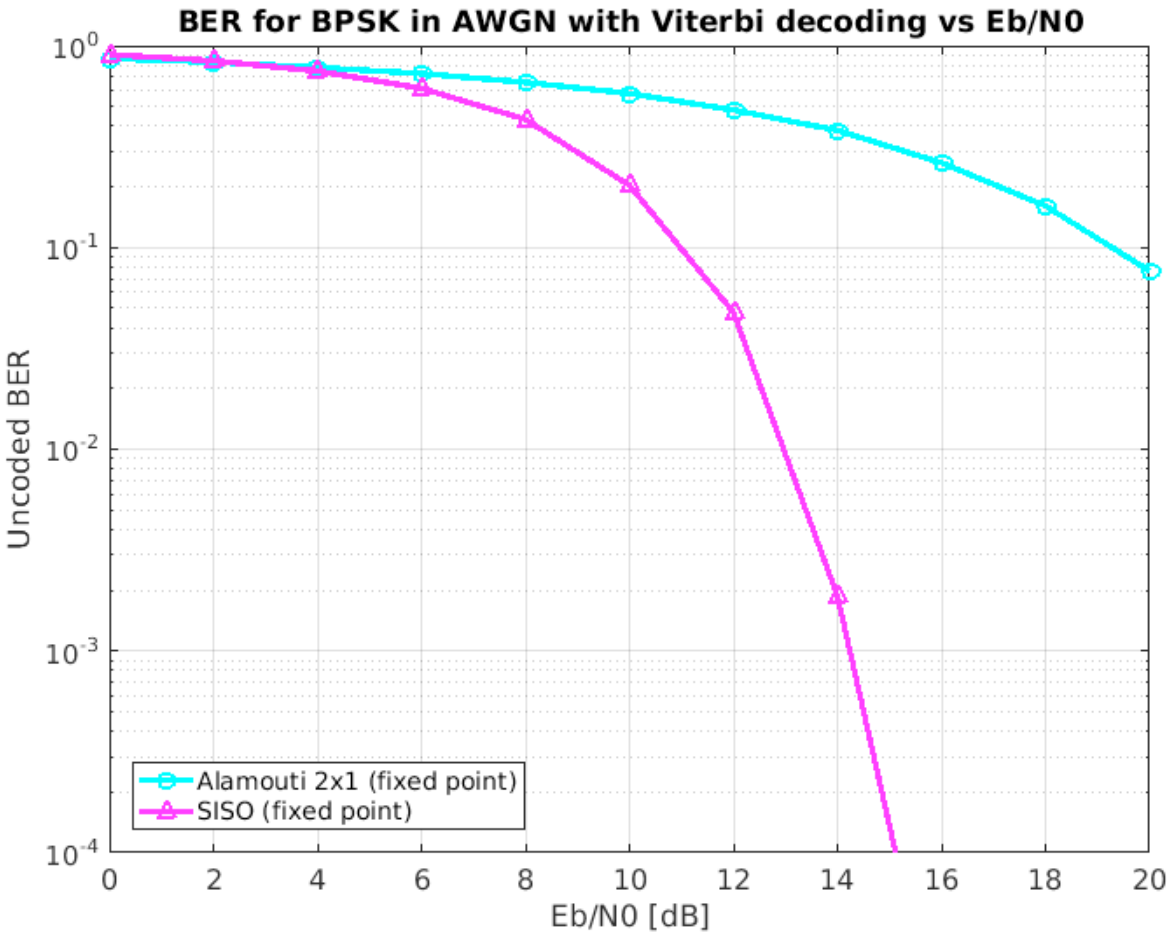

Fig. 31: Uncoded BER for Fixed-Point SISO and 2x1 Alamouti

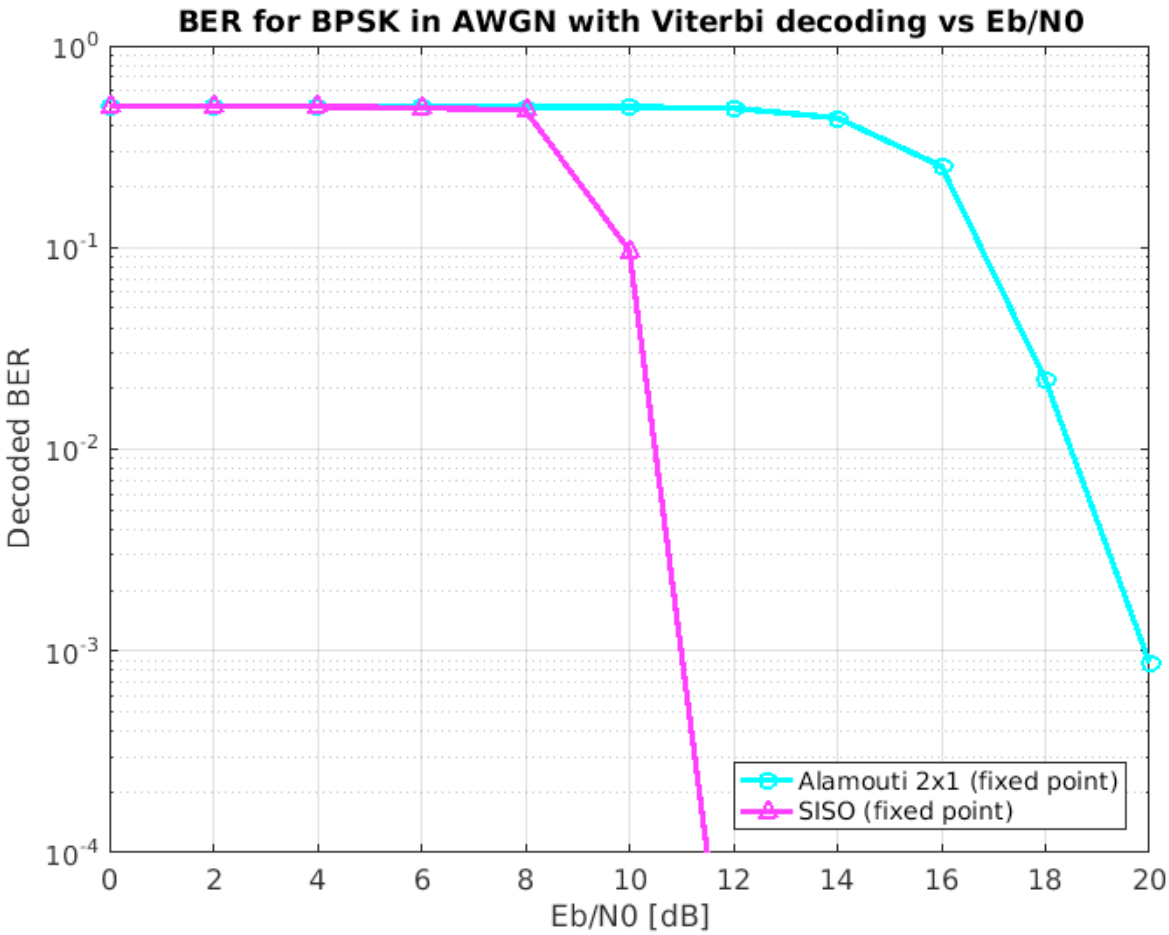

Fig. 32: Decoded BER for Fixed-Point SISO and 2x1 Alamouti

From Fig. 31 and Fig. 32 can also be concluded that MISO scheme need higher transmission power then pure 802.11p SISO transmission in order to obtain the same reliability.

When Fig. 31 and Fig. 32 are compared, it can be seen that for decoded BER better simulation results exists than uncoded BER. This proofs that Viterbi decoder increases the performance by optimizing the decoded bits and lowering the overall BER count. Because as already mentioned decoded BER is the error ratio obtained by comparing the payload data bits at the transmitter with the decoded bits at the output of the Viterbi decoder at the receiver.

5. Conclusion

This thesis investigates the performance of the physical layer defined in IEEE 802.11p. Performance of the OFDM and fixed-point OFDM in SISO and in 2x1 Alamouti system has been implemented and simulated.

First main part of this thesis is the Alamouti Space-Time Coding. As can be seen from simulation results 2x1 Alamouti scheme doesn't bring enhancement over SISO in terms of flow throughput. This is due to the space-time code rate and because of 2x1 Alamouti scheme estimates two impulse responses by utilizing two OFDM symbols.

But Alamouti Space-Time coding reduces the BER at a specific E_b/N_0, without any loss on the data rate, which gives Alamouti's code a significant advantage. Full transmission rate has higher practical interest because it is the right choice in case of lower E_b/N_0 and higher BER. As can be seen from simulations in uncoded BER till 6 dB E_b/N_0 and in decoded BER till 8 dB E_b/N_0 performs 2x1 Alamouti almost the same as SISO system and after 10 dB E_b/N_0, BER decreases more remarkably and at 20 dB E_b/N_0, BER reaches its minimum value but almost with 8 dB difference in comparison to SISO system.

It can be concluded that Alamouti multi-antenna system in vehicular communications will provide reliable performance and low complexity detection schemes. Furthermore, it is also clear that MISO Alamouti improves the system because transmitting data through many low-powered channels is more beneficial than one single high-powered channel system.

Second main part of the thesis is the fixed-point algorithm. From the simulation results it is seen that the fixed-point implementation suits not better than floating-point implementation. Especially, while using different number systems floating-point implementation overcomes fixed-point solution. Floating-point implementation is more useful while dealing with data of different ranges. Implementing fixed-point solutions word length is very important to achieve desired