

University of Mississippi

eGrove

Electronic Theses and Dissertations

Graduate School

2012

Analysis of Wavelet Based Alternatives for OFDM

Tassniem Rashed

Follow this and additional works at: <https://egrove.olemiss.edu/etd>



Part of the [Electrical and Computer Engineering Commons](#)

Recommended Citation

Rashed, Tassniem, "Analysis of Wavelet Based Alternatives for OFDM" (2012). *Electronic Theses and Dissertations*. 240.

<https://egrove.olemiss.edu/etd/240>

This Dissertation is brought to you for free and open access by the Graduate School at eGrove. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of eGrove. For more information, please contact egrove@olemiss.edu.

ANALYSIS OF WAVELET BASED ALTERNATIVES FOR OFDM

A Thesis
Presented for the
Master of Science Degree in Engineering
Science with Emphasis in Telecommunications
The University of Mississippi

by

TASSNIEM HUSSAIN RASHED

NOVEMBER 2011

Copyright Tassniem H. Rashed 2011
ALL RIGHTS RESERVED

ABSTRACT

The objective of this thesis is to analyze wavelet based alternatives for orthogonal frequency division multiplexing (OFDM) and find whether a better system performance is achieved when compared to the discrete Fourier transform (DFT)-based OFDM. We analyze DFT, discrete wavelet transform (DWT), and dual tree complex wavelet transform (DT-CWT) based systems in an additive white Gaussian noise (AWGN) channel. The analysis is verified by Monte Carlo simulation. The results in the thesis indicate that the bit error probability (BEP) performance is the same for all types of systems. This confirms some results presented in the literature but differs from others. Some report better BEP performance for the DWT based system than for the DFT based system, and some report worse. In addition, the literature reports better BEP performance for DT-CWT-based system than both DFT-based and DWT-based systems. We compare the peak to average power ratio (PAPR) for the alternatives. The results show improvement in PAPR for the wavelet based system. That is, the DT-CWT performs the best, then the DWT, and the worst is for the DFT based system.

This work is dedicated to my Lord.

As Prophet Ibrahim,

*peace be upon him and all Prophets, said*¹:

(قُلْ إِنَّ صَلَاتِي وَنُسُكِي وَمَحْيَايَ وَمَمَاتِي لِلَّهِ رَبِّ الْعَالَمِينَ،
لَا شَرِيكَ لَهُ وَبِذَلِكَ أُمِرْتُ وَأَنَا أَوَّلُ الْمُسْلِمِينَ)

Qur'an 6:162-163

¹The meaning could be translated as:
(Say, "Indeed, my prayer, my rites, my living and my dying are all for Allah, Lord of the worlds. No partner has He. And this I have been commanded, and I am the first of the Muslims).

LIST OF ABBREVIATIONS

MCM multi-carrier modulation

ISI inter-symbol interference

ICI inter-carrier interference

OFDM orthogonal frequency division multiplexing

DFT discrete Fourier transform

IDFT inverse discrete Fourier transform

FFT fast Fourier transform

IFFT inverse fast Fourier transform

DWT discrete wavelet transform

IDWT inverse discrete wavelet transform

QMF quadrature mirror filters

DT-CWT dual tree complex wavelet transform

IDT-CWT inverse dual tree complex wavelet transform

HT Hilbert transform

FIR finite impulse response

AWGN additive white Gaussian noise

ESD energy spectral density

PAPR peak to average power ratio

CCDF complementary cumulative distribution function

D Daubechies

QAM quadrature amplitude modulation

CW complex wavelet

CWP complex wavelet packet

SNR signal to noise ratio

SEP symbol error probability

BEP bit error probability

V-BLAST vertical Bell Laboratories layered space time

FB filter bank

ACKNOWLEDGEMENTS

“... All praise is due to Allah, who has guided us to this; and we would never have been guided if Allah had not guided us ...”².

I owe my deepest gratitude to my parents, Hussain and Aidah, the ones whom I admire.

I would like to show my gratitude to my brother Hamzeh for his countless and endless support, and to all my siblings.

It is an honor for me to thank my advisor Prof. John Daigle, who has made available his support in many ways.

I am indebted to thank my Professor Dr. Mustafa Matalgah for his effort and support.

I would like to thank my professors, teachers, colleagues, and family members, whom supported me in all aspects of life.

University, Mississippi

Tassniem Hussain Rashed

November 2011

²Quran 7:43

Table of Contents

1	INTRODUCTION	1
1.1	Motivation and Goals	1
1.2	Contributions	1
1.3	State of the Art	2
1.4	Outline of the Thesis	3
2	OFDM ALTERNATIVES	4
2.1	DFT-based OFDM	4
2.1.1	Discrete Fourier Transform	5
2.1.2	DFT-based OFDM System Model	5
2.2	DWT-based OFDM	8
2.2.1	Discrete Wavelet Transform	9
2.2.2	DWT-based OFDM System Model	12
2.3	DT-CWT-based OFDM	17
2.3.1	Dual Tree Complex Wavelet Transform	17
2.3.2	DT-CWT-based OFDM System Model	24
3	COMPARING THE OFDM ALTERNATIVES	27
3.1	Bit Error Probability (BEP) Comparison	27
3.1.1	DFT-based OFDM BEP Performance	27
3.1.2	DWT-based OFDM BEP Performance	30
3.1.3	DT-CWT-based OFDM BEP Performance	33
3.2	Impulse Response and Frequency Response Comparison	37
3.3	Energy Spectral Density (ESD)	45
3.4	Peak to Average Power Ratio (PAPR) Comparison	47
4	CONCLUSIONS	49
	Bibliography	52
A	Simulation code	55
A.1	Bit Error Probability (BEP) Code for OFDM Alternatives	55
A.1.1	FFT-based OFDM.	55

A.1.2	DWT-based OFDM, with Haar filters.	59
A.1.3	DWT-based OFDM, with D-6 filters.	66
A.1.4	DT-CWT-based OFDM, with q-shift filters.	73
A.2	Impulse Response, Frequency Response, and Energy Spectral Density (ESD) Code for OFDM Alternatives	81
A.2.1	FFT-based OFDM.	81
A.2.2	DWT-based OFDM, with Haar filters.	86
A.2.3	DWT-based OFDM, with D-6 filters.	95
A.2.4	DT-CWT-based OFDM, with q-shift filters.	104
A.3	Peak to Average Power Ratio (PAPR) Code for OFDM Alternatives	119

List of Tables

2.1	Filters coefficients: Haar.	14
3.1	Filters coefficients: D-6.	32
3.2	Filters coefficients: q-filters.	36
3.3	Filters coefficients: first stage filters.	36

List of Figures

2.1	FFT-based OFDM system model.	6
2.2	A two-channel filter bank.	9
2.3	Three stage DWT tree structure, analysis.	10
2.4	Three stage IDWT tree structure, synthesis.	10
2.5	DWT-based OFDM system model.	13
2.6	Three stage DT-CWT tree structure, analysis.	18
2.7	Three stage IDT-CWT tree structure, synthesis.	18
2.8	DT-CWT-based OFDM System Model.	24
3.1	BEP, 16-QAM, DFT-based OFDM, Rectangular Pulse Shaping, program in A.1.1.	29
3.2	BEP, 16-QAM, DWT-based OFDM, with Haar filters, and 5 stages, program in A.1.2.	31
3.3	BEP, 16-QAM, DWT-based OFDM, with D-6, and 3 stages, program in A.1.3.	33
3.4	BEP, 16-QAM, DT-CWT-based OFDM, with q-shift filters, and 2 stages, program in A.1.4.	35
3.5	Normalized DFT-based OFDM system response, program in A.2.1.	39

3.6	Normalized DWT-based OFDM system response, with Haar filters, program in A.2.2.	40
3.7	Normalized DWT-based OFDM system response, with D-6 filters, program in A.2.3.	41
3.8	Normalized DT-CWT-based OFDM frequency response, with q-shift filters, program in A.2.4.	42
3.9	Normalized DT-CWT-based OFDM impulse response, with q-shift filters, program in A.2.4.	43
3.10	Normalized DT-CWT-based OFDM amplitude of impulse response, with q-shift filters, program in A.2.4.	44
3.11	Normalized ESD for OFDM alternatives.	46
3.12	$P_{\text{PAPR}} = P\{\text{PAPR} > \text{PAPR}_0\}$ for OFDM alternatives, program in A.3. . . .	48

1. INTRODUCTION

The objective of this thesis is to analyze wavelet based alternatives for orthogonal frequency division multiplexing (OFDM), and find whether a better system performance is achieved when compared to the discrete Fourier transform (DFT)-based OFDM.

1.1 Motivation and Goals

Since the DFT-based OFDM system suffers from high peak to average power ratio (PAPR), two other OFDM alternatives, discrete wavelet transform (DWT)-based, and dual tree complex wavelet transform (DT-CWT)-based, are analyzed to find whether a better system performance is achieved. That is, we show the PAPR, bit error probability (BEP), impulse response, frequency response, and energy spectral density (ESD) performance for all alternatives.

1.2 Contributions

We analyze DFT, DWT, DT-CWT based systems in an additive white Gaussian noise (AWGN) channel. The results are verified by Monte Carlo simulation. The results of the analysis in this thesis indicate that the BEP performance is the same for all types of systems. This confirms some results presented in the literature but differs from others. Some report better BEP performance for DWT-based system than for the DFT-based system, and some

report worse. In addition, the literature reports better BEP performance for DT-CWT-based system than both DFT-based and DWT-based systems. We compare the PAPR for the alternatives. The results show improvement in PAPR for the wavelet based. That is, the DT-CWT performs the best, then the DWT, and the worst is for the DFT based. We study the systems' response.

1.3 State of the Art

The DFT-based OFDM system is described in the literature as a multi-carrier modulation (MCM) scheme. Typically, such a scheme partitions the transmitted datastream into multiple substreams to be sent over multiple subchannels, where these subchannels are orthogonal. Accordingly, the substream data rate is much less than the total rate. Thus the substream bandwidth is much less than the total bandwidth, in order to ensure each subchannel bandwidth being less than the coherence bandwidth of the channel. As a result, the subchannels withstand frequency-selective fading [1].

The literature shows same BEP performance in an AWGN channel for DFT-based OFDM system and in a single channel [1]. The PAPR is proportional to the number of subchannels in the OFDM system for the case of DFT-based system [1].

In [2, 3], the DWT-based OFDM system is described. It is shown numerically that the BEP of both DFT-based and DWT-based OFDM systems perform the same in an AWGN channel. In [4], it is shown that the BEP for DWT-based OFDM system has same performance as the DFT-based OFDM in an AWGN channel. In [5] and [6], the authors report better BEP performance in DWT-based OFDM system than in DFT-based OFDM system. On the other hand, in [7] the authors report worse BEP performance in DWT-based OFDM system than in DFT-based OFDM system, with more than 1 dB difference at 12 dB signal to noise ratio (SNR) per bit.

In [7], the authors propose DT-CWT-based OFDM system, and show better BEP performance than DFT-based and DWT-based systems. The results show more than 3 dB improvement compared to DFT-based and more than 4 dB improvement compared to DWT-based, both at 12 dB SNR per bit.

The authors in [4] present a procedure to reduce the PAPR in DWT-based system by searching better wavelet packet tree structure, they did not compare it to the DFT-based OFDM. In [7], the results show almost same PAPR performance for both DFT-based and DWT-based systems. In the same reference, results show 3dB improvement in PAPR for DT-CWT-based system over DFT-based and DWT-based systems at 0.1% of the complementary cumulative distribution function (CCDF).

In the literature there are other wavelet based systems. In [8], a vertical Bell Laboratories layered space time (V-BLAST)-based OFDM system is proposed. In [9], a complex wavelet (CW)-based OFDM system is proposed. In [10], a complex wavelet packet (CWP)-based OFDM system is described.

1.4 Outline of the Thesis

The thesis is organized as follows. The next chapter presents three different unitary linear transformations to multiplex the symbol stream to form an OFDM system. These transformations are DFT, DWT, and DT-CWT. Furthermore, a system and signal model description for these systems is presented. Chapter 3 is a comparison of the OFDM alternatives. We verify the BEP performance for the alternatives in the 802.11a standard by Monte Carlo simulation. We show the systems' impulse response, frequency response, ESD, and PAPR. Chapter 4 concludes the thesis.

2. OFDM ALTERNATIVES

This chapter describes three OFDM systems proposed in the literature –DFT, DWT, and DT-CWT– in one section each. In each system, a unitary linear transformation is applied to the input data and the difference among the methods is the difference in the transformation.

2.1 DFT-based OFDM

The DFT-based OFDM system is described in the literature as a MCM scheme. Typically, such a scheme partitions the transmitted datastream into multiple substreams to be sent over multiple subchannels, where these subchannels are orthogonal. Accordingly, the substream data rate is much less than the total rate. Thus the substream bandwidth is much less than the total bandwidth, in order to ensure each subchannel bandwidth being less than the coherence bandwidth of the channel. As a result, the subchannels withstand frequency-selective fading [1].

The DFT can efficiently be calculated using fast Fourier transform (FFT). The radix-2 algorithm, for instance, breaks the whole DFT calculation into 2-point DFTs. The computational efficiency is $(N/2)\log_2(N)$ complex multiplications for an N -point FFT and is N^2 for the DFT[11].

In this section, the first subsection presents the DFT. The second subsection presents the system model for the DFT-based OFDM system.

2.1.1 Discrete Fourier Transform

The N -point DFT of a discrete time sequence $x[n], n = 0, 1, \dots, N - 1$, is defined as

$$\text{DFT}\{x[n]\} = X[i] \equiv \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n] e^{-j2\pi ni/N}, \quad i = 0, 1, \dots, N - 1, \quad (2.1.1)$$

where the sequence $x[n]$ can be reconstructed from its DFT by the IDFT

$$\text{IDFT}\{X[i]\} = x[n] \equiv \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} X[i] e^{j2\pi ni/N}, \quad n = 0, 1, \dots, N - 1. \quad (2.1.2)$$

The operation of the DFT in matrix representation is

$$X = Qx, \quad (2.1.3)$$

where $X = [X_0, \dots, X_{N-1}]^T$, $x = [x_0, \dots, x_{N-1}]^T$, the T denotes transpose, Q is a matrix of dimension $N \times N$, and $W_N = \exp(-j2\pi/N)$ is the N^{th} root of unity [12], Q is represented as follows

$$Q = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & W_N & W_N^2 & \dots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \dots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \dots & W_N^{(N-1)^2} \end{bmatrix}. \quad (2.1.4)$$

Since Q is a unitary matrix, $Q^{-1} = Q^H$, the IDFT can be represented as

$$x = Q^{-1}X = Q^H X, \quad (2.1.5)$$

where the H denotes Hermitian transpose.

2.1.2 DFT-based OFDM System Model

The OFDM system was implemented in 1970 as a form of MCM. This took place when the FFT, a simple and less expensive implementation of the DFT, was proposed [13].

The DFT-based OFDM system is illustrated in Figure 2.1. Let X be a vector representation for a stream of N modulated symbols to be transmitted over each of the subchannels, $X = [X_0, X_1, \dots, X_{N-1}]^T$. A serial to parallel conversion is performed to the stream to give the N frequency components of the OFDM symbol. Subsequently, an inverse discrete Fourier transform (IDFT) changes the frequency components into time samples,

$$x = Q^{-1}X, \quad (2.1.6)$$

where vector x represents the stream of symbols that forms one OFDM symbol.

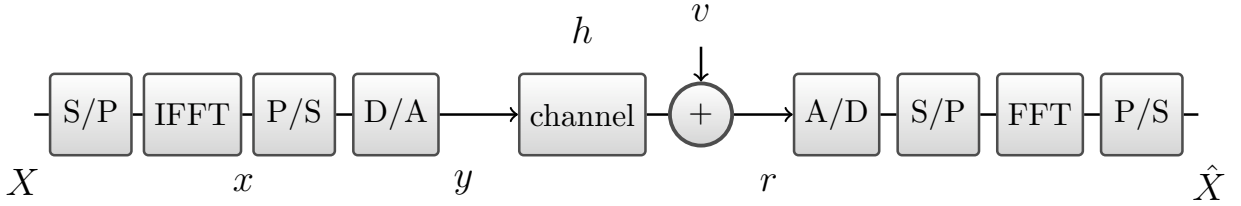


Figure 2.1. FFT-based OFDM system model.

The cyclic prefix is an overhead data stream being added to the data block to eliminate the inter-symbol interference (ISI), which is equivalent to a guard band of a duration of the channel delay spread. Since we are assuming no ISI in our study, we will assume no cyclic prefix is leading the OFDM symbol. Then, the received signal is

$$r = hx + v, \quad (2.1.7)$$

where v is a white Gaussian random vector of zero mean and variance σ_k^2 , for the k^{th} sub-channel.

For the case of assuming the transfer function, $h(t)$, to be a unit step function, then

$$r = x + v. \quad (2.1.8)$$

Due to linearity, the reconstructed vector of symbols can be given as

$$\begin{aligned}
\hat{X} &= Q[x + v] \\
&= QQ^{-1}X + Qv \\
&= X + v_Q,
\end{aligned} \tag{2.1.9}$$

where

$$\begin{aligned}
v_Q &= \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & W_N & W_N^2 & \dots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \dots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \dots & W_N^{(N-1)^2} \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{bmatrix} \\
&= \frac{1}{\sqrt{N}} \begin{bmatrix} \sum_{k=0}^{N-1} v_k \\ \sum_{k=0}^{N-1} v_k W_N^k \\ \sum_{k=0}^{N-1} v_k W_N^{2k} \\ \vdots \\ \sum_{k=0}^{N-1} v_k W_N^{(N-1)k} \end{bmatrix}.
\end{aligned} \tag{2.1.10}$$

Let B_n be the channel bandwidth and B_k be the k^{th} subchannel bandwidth. Then

$$B_n = NB_k. \tag{2.1.11}$$

Since the noise has zero mean, the in-channel noise power for the case of modulated transmitted signal is

$$\begin{aligned}
\sigma_n^2 &= 2 \int_{B_n} S_n(f) df \\
&= 2 \int_{B_n} \frac{\mathcal{N}_0}{2} df
\end{aligned}$$

$$= \mathcal{N}_0 B_n. \quad (2.1.12)$$

Similarly, the subchannel noise power is

$$\sigma_k^2 = \mathcal{N}_0 B_k. \quad (2.1.13)$$

Substitute (2.1.11) into (2.1.12), then the result into (2.1.13),

$$\sigma_k^2 = \sigma_n^2 / N, \quad (2.1.14)$$

Accordingly, each individual subchannel has one N^{th} the in-channel noise power σ_n^2 . As a result, the covariance matrix¹ of the vector v_Q can be represented as

$$\begin{aligned} \text{cov}(v_Q) &= \frac{1}{N} \begin{bmatrix} \text{Var}\left(\sum_{k=0}^{N-1} v_k\right) & \dots & \text{Cov}\left(\sum_{k=0}^{N-1} v_k, \sum_{k=0}^{N-1} v_k W_N^{(N-1)k}\right) \\ \vdots & \ddots & \vdots \\ \text{Cov}\left(\sum_{k=0}^{N-1} v_k, \sum_{k=0}^{N-1} v_k W_N^{(N-1)k}\right) & \dots & \text{Var}\left(\sum_{k=0}^{N-1} v_k W_N^{(N-1)k}\right) \end{bmatrix} \\ &= \frac{1}{N} \begin{bmatrix} N\sigma_n^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & N\sigma_n^2 \end{bmatrix} = \sigma_n^2 I_N, \end{aligned} \quad (2.1.15)$$

where I_N is an $N \times N$ identity matrix. Note here that the output SNR for the k^{th} subchannel is equal to the signal to noise ratio for a single carrier (single channel).

2.2 DWT-based OFDM

The word *wavelet* is defined as a small wave. In signal processing, wavelets are stretched and shifted versions of a basic bandpass wavelet function $\psi(t)$ combined with shifts of a low-pass scaling function, $\phi(t)$ [14]. These functions are continuous functions. If the shifting and scaling performed on the wavelets are continuous, then, the transformation is called

¹The covariance matrix whose (i, j) entry is $\text{Cov}(x_i, y_j) = E[(x_i - E[x_i])(y_j - E[y_j])]$, where $E[\cdot]$ denotes the mean value.

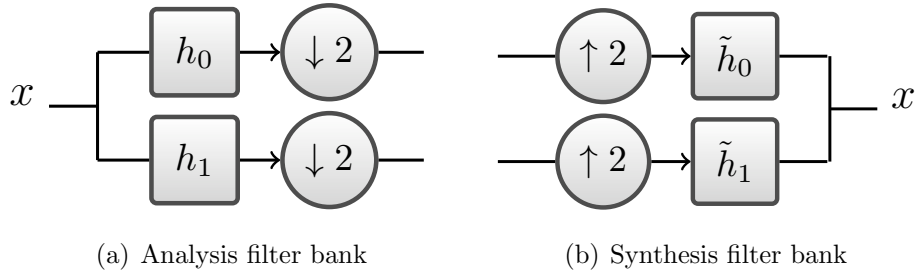


Figure 2.2. A two-channel filter bank.

continuous wavelet transform. Otherwise, if the shifting and scaling are discrete, then, the transform is called discrete. The first and most fundamental wavelet is the Haar basis, known since 1910, when the word wavelet was not used yet. The Haar family² forms an orthonormal basis.

In this section, the first subsection presents the DWT. The second subsection presents the system model for the DWT-based OFDM system.

2.2.1 Discrete Wavelet Transform

A wavelet family can be efficiently implemented via low-pass and high-pass filters forming a filter bank (FB). Accordingly, the wavelet implementation is translated into a filter design problem.

Consider the FB illustrated in Figure 2.2. Multiple consecutive FBs form a DWT tree structure, as shown in Figure 2.3. The filters h_0 and h_1 represent low-pass and high-pass filters, respectively. The symbol $\downarrow 2$ denotes downsampling by two, which means discard all samples with index modulo 2 other than zero [15].

It is clear in the analysis tree structure, in Figure 2.3, that the signal is split into a two channel division with a downsample for each. Then, the low-pass portion is divided by another two channel division with a downsample for each as well, and so on. For the synthesis tree, the inverse discrete wavelet transform (IDWT) shown in Figure 2.4, the

²wavelet and scaling functions.

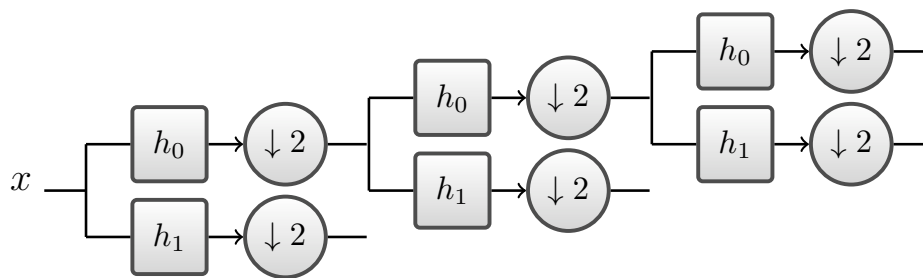


Figure 2.3. Three stage DWT tree structure, analysis.

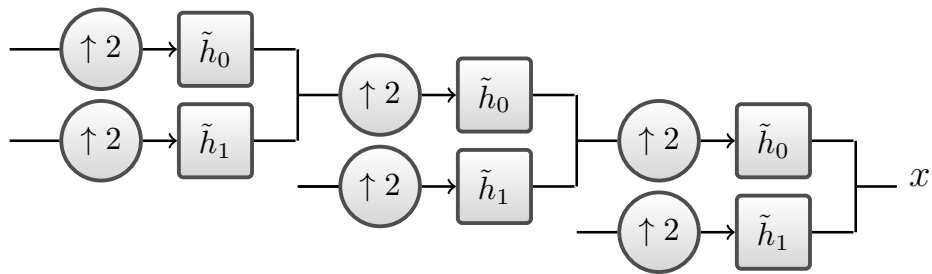


Figure 2.4. Three stage IDWT tree structure, synthesis.

opposite is performed. That is done by upsampling, $\uparrow 2$, which is adding a zero between each sample to get a vector with zeros in its odd samples, then filtering [16]. If the synthesis filters are time inverse of the analysis filters, the transformation is named orthogonal DWT. If not, it is named biorthogonal DWT.

Each stage of an analysis FB can be written in the matrix form as

$$Q_N = \begin{bmatrix} h_0(0) & h_0(1) & h_0(2) & h_0(3) & \dots & 0 & 0 \\ 0 & 0 & h_0(0) & h_0(1) & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ h_0(L-2) & h_0(L-1) & 0 & 0 & \dots & h_0(L-4) & h_0(L-3) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ h_1(0) & h_1(1) & h_1(2) & h_1(3) & \dots & 0 & 0 \\ 0 & 0 & h_1(0) & h_1(1) & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ h_1(L-2) & h_1(L-1) & 0 & 0 & \dots & h_1(L-4) & h_1(L-3) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} L_j \\ H_j \end{bmatrix}, \quad (2.2.1)$$

where L is the filter length, and L_j and H_j denote the low-pass and high-pass circular convolution matrixes, respectively, at the j^{th} stage with a shift by two. The synthesis FB can be written as

$$\tilde{Q}_N^{-1} = \begin{bmatrix} \tilde{h}_0(0) & 0 & \dots & \tilde{h}_0(L-2) & \dots & \tilde{h}_1(0) & 0 & \dots & \tilde{h}_1(L-2) & \dots \\ \tilde{h}_0(1) & 0 & \dots & \tilde{h}_0(L-1) & \dots & \tilde{h}_1(1) & 0 & \dots & \tilde{h}_1(L-1) & \dots \\ \tilde{h}_0(2) & \tilde{h}_0(0) & \dots & 0 & \dots & \tilde{h}_1(2) & \tilde{h}_1(0) & \dots & 0 & \dots \\ \tilde{h}_0(3) & \tilde{h}_0(1) & \dots & 0 & \dots & \tilde{h}_1(3) & \tilde{h}_1(1) & \dots & 0 & \dots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \dots & \ddots \\ 0 & 0 & \dots & \tilde{h}_0(L-4) & \dots & 0 & 0 & \dots & \tilde{h}_1(L-4) & \dots \\ 0 & 0 & \dots & \tilde{h}_0(L-3) & \dots & 0 & 0 & \dots & \tilde{h}_1(L-3) & \dots \end{bmatrix} = \begin{bmatrix} \tilde{L}_j^{-1} & \tilde{H}_j^{-1} \end{bmatrix}, \quad (2.2.2)$$

where \tilde{L}_j^{-1} and \tilde{H}_j^{-1} denote the transpose of the low-pass and high-pass circular convolution

matrixes, respectively, at the j^{th} stage with a shift by two. Both Q_N and \tilde{Q}_N^{-1} are square matrixes with dimension N .

Let the input to the DWT, X , illustrated in Figure 2.3, be a stream of N modulated symbols, $X = [X_0, X_1, \dots, X_{N-1}]^T$, where X_i denotes the i^{th} modulated symbol. A serial to parallel conversion is applied. Then, the N -point DWT of X is defined in matrix form as

$$X = W_N x, \quad (2.2.3)$$

where the sequence X can be reconstructed from its DWT by the IDWT as

$$\tilde{W}_N^{-1} X = \tilde{W}_N^{-1} (W_N x) = x. \quad (2.2.4)$$

with W_N and \tilde{W}_N^{-1} to be in general the transformation matrixes W_R and \tilde{W}_R^{-1} , for a tree structure of J stages and data rate R , then, they can be represented respectively as

$$W_R = \left[\begin{array}{c} \left[\begin{array}{c} \left[\begin{array}{c} \left[\begin{array}{c} L_J \\ H_J \end{array} \right] L_{J-1} \\ \vdots \\ H_{J-1} \end{array} \right] \cdots \\ L_2 \\ L_1 \end{array} \right] \end{array} \right], \quad (2.2.5)$$

and

$$\tilde{W}_R^{-1} = \left[\begin{array}{c} \left[\begin{array}{c} \left[\begin{array}{c} \left[\begin{array}{c} \tilde{L}_1 \\ \tilde{H}_1 \end{array} \right] \tilde{L}_2 \\ \vdots \\ \tilde{H}_2 \end{array} \right] \cdots \\ \tilde{L}_{J-1} \\ \tilde{L}_J \end{array} \right] \end{array} \right]^{-1}. \quad (2.2.6)$$

2.2.2 DWT-based OFDM System Model

The DWT-based OFDM system model was proposed by B. Negash and H. Nikookar in [2, 3]. As we compare the DWT-based and the DFT-based OFDM systems, the former uses

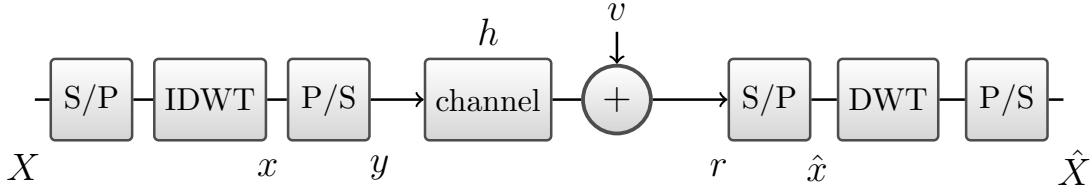


Figure 2.5. DWT-based OFDM system model.

orthogonal or biorthogonal wavelets as carriers, while the latter uses complex exponentials. The DWT-based system is illustrated in Figure 2.5.

Although both systems have the same performance in an AWGN channel [2, 3, 4], each one has its advantages over the other.

On one hand, the DWT-based system is 20% more bandwidth efficient, because the cyclic prefix is not required in the presence of the multipath. In addition to an 8% due to eliminating the pilot tones being used in the DFT-based system, as well as an ISI and inter-carrier interference (ICI) power reduction [4].

On the other hand, DWT suffers from four short comings: oscillation, shift variance, aliasing, and lack of directionality. Oscillation comes from the wavelet coefficients which oscillate around the singularity in a positive and negative way, which makes signal modeling very difficult. Shift variance, that is, a small shift in the signal causes a major difference in the wavelet oscillation at the singularity. Aliasing is due to consecutive non-ideal low-pass and high-pass filtering and up and down sampling. Lack of directionality occurs in more than one dimension [14].

Keep in mind that the DFT implementation does not have these four shortcomings. In the coming section, DT-CWT is presented as a wavelet transform that overcomes these problems.