BER performance with respect to floating point performance. Due to the limited word length, both the input signal and twiddle factors are truncated. In simulations 16-bit word length and signed integer fixed-point is used, which performs with 2 dB difference with respect to floating-point OFDM. The main reason is that signed integer fixed-point algorithm introduces overflows, which needs to be considered after each operation. Also the gaps between adjacent numbers are much larger in fixed-point then in floating-point. This means that floating-point has less quantization noise than fixed-point.

But fixed-point algorithm is easier and cheaper to implement and in practice it can be used with user-friendly tools in communication systems.

Bibliography

[1] A. Paier, „The Vehicular Radio Channel in the 5 GHz Band“, Dissertation, Vienna University of Technology, 2010.

[2] A. F. Molisch, „Wireless Communication“, IEEE Press, J. Wiley, 2010.

[3] B. Badic, „Space-Time Block Coding for Multiple Antenna Systems“, Dissertation, Vienna University of Technology, 2005.

[4] S. Andleeb Gillani, „Evaluation and Comparison of IEEE 802.11p and IEEE 802.11b as VANET MAC schemes“, Diploma thesis, Mohammad Ali Jinnah University Islamabad, 2010.

[5] IEEE Std 802.11a, Standards Board, „Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications – High speed Physical Layer in the 5 GHz Band“, 2003.

[6] V. Shivaldova, „Implementation of IEEE 802.11p Physical Layer Model in SIMULINK“, Master thesis, Vienna University of Technology, 2010.

[7] Y. M. Sandesh, R. Kasetty, „Implementation of Convolutional Encoder and Viterbi Decoder for Constraint Length 7 and Bit Rate 1/2“, Research Article, 2013.

[8] A. Costantini, „Implementation of an IEEE 802.11p transmitter in open-source Software Defined Radio“, Master thesis, Universita del Salento, 2008/2009.

[9] F. Hlawatsch, „Modulation and Detection“, Vienna University of Technology, 2010.

[10] Core Technologies, „Viterbi Algorithm for Decoding of Convolutional Codes“, 2004.

[11] J. R. Barry, E. A. Lee, D. G. Messerschmitt, „Digital Communication“, third edition, Springer, 2004.

[12] D. Niklos, „Implementation of Space-Time Codes in IEEE 802.11p-Based Systems“, Master thesis, Vienna University of Technology, 2012.

[13] A. G. M. Cillio, H. Corporaal, „Floating Point to Fixed Point Conversion of C Code“, Delft University of Technology, 1999.

[14] A. Haghparast, H. Penttinen, A. Huovilainen, „Fixed-Point Algorithm Development“, Helsinki University of Technology, 2006.

[15] SIMULINK-Dynamic System Simulation for Matlab (Using Simulink Version 3). The Math Works Inc.

List of Figures

List of Tables

Appendix

FFTW (more specifically version 3) is used as a helper library in the code.
Parts of the code are removed for clarity.