The only constraint on filters in the DWT is to use orthonormal quadrature mirror

filters (QMF) with perfect reconstruction, for example, Haar and/or Daubechies (D). The QMF is defined as a filter that satisfies $h_1(n) = (-1)^n h_0(L - 1 - n)$, where L is the length of the filter $h_0(n)$. Perfect reconstruction provides the ability to reconstruct the transformed data, that is, $W_R W_R^{-1} = I_R$, where I_R denotes the identity matrix with dimension R [17].

We proceed without loss of generality with Haar low-pass and high-pass filters. Haar filter coefficients are shown in Table 2.1.

Table 2.1. Filters coefficients: Haar.

n	$h_0(n)$	$h_1(n)$	$\tilde{h}_0(n)$	$\tilde{h}_1(n)$
0	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$
1	$\frac{1}{\sqrt{2}}$	$-\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$	$-\frac{1}{\sqrt{2}}$

The transmitted signal $y(t)$ is constructed by transforming the data via an IDWT. Each sub-branch of the tree will transform $R/2^i$ symbols, where R denotes the symbol rate and i denotes the number of stages on that particular sub-branch.

For instance, consider three stages of FBs as being illustrated in Figure 2.4. The first, second, third, and fourth sub-branches transform $R/2^3$, $R/2^3$, $R/2^2$, and $R/2$ symbols, respectively. As a result, $x(t)$ and $y(t)$ will both have a rate of R symbols, which is the same as the rate of X . In matrix form

$$x = \tilde{W}_8^{-1} X, \tag{2.2.7}$$

where

$$\tilde{W}_8^{-1} = \begin{bmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2} & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2} & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2} & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & 0 & \frac{1}{2} & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & 0 & -\frac{1}{2} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & 0 & -\frac{1}{2} & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} \end{bmatrix}. \quad (2.2.8)$$

The received signal $r(t)$ in matrix form is

$$r = hx + v, \quad (2.2.9)$$

where v is a white Gaussian random vector of zero mean and variance σ_n^2 . For the case of assuming the transfer function, $h(t)$, to be a unit step function

$$r = x + v. \quad (2.2.10)$$

Due to linearity, the reconstructed vector of symbols can be given as

$$\begin{aligned} \hat{X} &= W_8[x + v] \\ &= W_8\tilde{W}_8^{-1}X + W_8v \\ &= X + v_W, \end{aligned} \quad (2.2.11)$$

where

$$W_8 = \begin{bmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}, \quad (2.2.12)$$

and

$$v_W = \begin{bmatrix} \frac{1}{2\sqrt{2}} \sum_{k=1}^8 v_k \\ \frac{1}{2\sqrt{2}} \sum_{k=1}^8 v_k \\ \frac{1}{2} \sum_{k=1}^4 v_k \\ \frac{1}{2} \sum_{k=1}^4 v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k \end{bmatrix}. \quad (2.2.13)$$

The covariance matrix of the vector v_W can be represented as

$$\begin{aligned} \text{Cov}(v_W) &= \begin{bmatrix} \text{Var}(\frac{1}{2\sqrt{2}} \sum_{k=1}^8 v_k) & \dots & \text{Cov}(\frac{1}{2\sqrt{2}} \sum_{k=1}^8 v_k, \frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k) \\ \vdots & \ddots & \vdots \\ \text{Cov}(\frac{1}{2\sqrt{2}} \sum_{k=1}^8 v_k, \frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k) & \dots & \text{Var}(\frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k) \end{bmatrix} \\ &= \begin{bmatrix} \sigma_n^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_n^2 \end{bmatrix} = \sigma_n^2 I_8, \end{aligned} \quad (2.2.14)$$

where I_8 is an 8×8 identity matrix.

Note here that the output SNR for the k^{th} symbol is equal to the signal to noise ratio for an AWGN channel.

2.3 DT-CWT-based OFDM

The previous section states that the DT-CWT introduced by Kingsbury in 1998 [18] overcomes the short comings of the DWT. The approach is to implement an analytical wavelet transform, where the DT-CWT is constructed from two real DWTs, one being Hilbert transform (HT) of the other³. In [19], the author shows that wavelet pairs that form a HT pair can be implemented via two tree structures of filters with a half sample delay between them.

In this section, the first subsection presents three different procedures to implement DT-CWT which are proposed in [18], [20], and [21]. The second subsection presents the system model for a DT-CWT-based OFDM system.

2.3.1 Dual Tree Complex Wavelet Transform

The DT-CWT and the inverse dual tree complex wavelet transform (IDT-CWT) tree structures are illustrated in Figures 2.6 and 2.7, respectively, for the case of three stages each. For the analysis tree structure, the filters h_0 and h_1 represent low-pass and high-pass filters for the upper tree, and the filters g_0 and g_1 represent low-pass and high-pass filters for the lower tree, all respectively. For the synthesis tree structure, the filters \tilde{h}_0 and \tilde{h}_1 represent low-pass and high-pass filters for the upper tree, and the filters \tilde{g}_0 and \tilde{g}_1 represent low-pass and high-pass filters for the lower tree, all respectively. It is clear from the figures that the analysis and the synthesis tree structures are constructed from two DWT trees and two IDWT trees, respectively. For the dual tree complex wavelet transform to be analytic, the lower tree structure should give a Hilbert transform (HT) of the upper tree structure. Three procedures chosen from the literature to produce HT pairs are presented in this section.

³Hilbert transform condition:
 $|H_0(e^{j\omega})| = |G_0(e^{j\omega})|,$
 $\angle H_0(e^{j\omega}) = \angle G_0(e^{j\omega}) + 0.5\omega.$

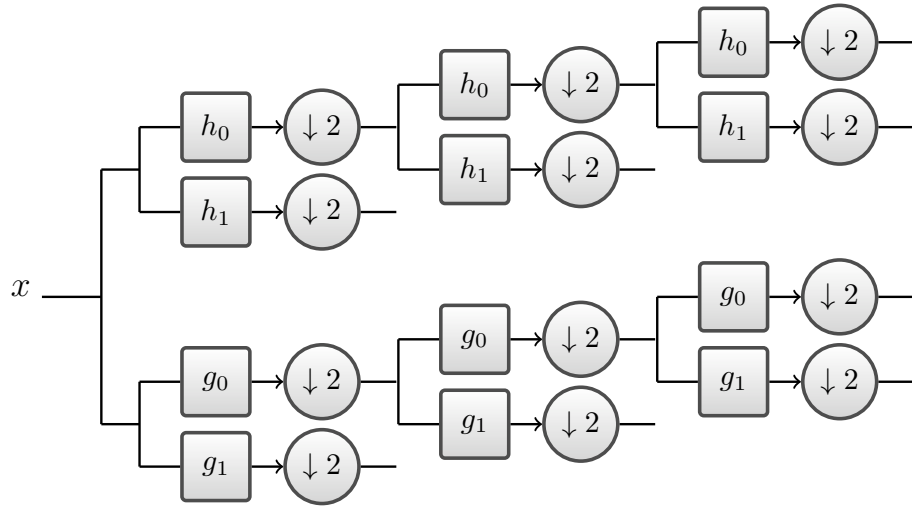


Figure 2.6. Three stage DT-CWT tree structure, analysis.

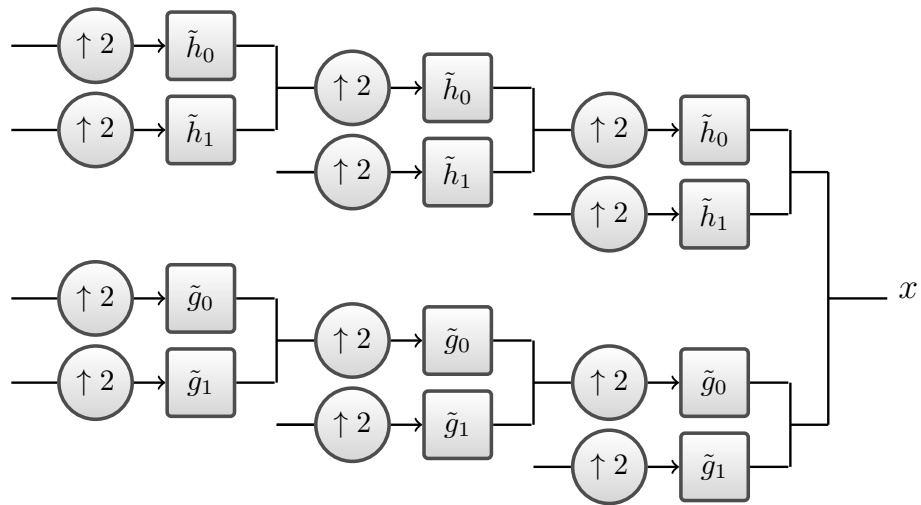


Figure 2.7. Three stage IDT-CWT tree structure, synthesis.

Each stage of an analysis FB for the upper tree can be written in the matrix form as

$$Q_{h_N} = \begin{bmatrix} h_0(0) & h_0(1) & h_0(2) & h_0(3) & \dots & 0 & 0 \\ 0 & 0 & h_0(0) & h_0(1) & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ h_0(L-2) & h_0(L-1) & 0 & 0 & \dots & h_0(L-4) & h_0(L-3) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ h_1(0) & h_1(1) & h_1(2) & h_1(3) & \dots & 0 & 0 \\ 0 & 0 & h_1(0) & h_1(1) & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ h_1(L-2) & h_1(L-1) & 0 & 0 & \dots & h_1(L-4) & h_1(L-3) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} L_{h_j} \\ H_{h_j} \end{bmatrix}, \quad (2.3.1)$$

where L is the filter length, and L_{h_j} and H_{h_j} denote the low-pass and high-pass circular convolution matrixes, respectively, at the j^{th} stage with a shift by two. The analysis FB for the lower tree can be written in the matrix form as

$$Q_{g_N} = \begin{bmatrix} g_0(0) & g_0(1) & g_0(2) & g_0(3) & \dots & 0 & 0 \\ 0 & 0 & g_0(0) & g_0(1) & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ g_0(L-2) & g_0(L-1) & 0 & 0 & \dots & g_0(L-4) & g_0(L-3) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ g_1(0) & g_1(1) & g_1(2) & g_1(3) & \dots & 0 & 0 \\ 0 & 0 & g_1(0) & g_1(1) & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ g_1(L-2) & g_1(L-1) & 0 & 0 & \dots & g_1(L-4) & g_1(L-3) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} L_{g_j} \\ H_{g_j} \end{bmatrix}, \quad (2.3.2)$$

where L is the filter length, and L_{g_j} and H_{g_j} denote the low-pass and high-pass circular convolution matrixes, respectively, at the j^{th} stage with a shift by two.

Let R , J , and j denote the symbol rate, the number of stages in the whole tree structure, and the j^{th} stage, respectively. Let L_j and H_j denote the low-pass and high-pass circular

convolution matrixes with a shift by two, for either upper or lower trees at the analysis side.

Then for either upper or lower transformations W_R can be given as

$$W_R = \left[\begin{array}{c} \left[\begin{array}{c} \left[\begin{array}{c} \left[\begin{array}{c} L_J \\ H_J \end{array} \right] L_{J-1} \\ H_{J-1} \end{array} \right] \cdots \\ \vdots \\ H_2 \\ H_1 \end{array} \right] L_2 \\ L_1 \end{array} \right]. \quad (2.3.3)$$

The synthesis FB for the upper tree can be written as

$$\begin{aligned} \tilde{Q}_{h_N}^{-1} &= \begin{bmatrix} \tilde{h}_0(0) & 0 & \dots & \tilde{h}_0(L-2) & \dots & \tilde{h}_1(0) & 0 & \dots & \tilde{h}_1(L-2) & \dots \\ \tilde{h}_0(1) & 0 & \dots & \tilde{h}_0(L-1) & \dots & \tilde{h}_1(1) & 0 & \dots & \tilde{h}_1(L-1) & \dots \\ \tilde{h}_0(2) & \tilde{h}_0(0) & \dots & 0 & \dots & \tilde{h}_1(2) & \tilde{h}_1(0) & \dots & 0 & \dots \\ \tilde{h}_0(3) & \tilde{h}_0(1) & \dots & 0 & \dots & \tilde{h}_1(3) & \tilde{h}_1(1) & \dots & 0 & \dots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \dots & \ddots \\ 0 & 0 & \dots & \tilde{h}_0(L-4) & \dots & 0 & 0 & \dots & \tilde{h}_1(L-4) & \dots \\ 0 & 0 & \dots & \tilde{h}_0(L-3) & \dots & 0 & 0 & \dots & \tilde{h}_1(L-3) & \dots \end{bmatrix} \\ &= \begin{bmatrix} \tilde{L}_{h_j}^{-1} & \tilde{H}_{h_j}^{-1} \end{bmatrix}, \end{aligned} \quad (2.3.4)$$

where L is the filter length, and \tilde{L}_{h_j} and \tilde{H}_{h_j} denote the low-pass and high-pass circular convolution matrixes, respectively, at the j^{th} stage with a shift by two. The synthesis FB for the lower tree can be written as

$$\begin{aligned} \tilde{Q}_{g_N}^{-1} &= \begin{bmatrix} \tilde{g}_0(0) & 0 & \dots & \tilde{g}_0(L-2) & \dots & \tilde{g}_1(0) & 0 & \dots & \tilde{g}_1(L-2) & \dots \\ \tilde{g}_0(1) & 0 & \dots & \tilde{g}_0(L-1) & \dots & \tilde{g}_1(1) & 0 & \dots & \tilde{g}_1(L-1) & \dots \\ \tilde{g}_0(2) & \tilde{g}_0(0) & \dots & 0 & \dots & \tilde{g}_1(2) & \tilde{g}_1(0) & \dots & 0 & \dots \\ \tilde{g}_0(3) & \tilde{g}_0(1) & \dots & 0 & \dots & \tilde{g}_1(3) & \tilde{g}_1(1) & \dots & 0 & \dots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \dots & \ddots \\ 0 & 0 & \dots & \tilde{g}_0(L-4) & \dots & 0 & 0 & \dots & \tilde{g}_1(L-4) & \dots \\ 0 & 0 & \dots & \tilde{g}_0(L-3) & \dots & 0 & 0 & \dots & \tilde{g}_1(L-3) & \dots \end{bmatrix} \\ &= \begin{bmatrix} \tilde{L}_{g_j}^{-1} & \tilde{H}_{g_j}^{-1} \end{bmatrix}, \end{aligned} \quad (2.3.5)$$

where L is the filter length, and \tilde{L}_{g_j} and \tilde{H}_{g_j} denote the low-pass and high-pass circular convolution matrixes, respectively, at the j^{th} stage with a shift by two.

Let R , J , and j denote the symbol rate, the number of stages in the whole tree structure, and the j^{th} stage, respectively. Let \tilde{L}_j and \tilde{H}_j denote the low-pass and high-pass circular convolution matrixes with a shift by two, for either upper or lower trees at the synthesis side. Then for either upper or lower transformations \tilde{W}_R^{-1} can be given as

$$\tilde{W}_R^{-1} = \left[\left[\left[\left[\left[\begin{array}{c} \tilde{L}_1 \\ \tilde{H}_1 \end{array} \right] \tilde{L}_2 \right] \cdots \right] \tilde{L}_{J-1} \right] \tilde{L}_J \right]^{-1}. \quad (2.3.6)$$

The three procedures chosen from the literature to produce HT pairs from the dual tree are presented below.