„utils.c“

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "utils.h"

double gen_uniform_random(void) {
   return rand() / (RAND_MAX + 1.);
}

double gen_rayleigh_random(double sigma) {
   double v = gen_uniform_random();
   return sigma * sqrt(-2 * log(v));
}

double gen_standard_normal_random(void) {
   double u = gen_uniform_random();
   return gen_rayleigh_random(1) * cos(M_PI * ( 2 * u - 1 ));
}

double gen_normal_random(double mean, double sigma) {
   return sigma * gen_standard_normal_random() + mean;
}

short gen_binomial_random(double p) {
   return gen_uniform_random() > p ? 1 : 0;
}

void leftshift(const short *in, short *out, unsigned long size, unsigned shift) {
```

```
    unsigned long i;

    for (i = 0; i < size - shift; i++) {
        out[i] = in[i + shift];
    }

    for (i = size - shift; i < size; i++) {
        out[i] = 0;
    }
}

void rightshift(const short *in, short *out, unsigned long size, unsigned shift) {

    unsigned long i;

    for (i = size - 1; i >= shift; i--) {
        out[i] = in[i - 1];
    }

    for (i = 0; i < shift; i++) {
        out[i] = 0;
    }
}
```

„ofdm.h“

```
#ifndef OFDM_H_
#define OFDM_H_

#include <fftw3.h>

#define N_DATA_SUBCARRIERS      48
#define N_PILOT_SUBCARRIERS     4
#define N_TOTAL_SUBCARRIERS     (N_DATA_SUBCARRIERS + N_PILOT_SUBCARRIERS
+ 1)
#define FFT_SIZE                64
```

```
#define CYCLIC_PREFIX_SIZE        16
#define OFDM_SYMBOL_SIZE          (FFT_SIZE + CYCLIC_PREFIX_SIZE)
#define EXT_OFDM_SYMBOL_SIZE    (OFDM_SYMBOL_SIZE + 1)

enum MODULATION_TYPE {
   BPSK,
   QPSK
};

enum CODING_RATE {
   RATE_1_2
};

#define BPSK_NORMALIZATION      1.0
#define QPSK_NORMALIZATION      0.707106781 // 1/sqrt(2)

static const double bpsk_i[] = { -1 * BPSK_NORMALIZATION, 1 * BPSK_NORMALIZATION };
static const double bpsk_q[] = { 0, 0 };

static const double qpsk_i[] = { -1 * QPSK_NORMALIZATION, 1 * QPSK_NORMALIZATION };
static const double qpsk_q[] = { -1 * QPSK_NORMALIZATION, 1 * QPSK_NORMALIZATION };

static const double subcarrier_polarities[] = {
1, 1, 1, 1, -1, -1, -1, 1, -1, -1, -1, -1, 1, 1, -1, 1, -1, -1, 1, 1, -1, 1, 1, -1, 1, 1, 1, 1, 1, 1, -1,1,1, 1, -
1, 1, 1, -1, -1, 1, 1, 1, -1, 1, -1, -1, -1, 1, -1, 1, -1, -1, 1, -1, -1, 1, 1, 1, 1, 1, -1, -1, 1, 1, -1, -1, 1, -
1, 1, -1, 1, 1, -1, -1, -1, 1, 1, -1, -1, -1, -1, 1, -1, -1, 1, -1, 1, 1, 1, 1, -1, 1, -1, 1, -1, 1, -1, -1, -1, -
1, -1, 1, -1, 1, 1, -1, 1, -1, 1, 1, 1, -1, -1, 1, -1, -1, -1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1 };

struct OFDM_PARAMETERS {
        enum MODULATION_TYPE modulation;
        char signal_rate;
        enum CODING_RATE coding_rate;
        unsigned n_bpsc;
        unsigned n_cbps;
        unsigned n_dbps;
};
```

„ofdm.c“

```
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "ofdm.h"
#include "utils.h"

#define MAX(x, y) (((x) > (y)) ? (x) : (y))

struct OFDM_PARAMETERS get_ofdm_parameters(enum MODULATION_TYPE mod) {

	struct OFDM_PARAMETERS p;
	p.modulation = mod;
	switch (mod) {

		case BPSK:
			p.coding_rate = RATE_1_2;
			p.n_bpsc = 1;
			p.n_cbps = 48;
			p.n_dbps = 24;
			p.signal_rate = 0x0D; //0b00001101
			break;

		case QPSK:
			p.coding_rate = RATE_1_2;
			p.n_bpsc = 2;
			p.n_cbps = 96;
			p.n_dbps = 48;
			p.signal_rate = 0x05; //0b00000101;
			break;
```

```
                default:
                        assert(0);
                        break;
        }
        return p;
}

void scrambler(const short *in, short *out, unsigned long size, short *initial_state) {

    unsigned long i;
    short carry;
    short *shift_register = initial_state;

    for (i = 0; i < size; i++) {
        carry = (shift_register[3] + shift_register[6]) % 2;
        rightshift(shift_register, shift_register, 7, 1);
        shift_register[0] = carry;

        out[i] = (in[i] + carry) % 2;
    }
}

void encoder(const short *in, short *out, unsigned long size) {

    unsigned long i;
    short shift_register[7] = { 0 };
    short out_1, out_2;

    for (i = 0; i < size; i++) {
        rightshift(shift_register, shift_register, 7, 1);
        shift_register[0] = in[i];
        out_1 = (shift_register[0] + shift_register[2] + shift_register[3] + shift_register[5] +
shift_register[6]) % 2;
        out_2 = (shift_register[0] + shift_register[1] + shift_register[2] + shift_register[3] +
shift_register[6]) % 2;

        out[2*i] = out_1;
```

```
        out[2*i + 1] = out_2;
    }
}

void interleaver(const short *in, short *out, unsigned N_CBPS, unsigned N_BPSC, unsigned long size) {

    unsigned sym;
    short i, j;
    short s, k;
    short first_permutation[N_CBPS];

    s = MAX((short)(N_BPSC / 2), 1);

    for (sym = 0; sym < (size / N_CBPS); sym++) {
        for (k = 0; k < N_CBPS; k++) {
            i = (((short)(N_CBPS / 16)) * (k % 16)) + ((short)floor(k / 16));
            first_permutation[i] = in[k + (sym * N_CBPS)];
        }

        for (i = 0; i < N_CBPS; i++) {
            j = s * ((short)floor(i / s)) + (i + N_CBPS - ((short)floor((16 * i) / N_CBPS))) % s;
            out[j + (sym * N_CBPS)] = first_permutation[i];
        }
    }
}

void mapper(const short *in, unsigned long size, enum MODULATION_TYPE mod, fftw_complex *out, unsigned sym) {

    struct OFDM_PARAMETERS p = get_ofdm_parameters(mod);
    unsigned long i, idx;

    for (i = 0; i < size; i += p.n_bpsc) {
        idx = i / p.n_bpsc;

        switch (p.modulation) {
```

```
            case BPSK:
                out[idx][0] = bpsk_i[in[i + (sym * p.n_cbps)]];
                out[idx][1] = bpsk_q[in[i + (sym * p.n_cbps)]];
                break;

            case QPSK:
                out[idx][0] = qpsk_i[in[i + (sym * p.n_cbps)]];
                out[idx][1] = qpsk_q[in[i+1 + (sym * p.n_cbps)]];
                break;
        }
    }
}

void add_pilots(fftw_complex *in, fftw_complex *out, unsigned symbol_index) {

        unsigned i;
        double polarity = subcarrier_polarities[symbol_index % 127];

        out[5][0] = polarity;
        out[5][1] = 0;
        out[19][0] = polarity;
        out[19][1] = 0;
        out[33][0] = polarity;
        out[33][1] = 0;
        out[47][0] = -polarity;
        out[47][1] = 0;
        out[26][0] = 0;
        out[26][1] = 0;

        for (i = 0; i < 48; i++) {
                if (0 <= i && i <= 4) {
                        out[i][0] = in[i][0];
                        out[i][1] = in[i][1];
                        continue;
                }
```

```
                if (5 <= i && i <= 17) {
                        out[i + 1][0] = in[i][0];
                        out[i + 1][1] = in[i][1];
                        continue;
                }
                if (18 <= i && i <= 23) {
                        out[i + 2][0] = in[i][0];
                        out[i + 2][1] = in[i][1];
                        continue;
                }
                if (24 <= i && i <= 29) {
                        out[i + 3][0] = in[i][0];
                        out[i + 3][1] = in[i][1];
                        continue;
                }
                if (30 <= i && i <= 42) {
                        out[i + 4][0] = in[i][0];
                        out[i + 4][1] = in[i][1];
                        continue;
                }
                if (43 <= i && i <= 47) {
                        out[i + 5][0] = in[i][0];
                        out[i + 5][1] = in[i][1];
                }
        }
}

void map_ofdm_to_ifft(fftw_complex *ofdm, fftw_complex *ifft) {

        unsigned i;
        ifft[0][0] = 0;
        ifft[0][1] = 0;

        for (i = 1; i <= 26; i++) {
                ifft[i][0] = ofdm[i + 26][0];
                ifft[i][1] = ofdm[i + 26][1];
```

```
        }

        for (i = 27; i <= 37; i++) {
                ifft[i][0] = 0;
                ifft[i][1] = 0;
        }

        for (i = 38; i <= 63; i++) {
                ifft[i][0] = ofdm[i - 38][0];
                ifft[i][1] = ofdm[i - 38][1];
        }
}

void perform_ifft_fft(fftw_complex *in, fftw_complex *out, int sign) {

        fftw_plan ifft;

        fftw_complex *input = fftw_alloc_complex(64);
        fftw_complex *output = fftw_alloc_complex(64);

        unsigned i;
        for (i = 0; i < 64; i++) {
                input[i][0] = in[i][0];
                input[i][1] = in[i][1];
        }

        ifft = fftw_plan_dft_1d(64, input, output, sign, FFTW_ESTIMATE);
        fftw_execute(ifft);

        for (i = 0; i < 64; i++) {
                out[i][0] = output[i][0];
                out[i][1] = output[i][1];
        }

        fftw_destroy_plan(ifft);
        fftw_free(input);
```

```
	fftw_free(output);
}

void ifft(fftw_complex *in, fftw_complex *out) {
	perform_ifft_fft(in, out, 1);
}

void normalize_ifft_output(fftw_complex *in, unsigned size, unsigned fftSize) {

	unsigned i;
	for (i = 0; i < size; i++) {
		in[i][0] *= 1.0 / (double)fftSize;
		in[i][1] *= 1.0 / (double)fftSize;
	}
}

void add_cyclic_prefix(fftw_complex *in, unsigned in_size, fftw_complex *out, unsigned
out_size, unsigned cp_length) {

	unsigned i;
	assert(cp_length < in_size);

	for (i = cp_length; i < out_size; i++) {
		out[i][0] = in[(i - cp_length) % in_size][0];
		out[i][1] = in[(i - cp_length) % in_size][1];
	}

	for (i = 0; i < cp_length; i++) {
		out[i][0] = in[(i + in_size - cp_length) % in_size][0];
		out[i][1] = in[(i + in_size - cp_length) % in_size][1];
	}
}

double gauss(double mean, double snr) {

	unsigned i;
```

```
        const unsigned NSUM = 12;

        double sigma = 1.0 / pow(10.0, snr/10.0);
        double x = 0.0;

        for (i = 0; i < NSUM; i++) {
                x += (double)rand() / RAND_MAX;
        }

        x -= NSUM / 2.0;
        x *= sqrt(12.0 / NSUM);

        return mean + sqrt(sigma) * x;
}

void channel(fftw_complex *in, fftw_complex *out, unsigned size, double ebn0) {

   unsigned i;

   for (i = 0; i < size; i++) {
      out[i][0] = in[i][0] + gen_normal_random(0.0, ebn0);
      out[i][1] = in[i][1] + gen_normal_random(0.0, ebn0);
   }
}

void alamouti_encoder(fftw_complex *in, fftw_complex *tx1, fftw_complex *tx2, unsigned size) {

   unsigned i;

   for (i = 0; i < size; i++) {
      tx1[i][0] = (i % 2 == 0) ? in[i][0] : -in[i][0];
      tx1[i][1] = (i % 2 == 0) ? in[i][1] : - (-1 * in[i][1]);

      tx2[i][0] = (i % 2 == 0) ? in[i + 1][0] : in[i - 1][0];
      tx2[i][1] = (i % 2 == 0) ? in[i + 1][1] : (-1 * in[i - 1][1]);
   }
```