Linear-phase biorthogonal procedure

This is the first procedure to be introduced in the literature. In [18], Kingsbury proposed the DT-CWT as a new technique for shift invariant and directional filters. He employs a symmetric low-pass filter of an odd length, denoted earlier as h_0 , and another symmetric low-pass filter of an even length, denoted as g_0 . For L odd:

$$h_0(n) = h_0(L - 1 - n) \quad (2.3.7)$$

$$g_0(n) = g_0(L - n). \quad (2.3.8)$$

This procedure gives biorthogonal set of filters.

From the fact that $h_0(n)$ is a symmetric filter of length N for $n = 0, 1, \dots, N - 1$, and

$g_0(n)$ is a symmetric filter of length $N + 1$ for $n = 0, 1, \dots, N$, it follows that

$$\angle H_0(e^{j\omega}) = -0.5(L - 1)\omega, \quad (2.3.9)$$

$$\angle G_0(e^{j\omega}) = -0.5L\omega. \quad (2.3.10)$$

Consequently, the half sample delay between the trees is satisfied, yet the magnitudes are not the same. Therefore, the filters are better designed so that they have approximately the same magnitude. Moreover, the two trees will be more symmetrical if the odd and even filters are being used alternately from level to level.

Quarter-shift procedure

The second procedure is introduced by Kingsbury in [20]. He employs one filter $h_0(n)$ with asymmetric coefficients of even length, with an envelope delay of a quarter sample period. The filter $h_0(n)$ for one tree, and the filter $g_0(n)$ for the other tree, where $g_0(n)$ is $h_0(n)$'s time inverse. Moreover, the synthesis filters are time inverse of the analysis filters. As a result, they will produce a total envelope delay of half sample period.

In this procedure, the magnitudes of both trees are exactly the same, because the synthesis filters are time inverse of the analysis filters; while the phase is not exactly. Therefore, the filters are designed so that they approximately satisfy the phase condition. Coefficients of filters of various lengths can be found in [20].

Common factor procedure

The third procedure is proposed by Selesnick in [21]. His procedure can be used for both orthogonal and biorthogonal filters. The filters are obtained by setting

$$h_0(n) = f(n) * d(n), \quad (2.3.11)$$

and

$$g_0(n) = f(n) * d(L - n), \quad (2.3.12)$$

where $*$ represents the discrete time convolution, and $d(n)$ is supported on $n = 0, 1, \dots, L$. In terms of z -transforms,

$$H_0(z) = F(z)D(z), \quad (2.3.13)$$

and

$$G_0(z) = F(z)z^{-L}D(1/z). \quad (2.3.14)$$

Let $A(z)$ be an all-pass transfer function⁴ defined as

$$A(z) := \frac{z^{-L}D(1/z)}{D(z)}, \quad (2.3.15)$$

then

$$G_0(z) = H_0(z)A(z), \quad (2.3.16)$$

accordingly

$$|G_0(e^{j\omega})| = |H_0(e^{j\omega})|. \quad (2.3.17)$$

Then $D(z)$ is chosen as a finite impulse response (FIR) to have

$$\angle A(e^{j\omega}) \approx -0.5\omega. \quad (2.3.18)$$

Finally, choose $F(z)$ as an FIR to have $h_0(n)$ and $g_0(n)$ meet the perfect reconstruction conditions. Coefficients of filters of various lengths can be found in [21].

⁴For an all-pass transfer function $A(e^{j\omega})$, $|A(e^{j\omega})| = 1$.

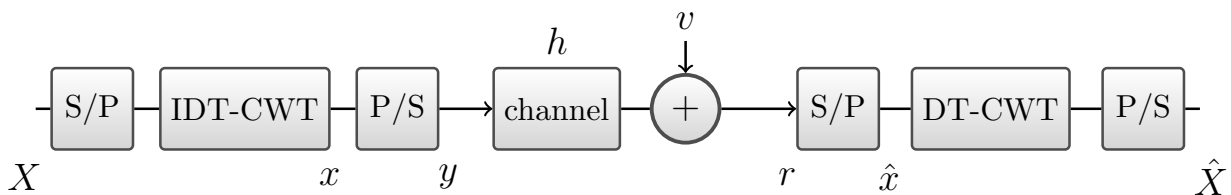


Figure 2.8. DT-CWT-based OFDM System Model.

2.3.2 DT-CWT-based OFDM System Model

When we compare the DWT-based and the DT-CWT-based OFDM systems, the latter uses two real DWTs, one being HT of the other. Figure 2.8 illustrates the system model, where the DT-CWT and the IDT-CWT blocks are shown in Figures 2.6 and 2.7, respectively.

In matrix form, the output of the upper and the lower tree structure is

$$x = \begin{bmatrix} x_h \\ x_g \end{bmatrix} = \begin{bmatrix} \tilde{W}_{hN}^{-1} \\ \tilde{W}_{gN}^{-1} \end{bmatrix} X, \quad (2.3.19)$$

where x_h and x_g are the output vectors of the upper and the lower tree structures, respectively, while X is a column vector of N modulated symbols to denote the input word, \tilde{W}_{hN}^{-1} and \tilde{W}_{gN}^{-1} are given in equation (2.3.6).

The received signal $r(t)$ in matrix form is

$$r = hx + v, \quad (2.3.20)$$

where v is a white Gaussian random column vector of N elements for each tree, with all elements of zero mean and variance σ_n^2 . For the case of assuming the transfer function, $h(t)$, to be a unit step function, then

$$r = x + v. \quad (2.3.21)$$

Due to linearity, the reconstructed vector of symbols can be given as

$$\begin{aligned}
\hat{X} &= \begin{bmatrix} W_{hN} & W_{gN} \end{bmatrix} (x + v) \\
&= \begin{bmatrix} W_{hN} & W_{gN} \end{bmatrix} \left(\begin{bmatrix} \tilde{W}_{hN}^{-1} \\ \tilde{W}_{gN}^{-1} \end{bmatrix} X + \begin{bmatrix} v_h \\ v_g \end{bmatrix} \right) \\
&= \begin{bmatrix} I_N & I_N \end{bmatrix} X + v_W \\
&= X + v_W,
\end{aligned} \tag{2.3.22}$$

where v_h and v_g are the noise vectors of the upper and the lower tree structures, W_{hN} and W_{gN} are given in equation (2.3.3). Furthermore

$$\begin{aligned}
v_W &= \begin{bmatrix} v_{Wh} & v_{Wg} \end{bmatrix} \\
&= \begin{bmatrix} W_{hN}v_h & W_{gN}v_g \end{bmatrix}.
\end{aligned} \tag{2.3.23}$$

As an example, if $N = 8$, or in other words, the tree structures are three stages each, then,

$$W_{hN}v_h = \begin{bmatrix} \frac{1}{2\sqrt{2}} \sum_{k=1}^8 v_k \\ \frac{1}{2\sqrt{2}} \sum_{k=1}^8 v_k \\ \frac{1}{2} \sum_{k=1}^4 v_k \\ \frac{1}{2} \sum_{k=1}^4 v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k \end{bmatrix}, \tag{2.3.24}$$

and

$$W_{gN}v_g = \begin{bmatrix} \frac{1}{2\sqrt{2}} \sum_{k=1}^8 v_k \\ \frac{1}{2\sqrt{2}} \sum_{k=1}^8 v_k \\ \frac{1}{2} \sum_{k=1}^4 v_k \\ \frac{1}{2} \sum_{k=1}^4 v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^2 v_k \end{bmatrix}. \quad (2.3.25)$$

Then the upper and the lower set of reconstructed symbols can be averaged to evaluate the final set of symbols. The covariance matrix for the final set of symbols can be written as

$$\begin{aligned} \text{Cov}\left(\frac{v_{Wh} + v_{Wg}}{2}\right) &= \frac{1}{2^2} \begin{bmatrix} \text{Var}\left(\frac{2}{2\sqrt{2}} \sum_{k=1}^8 v_k\right) & \dots & \text{Cov}\left(\frac{2}{2\sqrt{2}} \sum_{k=1}^8 v_k, \frac{2}{\sqrt{2}} \sum_{k=1}^2 v_k\right) \\ \vdots & \ddots & \vdots \\ \text{Cov}\left(\frac{2}{2\sqrt{2}} \sum_{k=1}^8 v_k, \frac{2}{\sqrt{2}} \sum_{k=1}^2 v_k\right) & \dots & \text{Var}\left(\frac{2}{\sqrt{2}} \sum_{k=1}^2 v_k\right) \end{bmatrix} \\ &= \begin{bmatrix} \sigma_n^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_n^2 \end{bmatrix} = \sigma_n^2 I_8, \end{aligned} \quad (2.3.26)$$

where I_8 is an 8×8 identity matrix.

Note here that the output SNR for the k^{th} symbol is equal to the signal to noise ratio for an AWGN channel.

3. COMPARING THE OFDM ALTERNATIVES

This chapter presents a comparison of the three OFDM systems presented in the earlier chapter. The first section presents the BEP performance for these three systems. The second, third, and fourth sections present the systems' impulse response, frequency response, ESD, and CCDF for the PAPR, respectively.

3.1 Bit Error Probability (BEP) Comparison

To compare analytical and numerical BEP performance, we choose $N = 64$ subchannels, where each subchannel uses 16-quadrature amplitude modulation (QAM) as in the case of IEEE 802.11a, and no overhead. Then, the symbol stream of 64 symbols form one OFDM symbol.

In this section, the first, second, and third subsections present the BEP performance for the systems DFT-based OFDM, DWT-based OFDM, and DT-CWT-based OFDM, respectively.

3.1.1 DFT-based OFDM BEP Performance

In a DFT-based OFDM system, the symbol error probability (SEP) performance on the k^{th} subchannel is the same performance of a single channel M-QAM system. If we assume a

unity channel gain with an AWGN with variance σ_n^2 , and N subchannels, then the SEP, P_{s_k} , for the k^{th} subchannel is [13]

$$P_{s_k} = \left[4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3\gamma_{s_k}}{M-1}} \right) - 4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right)^2 Q^2 \left(\sqrt{\frac{3\gamma_{s_k}}{M-1}} \right) \right], \quad (3.1.1)$$

where γ_{s_k} denotes the SNR per symbol on the k^{th} subchannel.

If we assume gray coding, where adjacent codes differ by one bit, then any symbol in error will cause exactly one bit to be in error [1]. For the M-QAM constellation, the BEP for the k^{th} subchannel is

$$\begin{aligned} P_{b_k} &\simeq \frac{1}{\log_2 M} P_{s_k} \\ &= \frac{1}{\log_2 M} \left[4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3\gamma_{s_k}}{M-1}} \right) - 4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right)^2 Q^2 \left(\sqrt{\frac{3\gamma_{s_k}}{M-1}} \right) \right] \\ &= \frac{1}{\log_2 M} \left[4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3 \log_2 M \gamma_{b_k}}{M-1}} \right) - 4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right)^2 Q^2 \left(\sqrt{\frac{3 \log_2 M \gamma_{b_k}}{M-1}} \right) \right], \end{aligned} \quad (3.1.2)$$

where $\gamma_{s_k} = \log_2 M \gamma_{b_k}$, and γ_{b_k} denotes the SNR per bit on the k^{th} subchannel.

The error performance for an OFDM system is evaluated by averaging over all subchannels [22, 23, 24, 25]. As a result, the average BEP for $N = 64$ subchannels is

$$\begin{aligned}
P_b &= \frac{1}{N} \sum_{k=1}^N P_{b_k} \\
&= \frac{1}{N} N \frac{1}{\log_2 M} \left[4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3 \log_2 M \gamma_{b_k}}{M-1}} \right) \right. \\
&\quad \left. - 4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right)^2 Q^2 \left(\sqrt{\frac{3 \log_2 M \gamma_{b_k}}{M-1}} \right) \right] \\
&= P_{b_k}.
\end{aligned} \tag{3.1.3}$$

Notice Figure 3.1 shows the analytical and simulation results assuming 16-QAM with rectangular pulse shaping.

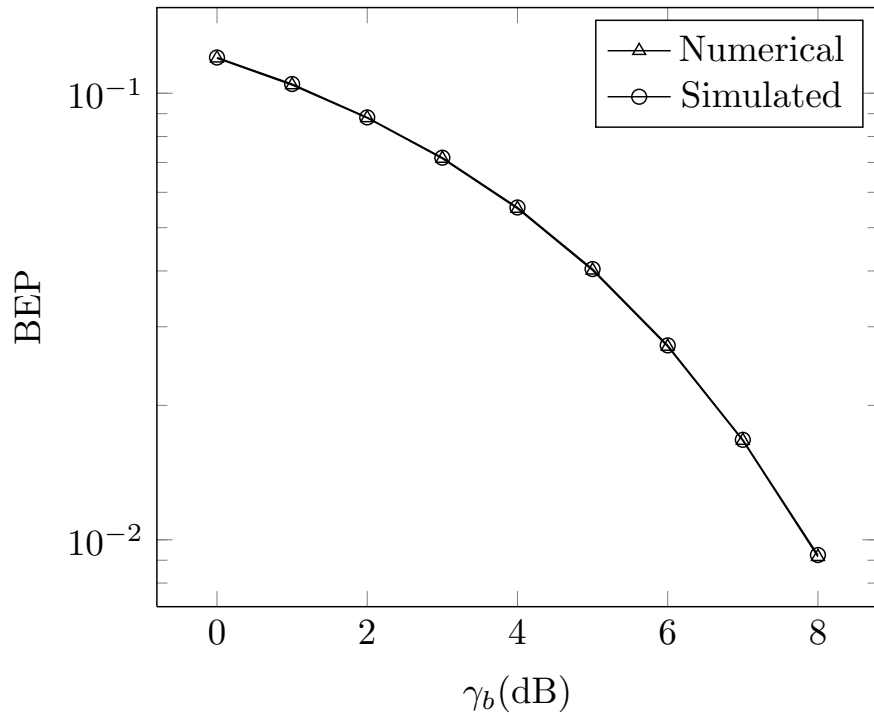


Figure 3.1. BEP, 16-QAM, DFT-based OFDM, Rectangular Pulse Shaping, program in A.1.1.

The BEP performance of each subchannel matches the BEP performance of a single AWGN channel.

3.1.2 DWT-based OFDM BEP Performance

In a DWT-based OFDM system, if we assume QAM modulated symbols, then the DWT is applied on both streams of the QAM symbols in parallel. Then, each transformation is transmitted on a carrier in quadrature with the other.

The SEP performance of the k^{th} symbol is the same performance of a single AWGN channel. If we assume a unity channel gain with an AWGN of variance σ_n^2 , and B branches, then the SEP, P_{s_k} , for the k^{th} QAM symbol is [13]

$$P_{s_k} = \left[4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3\gamma_{s_k}}{M-1}} \right) - 4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right)^2 Q^2 \left(\sqrt{\frac{3\gamma_{s_k}}{M-1}} \right) \right], \quad (3.1.4)$$

where γ_{s_k} denotes the SNR per symbol for the k^{th} QAM symbol.

If we assume gray coding for the M-QAM constellation, then the BEP for the k^{th} QAM symbol is

$$\begin{aligned} P_{b_k} &\simeq \frac{1}{\log_2 M} P_{s_k} \\ &= \frac{1}{\log_2 M} \left[4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3\gamma_{s_k}}{M-1}} \right) - 4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right)^2 Q^2 \left(\sqrt{\frac{3\gamma_{s_k}}{M-1}} \right) \right] \\ &= \frac{1}{\log_2 M} \left[4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3 \log_2 M \gamma_{b_k}}{M-1}} \right) - 4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right)^2 Q^2 \left(\sqrt{\frac{3 \log_2 M \gamma_{b_k}}{M-1}} \right) \right], \end{aligned} \quad (3.1.5)$$

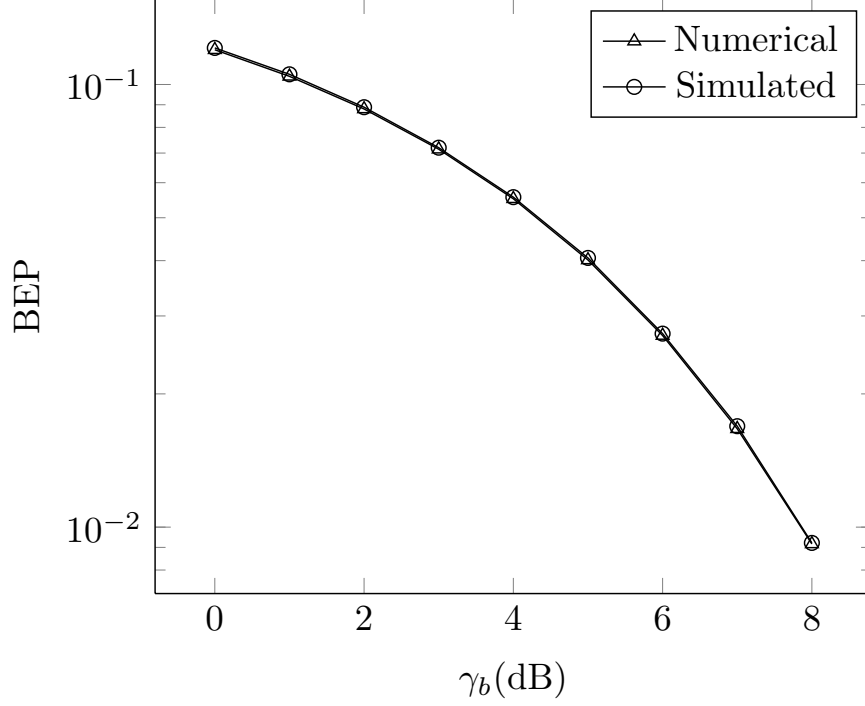


Figure 3.2. BEP, 16-QAM, DWT-based OFDM, with Haar filters, and 5 stages, program in A.1.2.

where $\gamma_{s_k} = \log_2 M \gamma_{b_k}$, and γ_{b_k} denotes the SNR per bit for the k^{th} QAM symbol.

As a result, the average BEP for $N = 64$ symbols is

$$\begin{aligned}
P_b &= \frac{1}{N} \sum_{k=1}^N P_{b_k} \\
&= \frac{1}{N} N \frac{1}{\log_2 M} \left[4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3 \log_2 M \gamma_{b_k}}{M-1}} \right) \right. \\
&\quad \left. - 4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right)^2 Q^2 \left(\sqrt{\frac{3 \log_2 M \gamma_{b_k}}{M-1}} \right) \right] \\
&= P_{b_k}.
\end{aligned} \tag{3.1.6}$$

Notice Figures 3.2 and 3.3 show the analytical form corresponding to the numerical results assuming 16-QAM with rectangular pulse shaping. The former employs Haar filters with five stages, while the latter employs D-6 filters with three stages. D-6 coefficients are shown in Table 3.1. Where D-6 is orthonormal QMF with perfect reconstruction. Both low-pass and high-pass filters are of length 6 [26].

Table 3.1. Filters coefficients: D-6.

n	$h_0(n)$	$h_1(n)$
0	0.332671	0.035226
1	0.806892	0.085441
2	0.459878	-0.135011
3	-0.135011	-0.459878
4	-0.085441	0.806892
5	0.035226	-0.332671

The BEP performance of each QAM symbol matches the BEP performance of a single AWGN channel. Thus, the DWT-based and the DFT-based OFDM systems have equal BEP performance in an AWGN channel.

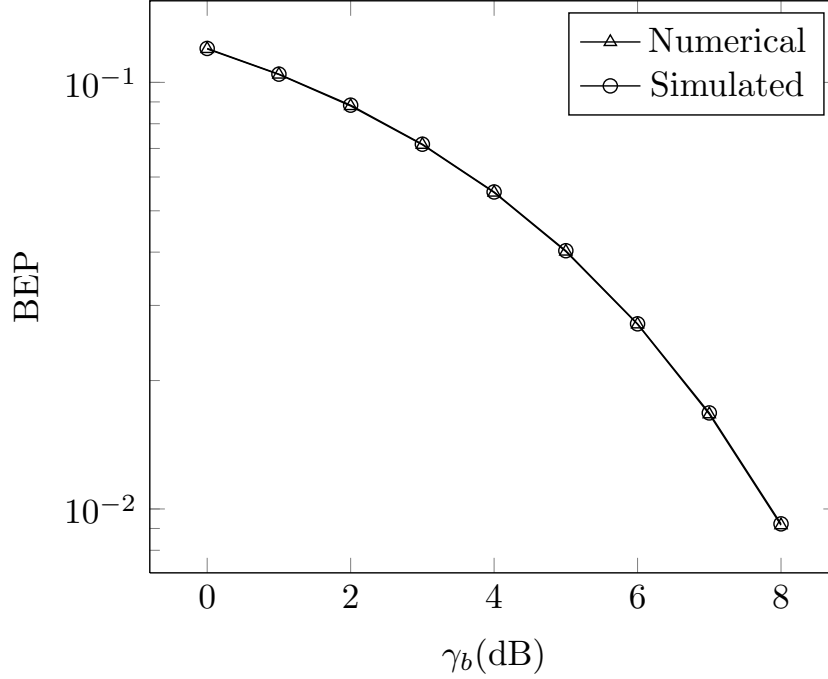


Figure 3.3. BEP, 16-QAM, DWT-based OFDM, with D-6, and 3 stages, program in A.1.3.

3.1.3 DT-CWT-based OFDM BEP Performance

In a DT-CWT-based OFDM system, if we assume QAM modulated symbols, then the DT-CWT is applied on both streams of the QAM symbols in parallel. Then, each transformation is transmitted on a carrier in quadrature with the other. Since the inphase and quadrature streams contain the same information, there is $2\times$ redundancy.

The SEP performance of the k^{th} symbol QAM is the same performance of a single AWGN channel. If we assume a unity channel gain with an AWGN of variance σ_n^2 , and B branches for each tree, then the SEP, P_{s_k} , for the k^{th} QAM symbol is [13]

$$\begin{aligned}
 P_{s_k} &= \left[4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3\gamma_{s_k}}{M-1}} \right) \right. \\
 &\quad \left. - 4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right)^2 Q^2 \left(\sqrt{\frac{3\gamma_{s_k}}{M-1}} \right) \right], \tag{3.1.7}
 \end{aligned}$$

where γ_{s_k} denotes the SNR per symbol for the k^{th} QAM symbol.

If we assume gray coding for the M-QAM constellation, then the BEP for the k^{th} QAM symbol is

$$\begin{aligned}
P_{b_k} &\simeq \frac{1}{\log_2 M} P_{s_k} \\
&= \frac{1}{\log_2 M} \left[4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3 \gamma_{s_k}}{M-1}} \right) \right. \\
&\quad \left. - 4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right)^2 Q^2 \left(\sqrt{\frac{3 \gamma_{s_k}}{M-1}} \right) \right] \\
&= \frac{1}{\log_2 M} \left[4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3 \log_2 M \gamma_{b_k}}{M-1}} \right) \right. \\
&\quad \left. - 4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right)^2 Q^2 \left(\sqrt{\frac{3 \log_2 M \gamma_{b_k}}{M-1}} \right) \right], \tag{3.1.8}
\end{aligned}$$

where $\gamma_{s_k} = \log_2 M \gamma_{b_k}$, and γ_{b_k} denotes the SNR per bit for the k^{th} QAM symbol.

As a result, the average BEP for $N = 64$ symbols is

$$\begin{aligned}
P_b &= \frac{1}{N} \sum_{k=1}^N P_{b_k} \\
&= \frac{1}{N} N \frac{1}{\log_2 M} \left[4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3 \log_2 M \gamma_{b_k}}{M-1}} \right) \right. \\
&\quad \left. - 4 \left(\frac{\sqrt{M}-1}{\sqrt{M}} \right)^2 Q^2 \left(\sqrt{\frac{3 \log_2 M \gamma_{b_k}}{M-1}} \right) \right] \\
&= P_{b_k}. \tag{3.1.9}
\end{aligned}$$

In a DT-CWT structure, the first and last stages of the analysis and synthesis tree structures, respectively, should be different than the other stages [14]. In our simulation,

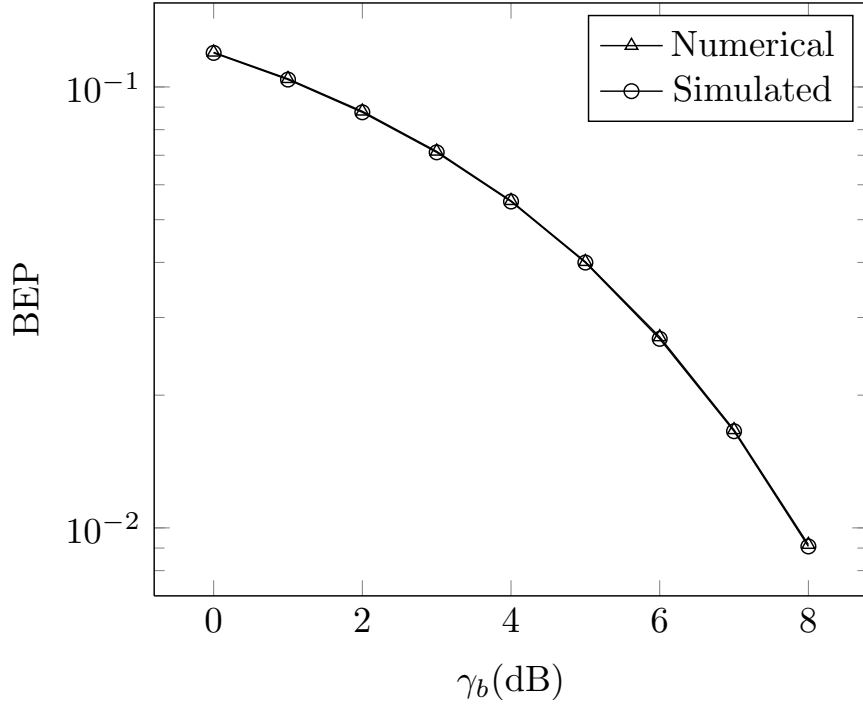


Figure 3.4. BEP, 16-QAM, DT-CWT-based OFDM, with q-shift filters, and 2 stages, program in A.1.4.

both trees employ q-shift filters for the stages other than the first stage for the analysis structure and the last stage for the synthesis structure. The remaining stages employ the same filters as the other stages but with specific alignments with respect to each other. The final stage filters are inverse of the first stage filters. Tables 3.2 and 3.3 show the q-shift filters and the first stage filters, respectively [27].

Notice Figure 3.4 shows the analytical results corresponding to the numerical results assuming 16-QAM with rectangular pulse shaping. The BEP performance of each QAM symbol matches the BEP performance of a single AWGN channel. As a result, it is clear that the DT-CWT-based, DWT-based, and DFT-based OFDM systems all have the same BEP performance in an AWGN channel.

Table 3.2. Filters coefficients: q-filters.

n	$h_0(n)$	$h_1(n)$
0	0.0351638	0.0
1	0.0	0.0
2	-0.0883294	-0.1143018
3	0.2338903	0.0
4	0.7602724	0.5875183
5	0.5875183	-0.7602724
6	0.0	0.2338903
7	-0.1143018	0.0883294
8	0.0	0.0
9	0.0	-0.0351638

Table 3.3. Filters coefficients: first stage filters.

n	$h_0(n)$	$h_1(n)$
0	0.0	0.0
1	-0.0883883	-0.0112268
2	0.0883883	0.0112268
3	0.6958800	0.0883883
4	0.6958800	0.0883883
5	0.0883883	-0.6958800
6	-0.0883883	0.6958800
7	0.0112268	-0.0883883
8	0.0112268	-0.0883883
9	0.0	0.0

3.2 Impulse Response and Frequency Response Comparison

This section presents the normalized responses of the three OFDM alternatives. We have chosen the word length $N = 64$. The system response for the DFT-Based OFDM system is obtained by setting the input of the inverse fast Fourier transform (IFFT) to a vector of all ones. Then, the output is the impulse response of the system. The FFT of the output is the frequency response, which is the input signal itself. On the other hand, the impulse response and the frequency response for the wavelet based systems are obtained by setting an arbitrary coefficient of the word to one, and all others to zero, and then doing an inverse transform on the word. We have chosen the 20th coefficient of the word to be one. Notice here that the impulse response shows the response of the system as a function of time, which is the wavelet for a tree structure, while the frequency response shows the response of the system as a function of frequency.

In Figure 3.5, the DFT-based OFDM system response is illustrated. The system's frequency response is illustrated in Figures 3.5(a) and 3.5(b). It is flat and real. In Figures 3.5(c) and 3.5(d), the system's impulse response is illustrated. Figure 3.5(e) shows the amplitude of the system's impulse response.

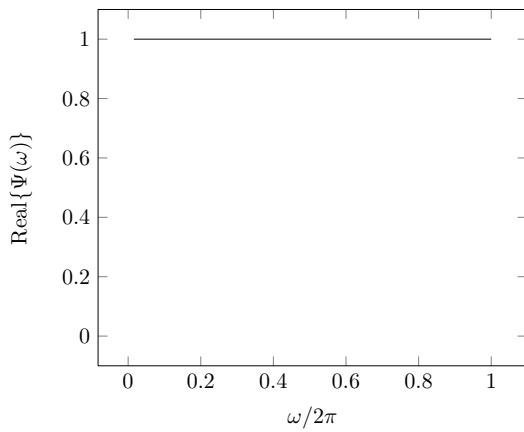
In Figure 3.6, the DWT-based OFDM system response with Haar filters is illustrated. The system's frequency response is illustrated in Figures 3.6(a) and 3.6(b). It is double sided. In Figures 3.6(c) and 3.6(d), the system's impulse response is illustrated. Figure 3.6(e) shows the amplitude of the system's impulse response.

In Figure 3.7, the DWT-based OFDM system response with D-6 filters is illustrated. The system's frequency response is illustrated in Figures 3.7(a) and 3.7(b). It is double sided. In Figures 3.7(c) and 3.7(d), the system's impulse response is illustrated. Figure 3.7(e) shows the amplitude of the system's impulse response.

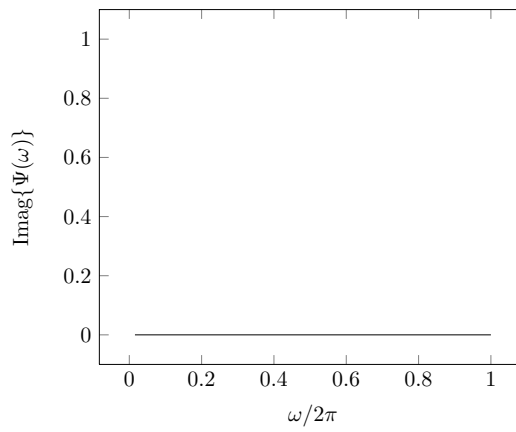
In Figure 3.8, the DT-CWT-based OFDM system frequency response with q-shift filters is illustrated. The system's upper tree frequency response is illustrated in Figures 3.8(a) and Figure 3.8(b). It is double sided. The system's lower tree frequency response is illustrated in Figures 3.8(c) and Figure 3.8(d). It is double sided. The system's dual tree frequency response is illustrated in Figures 3.8(e) and Figure 3.8(f). It is single sided.