```
}

void alamouti_channel(fftw_complex *in_tx1, fftw_complex *in_tx2, fftw_complex *out, fftw_complex h_1, fftw_complex h_2, unsigned size, double snr) {

   unsigned i;

   for (i = 0; i < size; i++) {
      out[i][0] = (in_tx1[i][0] * h_1[0]) + (in_tx2[i][0] * h_2[0]) + gauss(0.0, snr);
      out[i][1] = (in_tx1[i][1] * h_1[1]) + (in_tx2[i][1] * h_2[1]) + gauss(0.0, snr);

      out[i + 1][0] = (in_tx2[i + 1][0] * (-h_1[0])) + (in_tx1[i + 1][0] * h_2[0]) + gauss(0.0, snr);
      out[i + 1][1] = ((-1 * in_tx2[i + 1][1]) * (-h_1[1])) + ((-1 * in_tx1[i + 1][1]) * h_2[1]) + gauss(0.0, snr);
   }
}

void alamouti_decoder(fftw_complex *in, fftw_complex *out, fftw_complex h_1, fftw_complex h_2, unsigned size) {

   unsigned i;

   for (i = 0; i < size; i += 2) {
      out[i][0] = ((h_1[0] * in[i][0]) + (h_2[0] * in[i + 1][0]));
      out[i][1] = (((-1 * h_1[1]) * in[i][1]) + (h_2[1] * (-1 * in[i + 1][1])));

      out[i + 1][0] = ((h_2[0] * in[i][0]) - (h_1[0] * in[i + 1][0]));
      out[i + 1][1] = (((-1 * h_2[1]) * in[i][1]) - (h_1[1] * (-1 * in[i + 1][1])));
   }
}

void fft(fftw_complex *in, fftw_complex *out) {
         perform_ifft_fft(in, out, -1);
}

void change_sign(fftw_complex *in, unsigned size) {
```

```
	unsigned i;

	for (i = 0; i < size; i++) {
		if ((in[i][0] < 0 && in[i][0] > -1e-4) || in[i][0] == -0.0) {
			in[i][0] = 0.0;
		}
		if ((in[i][1] < 0 && in[i][1] > -1e-4) || in[i][1] == -0.0) {
			in[i][1] = 0.0;
		}
	}
}

void remove_cyclic_prefix(fftw_complex *in, unsigned in_size, fftw_complex *out, unsigned out_size, unsigned cp_length) {

	unsigned i;

	for (i = cp_length; i < in_size; i++) {
		out[(i - cp_length) % out_size][0] = in[i][0];
		out[(i - cp_length) % out_size][1] = in[i][1];
	}
}

void map_fft_to_ofdm(fftw_complex *fft, fftw_complex *ofdm) {

	unsigned i;

	for (i = 1; i <= 26; i++) {
		ofdm[i + 26][0] = fft[i][0];
		ofdm[i + 26][1] = fft[i][1];
	}

	for (i = 38; i <= 63; i++) {
		ofdm[i - 38][0] = fft[i][0];
		ofdm[i - 38][1] = fft[i][1];
	}
```

```
}

void remove_pilots(fftw_complex *in, fftw_complex *out) {

	unsigned i;

	for (i = 0; i < 53; i++) {
		if (0 <= i && i <= 4) {
			out[i][0] = in[i][0];
			out[i][1] = in[i][1];
			continue;
		}
		if (6 <= i && i <= 18) {
			out[i - 1][0] = in[i][0];
			out[i - 1][1] = in[i][1];
			continue;
		}
		if (20 <= i && i <= 25) {
			out[i - 2][0] = in[i][0];
			out[i - 2][1] = in[i][1];
			continue;
		}
		if (27 <= i && i <= 32) {
			out[i - 3][0] = in[i][0];
			out[i - 3][1] = in[i][1];
			continue;
		}
		if (34 <= i && i <= 46) {
			out[i - 4][0] = in[i][0];
			out[i - 4][1] = in[i][1];
			continue;
		}
		if (48 <= i && i <= 52) {
			out[i - 5][0] = in[i][0];
			out[i - 5][1] = in[i][1];
		}
```

```
        }
}

void demapper(fftw_complex *in, unsigned long size, enum MODULATION_TYPE mod, short
*out) {

   struct OFDM_PARAMETERS p = get_ofdm_parameters(mod);
   unsigned long i, idx;

   for (i = 0; i < size; i++) {
      idx = i * p.n_bpsc;

      switch (p.modulation) {

         case BPSK:
               if (in[i][0] <= 0) out[idx] = 0;
               else out[idx] = 1;
               break;

         case QPSK:
               if (in[i][0] <= 0) out[idx] = 0;
               else out[idx] = 1;
               if (in[i][1] <= 0) out[idx + 1] = 0;
               else out[idx + 1] = 1;
               break;
      }
   }
}

void regrouping(const short *in, unsigned in_size, short *out, unsigned index) {

   unsigned i;

   for (i = 0; i < in_size; i++) {
      out[(index * in_size) + i] = in[i];
   }
```

```
}

void deinterleaver(const short *in, short *out, unsigned N_CBPS, unsigned N_BPSC, unsigned long size) {

	unsigned sym;
	short i, j;
	short s, k;
	short first_permutation[N_CBPS];

	s = MAX((short)(N_BPSC / 2), 1);

	for (sym = 0; sym < (size / N_CBPS); sym++) {
		for (j = 0; j < N_CBPS; j++) {
			i = s * ((short)floor(j / s)) + (j + ((short)floor((16 * j) / N_CBPS))) % s;
			first_permutation[i] = in[j + (sym * N_CBPS)];
		}

		for (i = 0; i < N_CBPS; i++) {
			k = 16 * i - (N_CBPS - 1) * ((short)floor((16 * i) / N_CBPS));
			out[k + (sym * N_CBPS)] = first_permutation[i];
		}
	}
}

unsigned long ber(short *x, short *y, unsigned long size) {

	unsigned long i, count = 0;

	for (i = 0; i < size; i++) {
		if (x[i] != y[i])
			++count;
	}
	return count;
}
```

„viterbi.h“

```
#ifndef VITERBI_H_
#define VITERBI_H_

struct pair {
    int first;
    int second;
};

typedef int** Trellis;