In Figure 3.9, the DT-CWT-based OFDM system impulse response with q-shift filters is illustrated. The system's upper tree impulse response is illustrated in Figures 3.9(a) and 3.9(b). The system's lower tree impulse response is illustrated in Figures 3.9(c) and 3.9(d). The system's dual tree impulse response is illustrated in Figures 3.9(e) and 3.9(f).

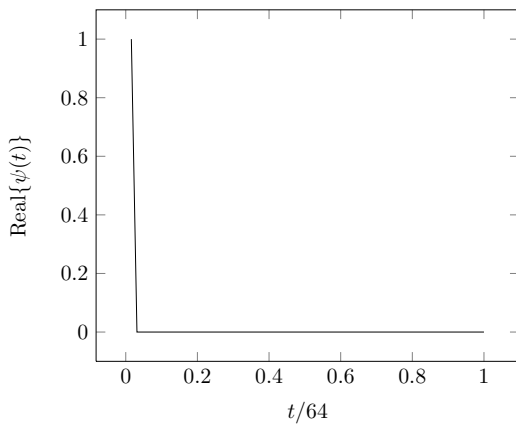
In Figure 3.10, the amplitude of the system's upper tree and lower tree impulse response are illustrated in Figures 3.10(a) and 3.10(b), respectively. Figure 3.10(c) shows the amplitude of the system's dual tree impulse response.



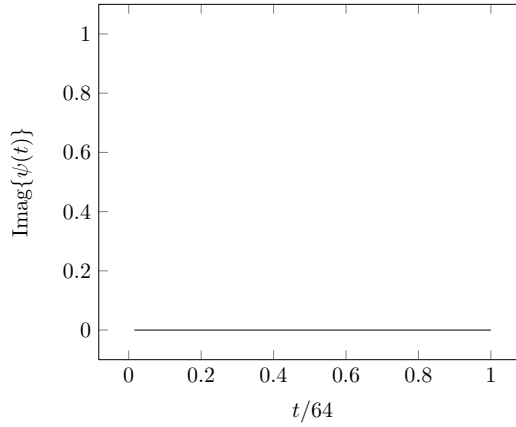
(a) System's frequency response, real



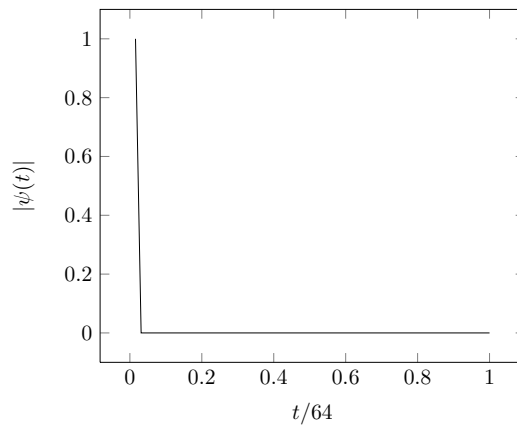
(b) System's frequency response, imaginary



(c) System's impulse response, real

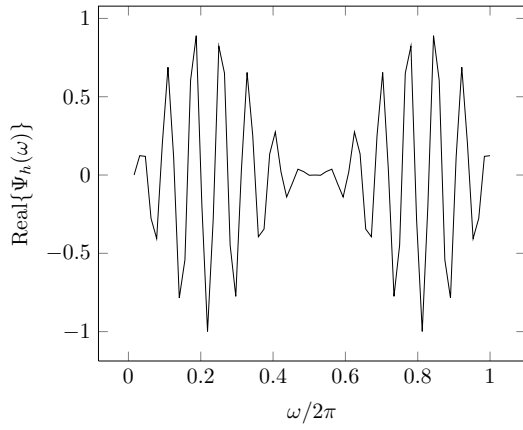


(d) System's impulse response, imaginary

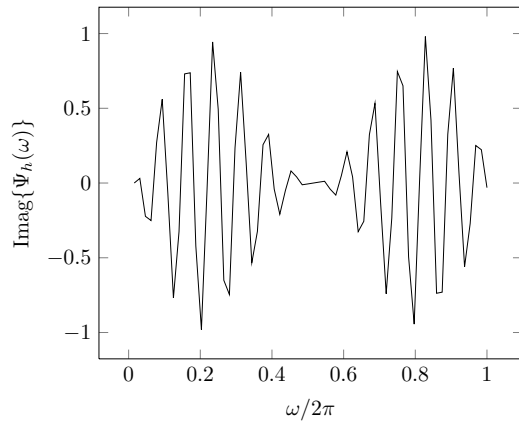


(e) Amplitude of system's impulse response

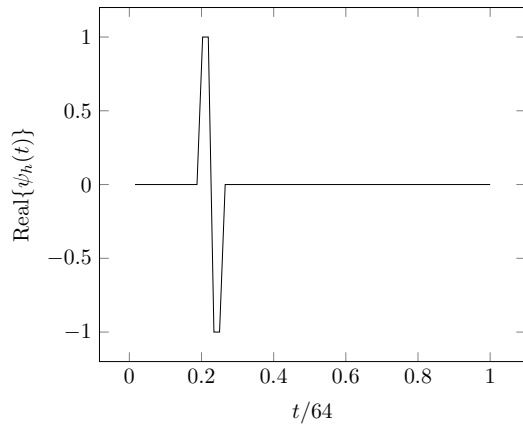
Figure 3.5. Normalized DFT-based OFDM system response, program in A.2.1.



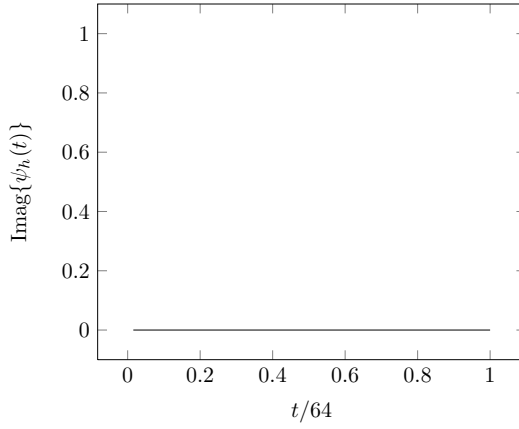
(a) System's frequency response, real



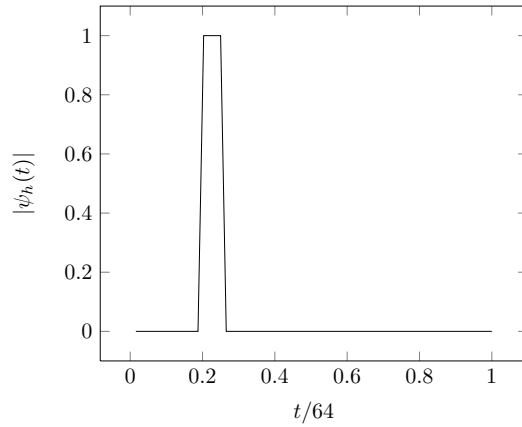
(b) System's frequency response, imaginary



(c) System's impulse response, real

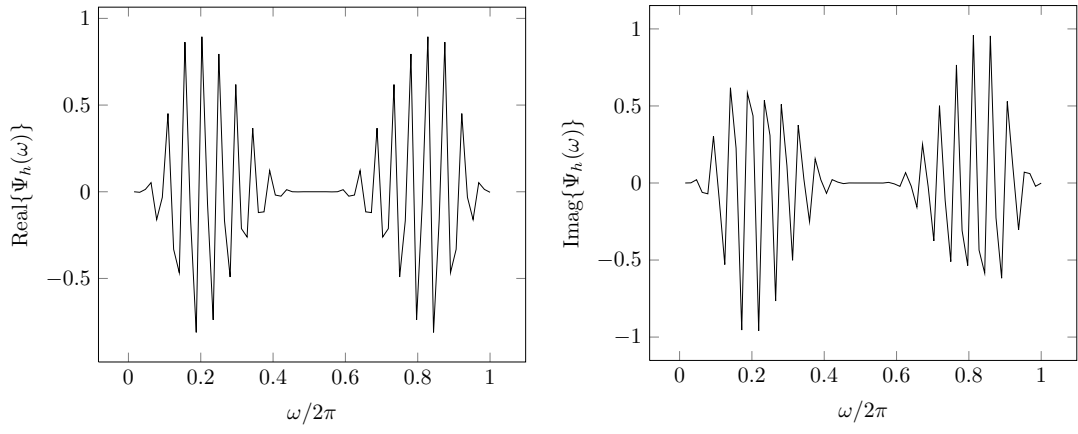


(d) System's impulse response, imaginary

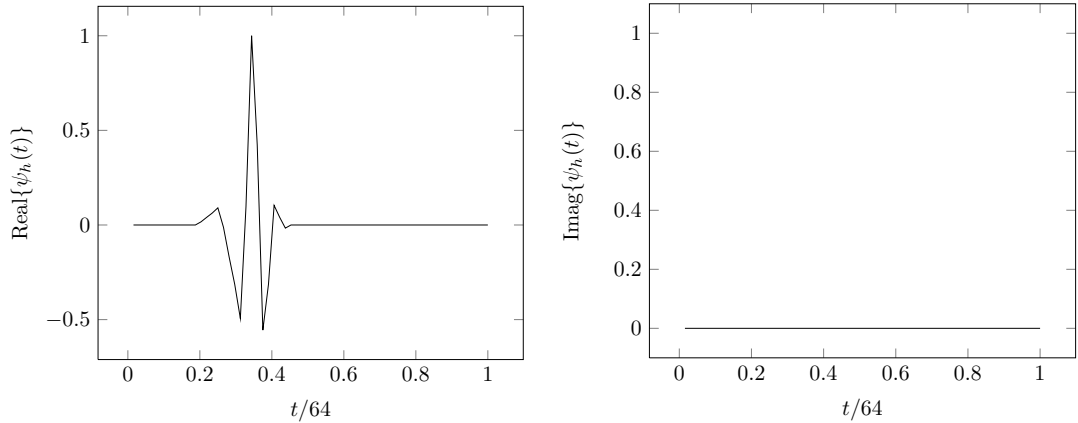


(e) Amplitude of system's impulse response

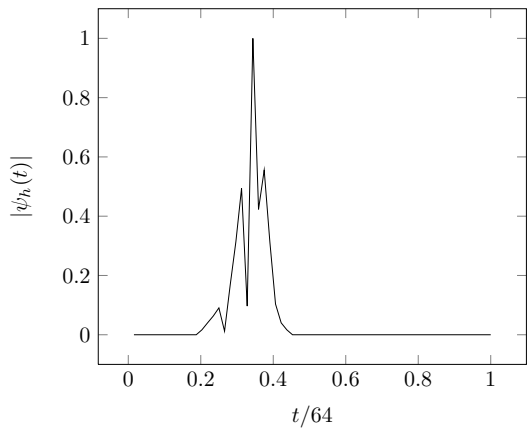
Figure 3.6. Normalized DWT-based OFDM system response, with Haar filters, program in A.2.2.



(a) System's frequency response, real (b) System's frequency response, imaginary

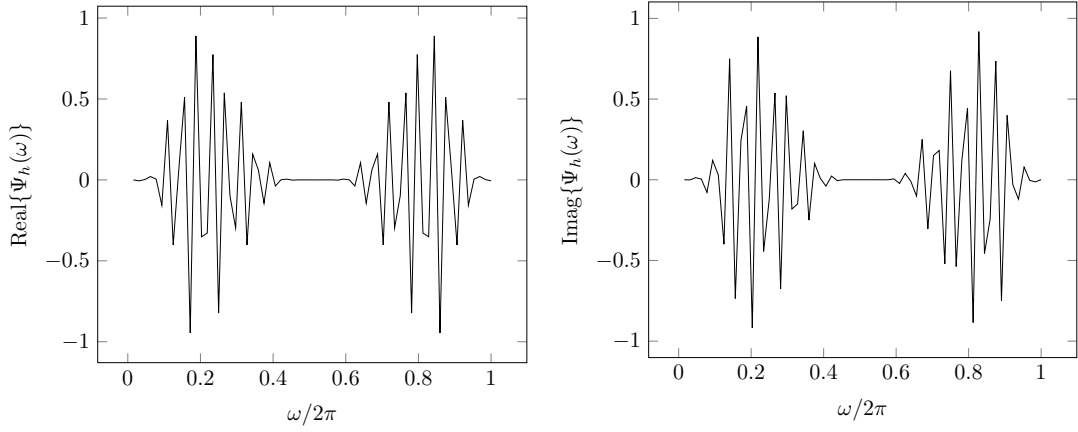


(c) System's impulse response, real (d) System's impulse response, imaginary

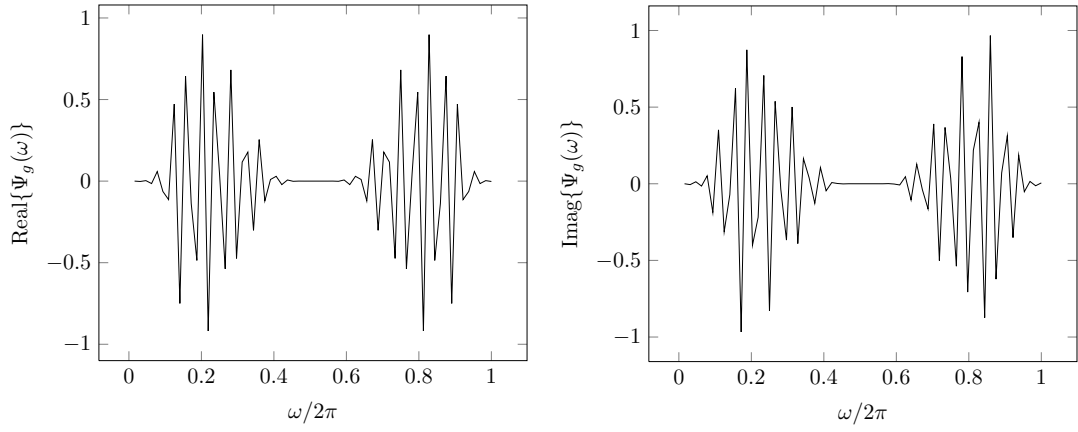


(e) Amplitude of system's impulse response

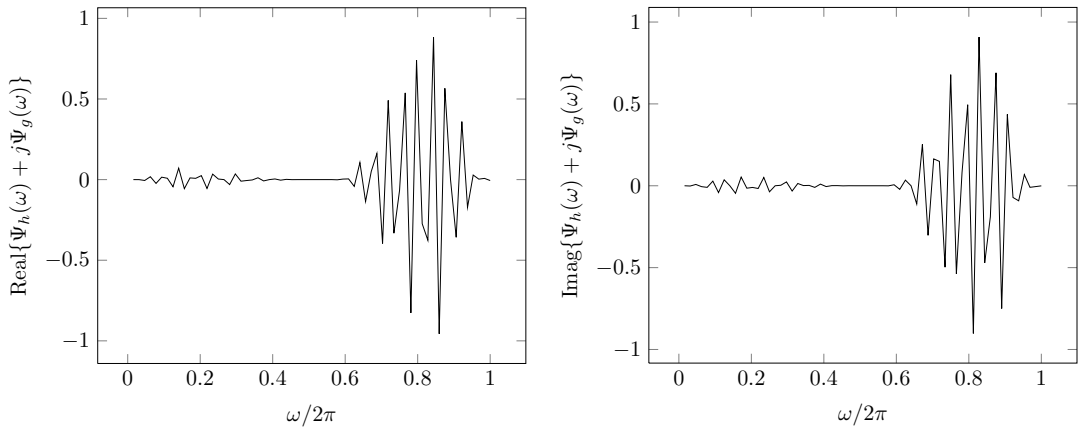
Figure 3.7. Normalized DWT-based OFDM system response, with D-6 filters, program in A.2.3.