#define CONSTRAINT    7
#define PARITY_BITS   2

const unsigned POLYNOMIALS[] = { 109, 79 };

short **OUTPUTS;
```

“viterbi.c”

```
#include <assert.h>
#include <limits.h>
#include <string.h>
#include "viterbi.h"

int reverse_bits(int num_bits, int input) {

    assert(input < (1 << num_bits));
    int output = 0;

    while (num_bits-- > 0) {
        output = (output << 1) + (input & 1);
        input >>= 1;
```

```
    }
    return output;
}

unsigned min_element_index(const int *arr, unsigned first, unsigned last) {

    assert(first < last);

    unsigned i, idx = first;
    int minimum = arr[first];

    for (i = first + 1; i < last; i++) {
        if (arr[i] < minimum) {
            minimum = arr[i];
            idx = i;
        }
    }
    return idx;
}

unsigned hamming_distance(const short *x, const short *y, unsigned size) {

    unsigned i, distance = 0;

    for (i = 0; i < size; i++) {
        if (x[i] != y[i])
            ++distance;
    }
    return distance;
}

void initialize_outputs() {
```

```
    int i, j, k;

    OUTPUTS = (short **)malloc((1 << CONSTRAINT) * sizeof(short *));
    for (i = 0; i < (1 << CONSTRAINT); i++) {
        OUTPUTS[i] = (short *)malloc(PARITY_BITS * sizeof(short));
    }

    for (i = 0; i < (1 << CONSTRAINT); i++) {
        for (j = 0; j < PARITY_BITS; j++) {
            int polynomial = reverse_bits(CONSTRAINT, POLYNOMIALS[j]);
            int input = i;
            int output = 0;

            for (k = 0; k < CONSTRAINT; k++) {
                output ^= (input & 1) & (polynomial & 1);
                polynomial >>= 1;
                input >>= 1;
            }

            OUTPUTS[i][j] = output ? 1 : 0;
        }
    }
}

const short* output(int current_state, int input) {
    return OUTPUTS[(current_state | (input << (CONSTRAINT - 1)))];
}

unsigned branch_metric(const short *in, int source_state, int target_state) {

    assert((target_state & ((1 << (CONSTRAINT - 2)) - 1)) == source_state >> 1);
```

```
    const short *out = output(source_state, target_state >> (CONSTRAINT - 2));

    return hamming_distance(in, out, PARITY_BITS);
}

struct pair path_metric(const short *in, const int *prev_path_metrics, int state) {

    struct pair p;

    int s = (state & ((1 << (CONSTRAINT - 2)) - 1)) << 1;
    int source_state1 = s | 0;
    int source_state2 = s | 1;

    int pm1 = prev_path_metrics[source_state1];
    if (pm1 < INT_MAX) {
        pm1 += branch_metric(in, source_state1, state);
    }
    int pm2 = prev_path_metrics[source_state2];
    if (pm2 < INT_MAX) {
        pm2 += branch_metric(in, source_state2, state);
    }

    if (pm1 <= pm2) {
        p.first = pm1;
        p.second = source_state1;
        return p;
    } else {
        p.first = pm2;
        p.second = source_state2;
        return p;
    }
}
```

```
void update_path_metrics(const short *in, int *path_metrics, int path_size, Trellis trellis, int
index) {

    unsigned i;
    int *new_path_metrics = (int *)malloc(sizeof(int) * path_size);
    int *new_trellis_column = (int *)malloc(sizeof(int) * (1 << (CONSTRAINT - 1)));

    for (i = 0; i < path_size; i++) {
        struct pair p = path_metric(in, path_metrics, i);
        new_path_metrics[i] = p.first;
        new_trellis_column[i] = p.second;
    }

    memcpy(path_metrics, new_path_metrics, (path_size * sizeof * path_metrics));
    memcpy(trellis[index], new_trellis_column, ((1 << (CONSTRAINT - 1)) * sizeof *
new_trellis_column));

    free(new_path_metrics);
    free(new_trellis_column);
}

void viterbi_decoder(const short *in, short *out, unsigned long size) {

    initialize_outputs();

    long i, j, idx = 0;

    Trellis trellis = malloc((size / 2) * sizeof(int *));
    for (i = 0; i < (size / 2); i++) {
        trellis[i] = (int *)malloc(sizeof(int) * (1 << (CONSTRAINT - 1)));
    }
```

```
int *path_metrics = (int *)malloc(sizeof(int) * (1 << (CONSTRAINT - 1)));
path_metrics[0] = 0;
for (i = 1; i < (1 << (CONSTRAINT - 1)); i++) {
    path_metrics[i] = INT_MAX;
}

for (i = 0; i < size; i += PARITY_BITS) {
    short current_bits[PARITY_BITS];
    current_bits[0] = in[i];
    current_bits[1] = (i + 1) >= size ? 0 : in[i + 1];

    update_path_metrics(current_bits, path_metrics, (1 << (CONSTRAINT - 1)), trellis, idx);
    idx++;
}

int state = min_element_index(path_metrics, 0, (1 << (CONSTRAINT - 1)));

for (i = (size / 2) - 1; i >= 0; i--) {
        out[i] = state >> (CONSTRAINT - 2) ? 1 : 0;
       state = trellis[i][state];
}


free(path_metrics);

for (i = 0; i < (size / 2); i++) {
    free(trellis[i]);
}
free(trellis);

for (i = 0; i < (1 << CONSTRAINT); i++) {
```

```
        free(OUTPUTS[i]);
    }
    free(OUTPUTS);
}
```

“simul.c”

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

#include "ofdm.h"
#include "utils.h"

#define MAX_SNR 20

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

    unsigned long i, j;
    unsigned long decoded_ber, uncoded_ber;
    unsigned long n_bytes, n_bits, n_pad, n_sym;
    unsigned mod, alamouti;

    struct OFDM_PARAMETERS params;
    struct timespec start, finish;
    double elapsed_time;

    FILE *fp;
    char filename[32] = "";
```

```
 short *input;
 short *scrambler_output;
 short *encoder_output;
 short *interleaver_output;
fftw_complex *modulation_output;

 // alamouti only
 fftw_complex *tx1;
 fftw_complex *tx2;
 fftw_complex *alamouti_pilots_added_tx1;
 fftw_complex *alamouti_pilots_added_tx2;
 fftw_complex *ifft_input_tx1;
 fftw_complex *ifft_input_tx2;
 fftw_complex *ifft_output_tx1;
 fftw_complex *ifft_output_tx2;
 fftw_complex *cyclic_prefix_tx1;
 fftw_complex *cyclic_prefix_tx2;
 fftw_complex *alamouti_decoder_output;

 // siso only
 fftw_complex *pilots_added_output;
 fftw_complex *ifft_input;
 fftw_complex *ifft_output;
 fftw_complex *cyclic_prefix_output;

 fftw_complex *channel_output;
 fftw_complex *fft_input;
 fftw_complex *fft_output;
 fftw_complex *pilots_remove_input;
 fftw_complex *demodulation_input;
 short *demodulation_output;
```

```
short *deinterleaver_input;
short *deinterleaver_output;
short *decoder_output;
short *descrambler_output;

printf("\nn_bytes: ");
scanf("%lu", &n_bytes);

n_bits = n_bytes * 8;

printf("\nModulation (BPSK : 1 / QPSK : 2): ");
scanf("%u", &mod);

switch (mod) {

   case 1:
      params = get_ofdm_parameters(BPSK);
      strcat(filename, "ber_bpsk");
      break;
   case 2:
      params = get_ofdm_parameters(QPSK);
      strcat(filename, "ber_qpsk");
      break;
   default:
      fprintf(stderr, "Error: invalid modulation\n");
      exit(1);
}

printf("\nAlamouti (Yes : 1 / No : 0): ");
scanf("%u", &alamouti);

if (alamouti != 0 && alamouti != 1) {
```

```
    fprintf(stderr, "Error: invalid value for alamouti\n");
    exit(1);
}

if (alamouti) strcat(filename, "_alamouti.csv");
else strcat(filename, "_siso.csv");

if ((fp = fopen(filename, "w+")) == NULL) {
    fprintf(stderr, "Error: couldn't open file to write\n");
    exit(1);
}

if (n_bits % params.n_cbps != 0) {
    n_pad = params.n_cbps - (n_bits % params.n_cbps);
    n_bits += n_pad;
}
else n_pad = 0;

n_sym = (unsigned long)ceil(n_bits / (double)params.n_dbps);

input = (short *)malloc(sizeof(short) * n_bits);
scrambler_output = (short *)malloc(sizeof(short) * n_bits);
encoder_output = (short *)malloc(sizeof(short) * n_bits * 2);
interleaver_output = (short *)malloc(sizeof(short) * n_bits * 2);
modulation_output = fftw_alloc_complex(N_DATA_SUBCARRIERS);

if (alamouti) {
    tx1 = fftw_alloc_complex(N_DATA_SUBCARRIERS);
    tx2 = fftw_alloc_complex(N_DATA_SUBCARRIERS);
    alamouti_pilots_added_tx1 = fftw_alloc_complex(N_TOTAL_SUBCARRIERS);
    alamouti_pilots_added_tx2 = fftw_alloc_complex(N_TOTAL_SUBCARRIERS);
    ifft_input_tx1 = fftw_alloc_complex(FFT_SIZE);
```

```
    ifft_input_tx2 = fftw_alloc_complex(FFT_SIZE);
    ifft_output_tx1 = fftw_alloc_complex(FFT_SIZE);
    ifft_output_tx2 = fftw_alloc_complex(FFT_SIZE);
    cyclic_prefix_tx1 = fftw_alloc_complex(EXT_OFDM_SYMBOL_SIZE);
    cyclic_prefix_tx2 = fftw_alloc_complex(EXT_OFDM_SYMBOL_SIZE);
    alamouti_decoder_output = fftw_alloc_complex(N_DATA_SUBCARRIERS);
} else {
    pilots_added_output = fftw_alloc_complex(N_TOTAL_SUBCARRIERS);
    ifft_input = fftw_alloc_complex(FFT_SIZE);
    ifft_output = fftw_alloc_complex(FFT_SIZE);
    cyclic_prefix_output = fftw_alloc_complex(EXT_OFDM_SYMBOL_SIZE);
}

channel_output = fftw_alloc_complex(EXT_OFDM_SYMBOL_SIZE);
fft_input = fftw_alloc_complex(FFT_SIZE);
fft_output = fftw_alloc_complex(FFT_SIZE);
pilots_remove_input = fftw_alloc_complex(N_TOTAL_SUBCARRIERS);
demodulation_input = fftw_alloc_complex(N_DATA_SUBCARRIERS);
demodulation_output = (short *)malloc(sizeof(short) * params.n_cbps);
deinterleaver_input = (short *)malloc(sizeof(short) * n_bits * 2);
deinterleaver_output = (short *)malloc(sizeof(short) * n_bits * 2);
decoder_output = (short *)malloc(sizeof(short) * n_bits);
descrambler_output = (short *)malloc(sizeof(short) * n_bits);

if (descrambler_output == NULL) {
    fprintf(stderr, "Error: couldn't allocate memory\n");
    exit(1);
}

if (sysconf(_SC_MONOTONIC_CLOCK) > 0) {
    clock_gettime(CLOCK_MONOTONIC, &start);
}
```

```
    else clock_gettime(CLOCK_REALTIME, &start);

    srand((unsigned)time(NULL));
    for (i = 0; i < n_bits - n_pad; i++) {
        input[i] = gen_binomial_random(0.5);
    }

    if (n_pad > 0) {
        for (i = n_bits - n_pad; i < n_bits; i++) {
            input[i] = 0;
        }
    }

    for (j = 0; j <= MAX_SNR; j += 2) {

        double ebn0 = exp10((double)-1 * j/10);

        short scrambler_init[]   = { 1, 1, 1, 1, 1, 1, 1 };
        short descrambler_init[] = { 1, 1, 1, 1, 1, 1, 1 };

        fftw_complex h1 = { 1.0, 1.0 };
        fftw_complex h2 = { 1.0, 1.0 };

        scrambler(input, scrambler_output, n_bits, scrambler_init);