(a) System's upper tree frequency response, real (b) System's upper tree frequency response, imaginary

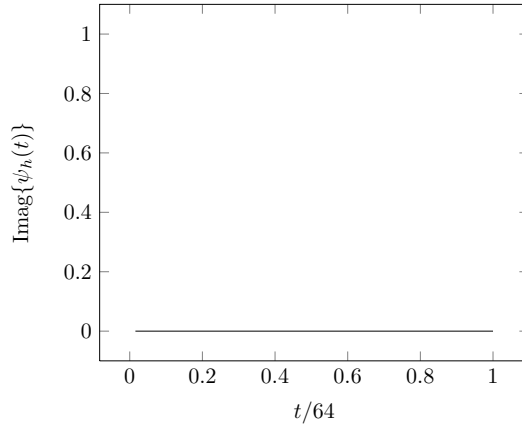
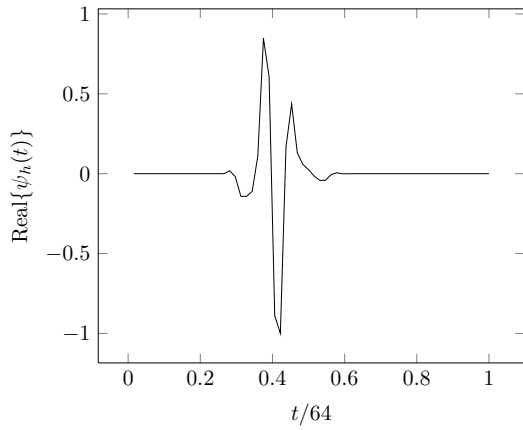


(c) System's lower tree frequency response, real (d) System's lower tree frequency response, imaginary

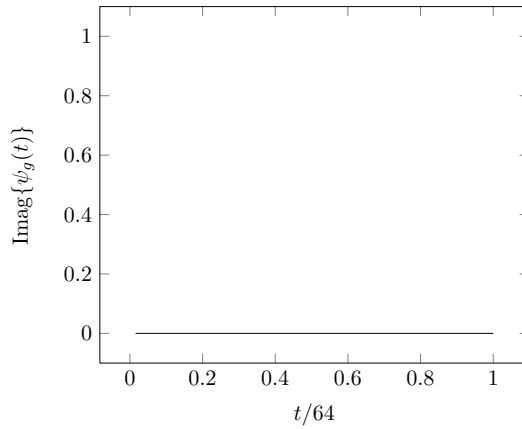
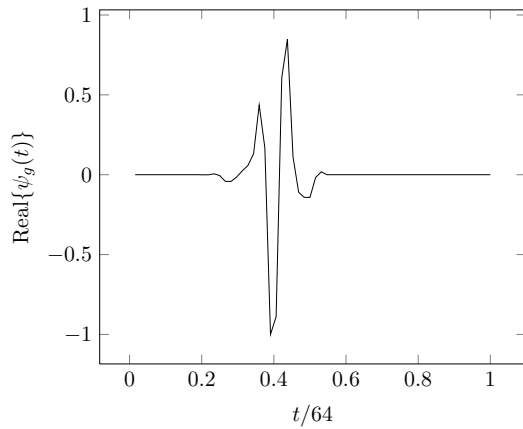


(e) System's dual tree frequency response, real (f) System's dual tree frequency response, imaginary

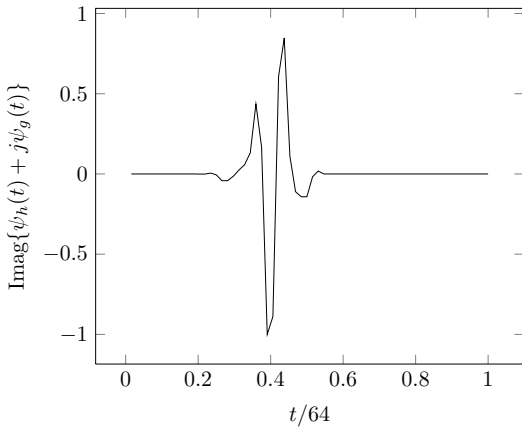
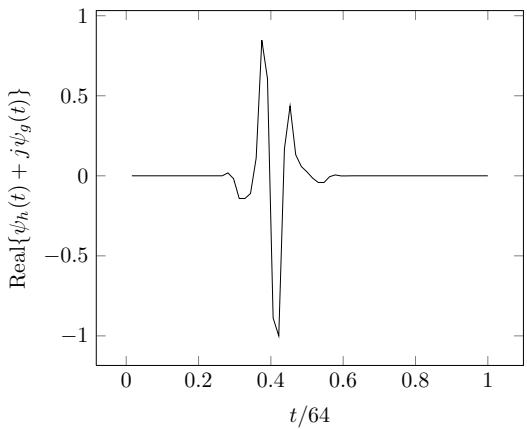
Figure 3.8. Normalized DT-CWT-based OFDM frequency response, with q-shift filters, program in A.2.4.



(a) System's upper tree impulse response, real (b) System's upper tree impulse response, imaginary

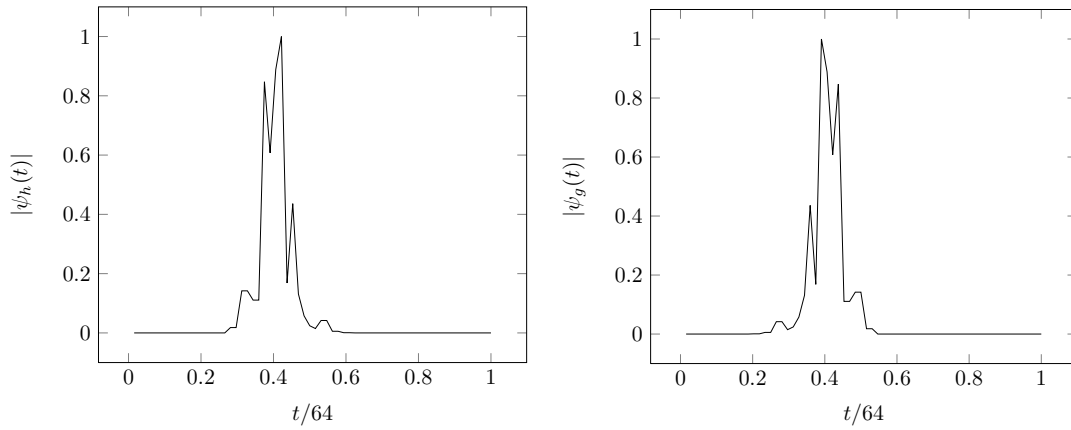


(c) System's lower tree impulse response, real (d) System's lower tree impulse response, imaginary

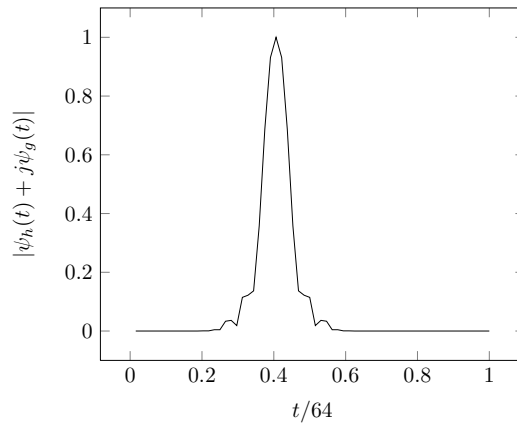


(e) System's dual tree impulse response, real (f) System's dual tree impulse response, imaginary

Figure 3.9. Normalized DT-CWT-based OFDM impulse response, with q-shift filters, program in A.2.4.



(a) Amplitude of system's upper tree impulse response (b) Amplitude of system's lower tree impulse response



(c) Amplitude of system's dual tree impulse response

Figure 3.10. Normalized DT-CWT-based OFDM amplitude of impulse response, with q-shift filters, program in A.2.4.

3.3 Energy Spectral Density (ESD)

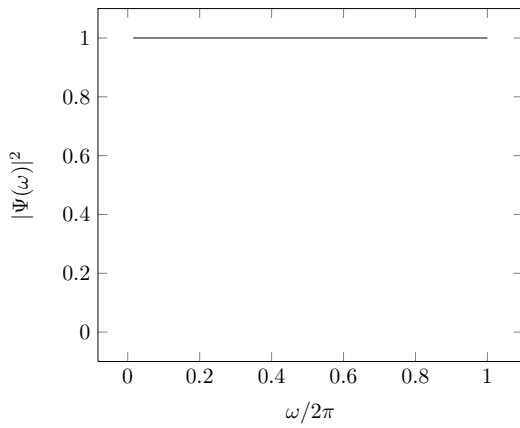
The ESD shows the energies for all frequency components of the signal. It is defined for a signal $f(t)$ as [28]

$$\text{ESD}\{f(\omega)\} = |F(\omega)|^2, \quad (3.3.1)$$

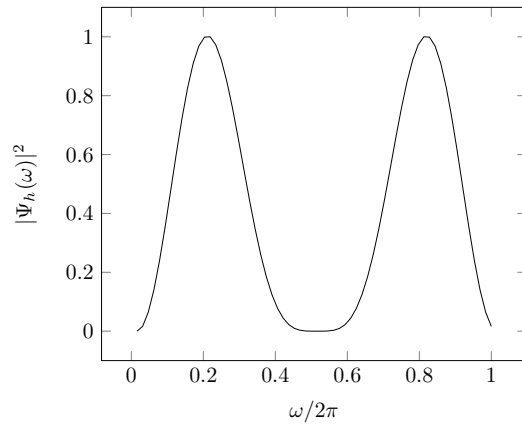
where $|F(\omega)|$ is the absolute value of the Fourier transform of $f(t)$.

Figure 3.11 shows the normalized ESD for the outputs of DFT, DWT, and DT-CWT-based OFDM systems. The results are obtained by squaring the absolute value of the frequency response of the systems.

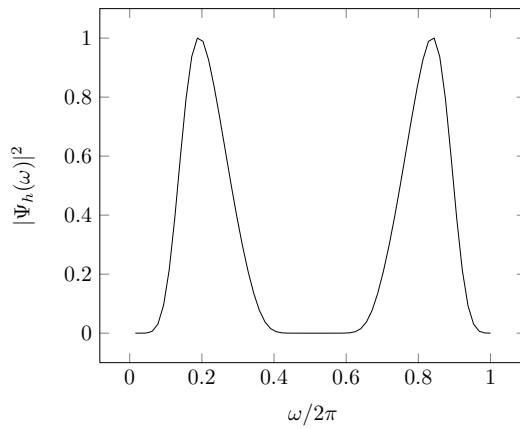
In Figure 3.11(a), the ESD for the DFT-based system is flat. In Figures 3.11(b), 3.11(c), 3.11(d) and 3.11(e), the DWT with Haar filters, the DWT with the D-6 filters, the upper tree of the DT-CWT, and the lower tree of the DT-CWT, respectively, give a double sided ESD. When the previous four ESD Figures are compared, it is clear that as the filter lengths are closer the ESDs are more alike. However, in figure 3.11(f), it is shown that the ESD is single sided for the DT-CWT, because it is constructed from $2\times$ redundancy, where one is Hilbert transform of the other.



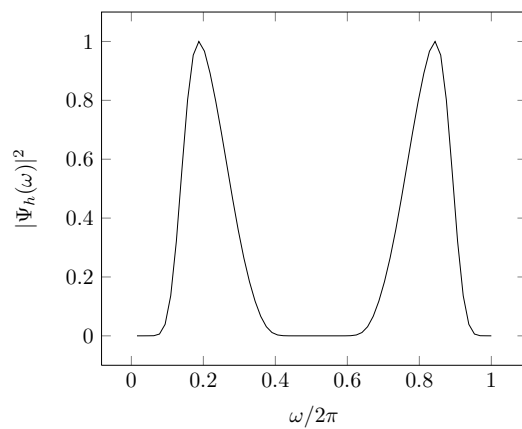
(a) ESD for DFT-based OFDM, program in A.2.1



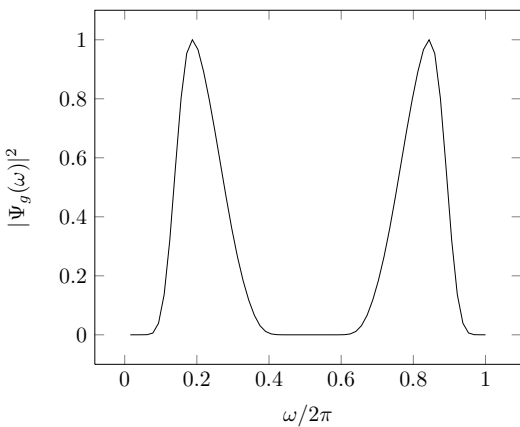
(b) ESD for DWT-based OFDM, Haar filters, program in A.2.2



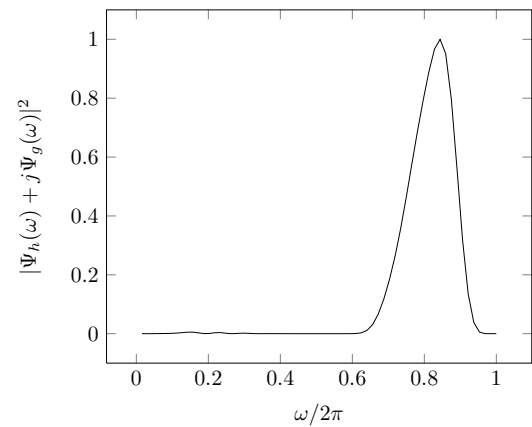
(c) ESD for DWT-based OFDM, D-6 filters, program in A.2.3



(d) ESD for DT-CWT-based OFDM, upper tree, program in A.2.4



(e) ESD for DT-CWT-based OFDM, lower tree, program in A.2.4



(f) ESD for DT-CWT-based OFDM, dual tree, program in A.2.4

Figure 3.11. Normalized ESD for OFDM alternatives.

3.4 Peak to Average Power Ratio (PAPR) Comparison

The peak to average power ratio is an important factor in communication systems. That is, at the transmitter side, the power amplifier responses linearly before starting to move into a nonlinear region. To avoid signal distortion, operation in the linear region is needed [1]. The PAPR can be defined as [29]

$$\text{PAPR(dB)} = 10 \log_{10} \left(\frac{\text{Peak Power}}{\text{Average Power}} \right). \quad (3.4.1)$$

Accordingly, the PAPR is better to be close to one. That is, the peak to be as close as possible to the average.