        encoder(scrambler_output, encoder_output, n_bits);

        interleaver(encoder_output, interleaver_output, params.n_cbps, params.n_bpsc, n_bits *
2);

        for (i = 0; i < n_sym; i++) {
```

```
        mapper(interleaver_output, params.n_cbps, params.modulation, modulation_output, i);

        if (alamouti) {

            alamouti_encoder(modulation_output, tx1, tx2, N_DATA_SUBCARRIERS);

            change_sign(tx1, N_DATA_SUBCARRIERS);
            change_sign(tx2, N_DATA_SUBCARRIERS);

            add_pilots(tx1, alamouti_pilots_added_tx1, i + 1);
            add_pilots(tx2, alamouti_pilots_added_tx2, i + 1);

            map_ofdm_to_ifft(alamouti_pilots_added_tx1, ifft_input_tx1);
            map_ofdm_to_ifft(alamouti_pilots_added_tx2, ifft_input_tx2);

            fixed_point(ifft_input_tx1, ifft_input_tx1, FFT_SIZE);
            fixed_point(ifft_input_tx2, ifft_input_tx2, FFT_SIZE);

            ifft(ifft_input_tx1, ifft_output_tx1);
            ifft(ifft_input_tx2, ifft_output_tx2);

            normalize_ifft_output(ifft_output_tx1, FFT_SIZE, FFT_SIZE);
            normalize_ifft_output(ifft_output_tx2, FFT_SIZE, FFT_SIZE);

            add_cyclic_prefix(ifft_output_tx1, FFT_SIZE, cyclic_prefix_tx1,
EXT_OFDM_SYMBOL_SIZE, CYCLIC_PREFIX_SIZE);
            add_cyclic_prefix(ifft_output_tx2, FFT_SIZE, cyclic_prefix_tx2,
EXT_OFDM_SYMBOL_SIZE, CYCLIC_PREFIX_SIZE);

            defixed_point(cyclic_prefix_tx1, cyclic_prefix_tx1, EXT_OFDM_SYMBOL_SIZE);
            defixed_point(cyclic_prefix_tx2, cyclic_prefix_tx2, EXT_OFDM_SYMBOL_SIZE);
```

```
            alamouti_channel(cyclic_prefix_tx1, cyclic_prefix_tx2, channel_output, h1, h2,
EXT_OFDM_SYMBOL_SIZE, (double)j);

        } else {

            add_pilots(modulation_output, pilots_added_output, i + 1);

            map_ofdm_to_ifft(pilots_added_output, ifft_input);

            fixed_point(ifft_input, ifft_input, FFT_SIZE);

            ifft(ifft_input, ifft_output);

            normalize_ifft_output(ifft_output, FFT_SIZE, FFT_SIZE);

            add_cyclic_prefix(ifft_output, FFT_SIZE, cyclic_prefix_output,
EXT_OFDM_SYMBOL_SIZE, CYCLIC_PREFIX_SIZE);

            defixed_point(cyclic_prefix_output, cyclic_prefix_output,
EXT_OFDM_SYMBOL_SIZE);

            channel(cyclic_prefix_output, channel_output, EXT_OFDM_SYMBOL_SIZE, ebn0);
        }

        fixed_point(channel_output, channel_output, EXT_OFDM_SYMBOL_SIZE);

        remove_cyclic_prefix(channel_output, EXT_OFDM_SYMBOL_SIZE, fft_input,
FFT_SIZE, CYCLIC_PREFIX_SIZE);

        fft(fft_input, fft_output);

        change_sign(fft_output, FFT_SIZE);
```

```
            defixed_point(fft_output, fft_output, FFT_SIZE);

            map_fft_to_ofdm(fft_output, pilots_remove_input);

            remove_pilots(pilots_remove_input, demodulation_input);

            if (alamouti) {
                alamouti_decoder(demodulation_input, alamouti_decoder_output, h1, h2,
N_DATA_SUBCARRIERS);

                demapper(alamouti_decoder_output, N_DATA_SUBCARRIERS, params.modulation,
demodulation_output);

            } else {
                demapper(demodulation_input, N_DATA_SUBCARRIERS, params.modulation,
demodulation_output);
            }

            regrouping(demodulation_output, params.n_cbps, deinterleaver_input, i);
        }

        deinterleaver(deinterleaver_input, deinterleaver_output, params.n_cbps, params.n_bpsc,
n_bits * 2);

        viterbi_decoder(deinterleaver_output, decoder_output, n_bits * 2);

        scrambler(decoder_output, descrambler_output, n_bits, descrambler_init);

        double theory_siso_ber = erfc(sqrt(exp10((double)j/10))) * 0.5;
        double p_alamouti = 0.5 - (0.5 * pow(1.0 + (double)2/j, (double)-0.5));
        double theory_alamouti_ber = pow(p_alamouti, 2.0) * (1 + 2 * (1 - p_alamouti));
        if (j == 0) {
```

```
            printf("\nBER for %s modulation in AWGN with Viterbi decoder (%s)\n", (mod == 1) ?
"BPSK" : "QPSK", (alamouti == 1) ? "nTx=2, nRx=1" : "nTx=1, nRx=1");
            printf("|-------------------------------------------------------------|\n");
            printf("|Eb/N0 | BER (sim) | BER (sim) | BER (siso) | BER (alamouti) |\n");
            printf("| (dB) | (uncoded) | (decoded) |  (theory)  |    (theory)    |\n");
            printf("|-------------------------------------------------------------|\n");

            fprintf(fp, "Eb/N0, BER_uncoded, BER_decoded\n");
        }

        uncoded_ber = ber(encoder_output, deinterleaver_output, n_bits * 2);
        decoded_ber = ber(scrambler_output, decoder_output, n_bits);
        printf("|  %2u  | %lf | %lf   |  %lf  |    %lf    |\n", j, (double)uncoded_ber/(n_bits),
(double)decoded_ber/n_bits, theory_siso_ber, theory_alamouti_ber);
        fprintf(fp, "%u, %lf, %lf\n", j, (double)uncoded_ber/(n_bits), (double)decoded_ber/n_bits);
    }

    printf("|-------------------------------------------------------------|\n");

    if (sysconf(_SC_MONOTONIC_CLOCK) > 0) {
        clock_gettime(CLOCK_MONOTONIC, &finish);
    }
    else clock_gettime(CLOCK_REALTIME, &finish);

    elapsed_time = (finish.tv_sec - start.tv_sec);
    elapsed_time += (finish.tv_nsec - start.tv_nsec) / 1000000000.0;
    printf("\nElapsed time: %.3lf\n", elapsed_time);

    fclose(fp);

    return 0;
}
```