Comparison of PAPRs is accomplished via the complementary cumulative distribution function (CCDF) of the PAPR;

$$P_{\text{PAPR}} = P\{\text{PAPR} > \text{PAPR}_0\}, \quad (3.4.2)$$

where P_{PAPR} is the probability that the PAPR is larger than a value PAPR_0 .

Figure 3.12 shows the P_{PAPR} for DFT, DWT, and DT-CWT-based OFDM systems based on averaging 3×10^5 OFDM symbols. The least P_{PAPR} is for the DT-CWT-based system compared to the DFT-based and the DWT-based OFDM systems. The P_{PAPR} for the DWT-based with D-6 is close to the upper and the lower tree structures when considering them individually, because their filters length is relatively close, while Haar filters have length two. It is clear from the figure that the DFT case yields the worst PAPR, while the DT-CWT yields the best. It follows from the fact that, the maximum PAPR for the DFT-based is proportional to the number of subcarriers as they add constructively. On the other hand, the PAPR for the DT-CWT-based is lower, due to the fact that the average power of the upper and the lower trees are added linearly.

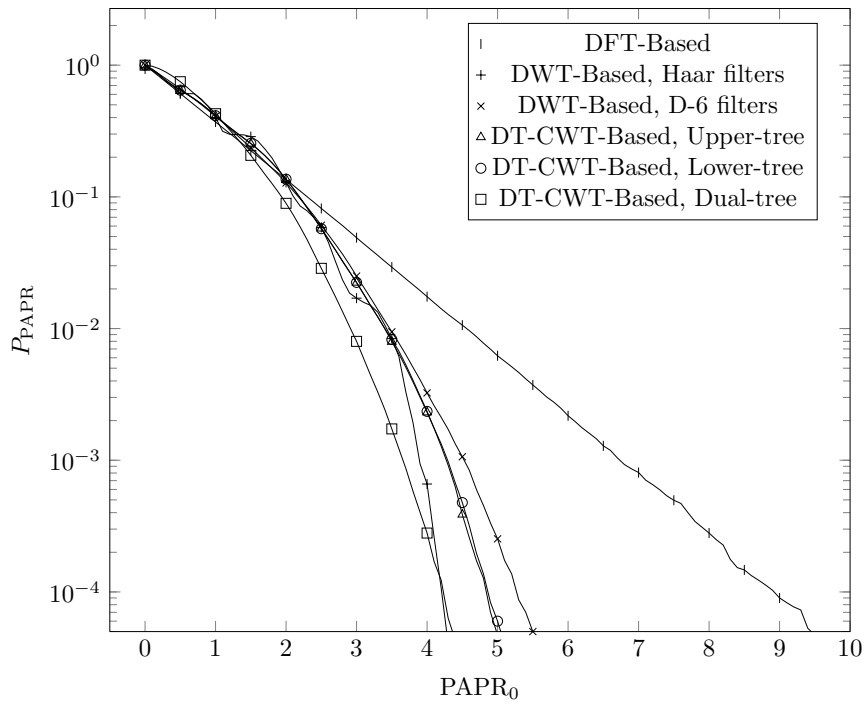


Figure 3.12. $P_{\text{PAPR}} = P\{\text{PAPR} > \text{PAPR}_0\}$ for OFDM alternatives, program in A.3.

4. CONCLUSIONS

This thesis was an attempt to verify some promising BEP performance results in the literature for the DT-CWT-based OFDM system in an AWGN channel, and to extend the study in fading channels. Unfortunately, we were unable to produce any configuration of a DT-CWT-based OFDM system that results in a 3 dB improvement in SNR. In addition, we were unable to establish correspondence with the authors of the papers in order to clarify the claims made in the papers.

Throughout the thesis, a development of a solid knowledge in several areas such as system analysis, digital signal processing, filters, filter banks, wavelets, wavelet transform, data transform, up and down sampling, Hilbert pairs, Hilbert transform, orthogonal transform, biorthogonal transform, OFDM, system performance, probability evaluation, BEP performance, complementary cumulative distribution function (CCDF), PAPR performance, ESD performance, impulse response, frequency response, vector and matrix analysis, Siclab programming, Beamer programming, and Latex programming was achieved.

In this thesis, a descriptive signal and system model of three OFDM systems proposed in the literature, which are the DFT-based OFDM, the DWT-based OFDM, and the DT-CWT-based OFDM, were presented. A comparison of the BEP performance of the three systems in an AWGN channel was presented. The analytical results were verified by Monte Carlo simulation. In addition, a comparison of the impulse response, frequency response, PAPR, ESD for all systems was illustrated.

As a result, the DFT-based, DWT-based, and DT-CWT-based OFDM systems all had

the same BEP performance in an AWGN channel. For the system response, the DFT-based OFDM system yielded a flat frequency response. The DT-CWT-based OFDM system yielded single sided frequency response for a single non-zero coefficient per word, because the output of the system was constructed from $2\times$ redundancy, where one was Hilbert transform of the other. However, the frequency response for the DWT and for a single tree of the DT-CWT yielded a double side band.

The ESD was single sided for the DT-CWT-based system, because of the same reason for the single side frequency response stated earlier. However, the DWT with Haar filters, the DWT with D-6 filters, the upper tree of the DT-CWT, and the lower tree of the DT-CWT gave a double side ESD. When the previous four ESDs were compared, it was clear that as the filter lengths were closer the ESDs were more alike. The ESD for the DFT-based system was flat.

The CCDF for the PAPR showed the least P_{PAPR} for the DT-CWT-based OFDM system compared to the DFT-based and the DWT-based OFDM systems. The P_{PAPR} for the DWT-based with D-6 was close to the upper and the lower tree structures when considering them individually, because their filter lengths were relatively close, while Haar filters were not. It was clear from the results that the DFT case yielded the worst PAPR, while the DT-CWT yielded the best. It follows from the fact that the maximum PAPR for the DFT-based was proportional to the number of subcarriers as they add constructively. On the other hand, the PAPR for the DT-CWT-based was lower, due to the fact that the average power of the upper and the lower trees were added linearly.

Although the results in the thesis showed an improvement in the PAPR for the DT-CWT-based OFDM system compared to the other alternatives, using DT-CWT basis sacrifices the ability to allocate power as a function frequency to cope with fading, which is the main advantage of using DFT-based OFDM. As future work, other advantages for DT-CWT basis are to be explored for different channel conditions.

BIBLIOGRAPHY

Bibliography

- [1] Andrea Goldsmith, *Wireless Communications*, Cambridge University Press, Aug 8, 2005.
- [2] B.G. Negash and H. Nikookar, “Wavelet-based multicarrier transmission over wireless channels”, *Vehicular Technology Conference*, 2001.
- [3] B.G. Negash and H. Nikookar, “Wavelet based OFDM for wireless channels”, *Electronic Letters*, IEE, 2000.
- [4] Volkan Kumbasar and Oguz Kucur, “Better wavelet packet structure for PAPR reduction in WOFDM”, *Digital Signal Processing*, Elsevier Inc., 2008.
- [5] C. Van Bouwel, and J. Potemans, S. Schepers, B. Nauwelaers, and A. Van de Capelle, “Wavelet packet based multicarrier modulation”, *Symposium on Communications and Vehicular Technology*, 2000.
- [6] Antony Jamin and Petri Mahonen, “Wavelet packet modulation for wireless communications”, *Wireless Communications & Mobile Computing Journal*, Vol. 5, Issue 2, March 2005.
- [7] Mohamed Nerma, Nidal Kamel, and Varun Jeoti, “An OFDM system based on dual tree complex wavelet transform (DT-CWT)”, *Signal Processing: An International Journal*, 2009.
- [8] Z. Rafique, N. Gohar, and M.J. Mughal, “Performance comparison of OFDM and WOFDM based V-BLAST wireless systems”, *IEICE Trans Commun*, 2005.
- [9] Matthieu Gautier and Joel Lienard “Performance of complex wavelet packet based multicarrier transmission through double dispersive channel”, *Norwegian Signal Processing Society*, 2006.
- [10] Xiaodong Zhang and Guangguo Bi, “OFDM scheme based on complex orthogonal wavelet packet”, *12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 2001.
- [11] Clive Maxfield and John Bird, *Electrical Engineering*, Newnes, Aug 25, 2008.
- [12] John G. Proakis and Dimitris G. Manolakis, *Digital Signal Processing, Principles, Algorithms, and Applications*, Prentice Hall, 3rd edition, 1995.
- [13] John G. Proakis and Masoud Salehi, *Digital Communications*, McGraw-Hill, 2008.
- [14] Ivan W. Selesnick, Richard G. Baraniuk, and Nick G. Kingsbury, “A coherent framework for multiscale signal and image processing: the dual-tree complex wavelet transform”, *IEEE Signal Processing Magazine*, Nov, 2005.
- [15] Martin Vetterli and Jelena Kovacevic, *Wavelets and Subband Coding*, Prentice-Hall, 1995.
- [16] Gilbert Strang and Truong Nguyen, *Wavelets and Filter Banks*, Wellesley-Cambridge Press, 1997.

- [17] Patrick J. Van Fleet, *Discrete Wavelet Transformations: An Elementary Approach With Applications*, John Wiley & Sons, 2008.
- [18] Nick G. Kingsbury, "The dual-tree complex wavelet transform: a new efficient tool for image restoration and enhancement", *Proc. European Signal Processing Conf.*, Rhodes, Sept. 1998.
- [19] Ivan W. Selesnick, "Hilbert transform pairs of wavelet bases", *IEEE Signal Processing Letters*, Vol. 8, No. 6, June 2001.
- [20] Nick G. Kingsbury "A dual-tree complex wavelet transform with improved orthogonality and symmetry properties", *Image Processing, 2000. Proceedings. 2000 International Conference on*, pp.375-378, Vol.2, 10-13 Sept. 2000.
- [21] Ivan W. Selesnick, "A coherent framework for multiscale signal and image processing: the dual-tree complex wavelet transform", *IEEE Transaction on Signal Processing*, Vol. 50, No. 5, May 2002.
- [22] Xianbin Wang, Tjhung, T.T. and Ng, C.S., "Error probability performance of OFDM-ADSL systems", *Global Telecommunications Conference, 1998. The Bridge to Global Integration*. IEEE 1998.
- [23] Himlal A. Suraweera, Xiaolin Zhou and J. Armstrong, "On the calculation of OFDM error performance with phase noise in AWGN and fading channels", *Vehicular Technology Conference*, IEEE 2006.
- [24] Athanathan, K. and Tellambura, C., "Probability of error calculation of OFDM systems with frequency offset", *IEEE Transactions on Communications*, 2001.
- [25] Jacques Van Wyk and Louis Linde, "Bit error probability for a M-ary QAM OFDM-based system", *AFRICON 2007*.
- [26] I. Daubechies, *Ten Lectures On Wavelets*, Piladelphia, PA: SIAM, 1992.
- [27] <http://taco.poly.edu/WaveletSoftware/>
- [28] B. P. Lathi, *Modern Digital and Analog Communication Systems*, Oxford University Press, 1998.
- [29] Ariel Luzzatto and Gadi Shirazi, *Wireless Transceiver Design: Mastering the Design of Modern Wireless Equipment and Systems*, John Wiley and Sons, 2007.

APPENDICES

A. Simulation code

All codes are written using Scilab-5.2.2.

A.1 Bit Error Probability (BEP) Code for OFDM Alternatives

A.1.1 FFT-based OFDM.

```
1  clc
2  clear
3  //-----
4  // Q_FUNCTION
5  function [q_function] = Q_FUNCTION(some_value),
6    q_function = 0.5*erfc(sqrt(0.5)*some_value);
7  endfunction
8  //-----
9  // the transmitter side
10 //-----
11 data_length = 64;
12 number_channels = 64;
13 sqrt_no_channels = sqrt(number_channels);
14 data_before_norm = zeros(data_length,2);
15 data = zeros(data_length,1);
16 output = zeros(data_length,1);
17 output_copy = zeros(data_length,1);
18 constructed_data = zeros(data_length,1);
19 reconstructed_data1 = zeros(data_length,2);
20 reconstructed_data = zeros(data_length,2);
21 number_of_times_to_avg_on = 3*10^4;
22 //-----
23 // data has to be of even length
```

```

24 data_before_norm = grand(data_length,2,'uin',0,1);
25 random = rand(data_before_norm,'normal');
26 for L = 1:data_length,
27     if random(L,1) > 0 then
28         data_before_norm(L,1) = 2*data_before_norm(L,1)+1;
29     else data_before_norm(L,1) = -1*(2*data_before_norm(L,1)+1);
30     end
31     if random(L,2) > 0 then
32         data_before_norm(L,2) = 2*data_before_norm(L,2)+1;
33     else data_before_norm(L,2) = -1*(2*data_before_norm(L,2)+1);
34     end
35 end
36 normalization = sqrt(10);
37 data = (data_before_norm(1:data_length,1)+%i*data_before_norm(1:
    data_length,2))/normalization; //normalized
38 //—————
39 ifft_data = sqrt(no_channels)*ifft(data);
40 output = ifft_data;
41 //—————
42 // the channel
43 //—————
44 counter1 = zeros(1,9);
45 for signal_to_noise_ratio = 0:8,
46     for no_of_times_to_avg_on = 1:number_of_times_to_avg_on,
47         output_copy = output;
48         some_constants = 1/normalization*sqrt(10/8*10(-0.1*
            signal_to_noise_ratio));
49         // both real and imag noise have same power as regular noise "one
            dimensional noise" because nbar^2=nbar_c^2=nbar_s^2
50         real_noise = some_constants*rand(data_length,1,'normal');
51         imag_noise = %i*some_constants*rand(data_length,1,'normal');
52         output_copy = output_copy + real_noise + imag_noise;
53 //—————
54 // the receiver side
55 //—————
56 fft_data = normalization/sqrt(no_channels)*fft(output_copy);
57 reconstructed_data1(1:data_length,1) = real(fft_data);
58 reconstructed_data1(1:data_length,2) = imag(fft_data);
59 //—————
60 // error countor:
61 //—————
62 sign_reconstructed_data1 = zeros(data_length,2);
63 abs_reconstructed_data1 = zeros(data_length,2);

```

```

64 sign_reconstructed_data1 = sign(reconstructed_data1);
65 abs_reconstructed_data1 = abs(reconstructed_data1);
66 //—————
67 for L = 1:data.length ,
68     if (abs_reconstructed_data1(L,1) < 2 & abs_reconstructed_data1(L,2) <
        2) then
69     reconstructed_data(L,1) = 1*sign_reconstructed_data1(L,1);
70     reconstructed_data(L,2) = 1*sign_reconstructed_data1(L,2);
71
72     elseif (abs_reconstructed_data1(L,1) >= 2 & abs_reconstructed_data1(L
        ,2) < 2) then
73     reconstructed_data(L,1) = 3*sign_reconstructed_data1(L,1);
74     reconstructed_data(L,2) = 1*sign_reconstructed_data1(L,2);
75
76     elseif (abs_reconstructed_data1(L,1) < 2 & abs_reconstructed_data1(L
        ,2) >= 2) then
77     reconstructed_data(L,1) = 1*sign_reconstructed_data1(L,1);
78     reconstructed_data(L,2) = 3*sign_reconstructed_data1(L,2);
79
80     elseif (abs_reconstructed_data1(L,1) >= 2 & abs_reconstructed_data1(L
        ,2) >= 2) then
81     reconstructed_data(L,1) = 3*sign_reconstructed_data1(L,1);
82     reconstructed_data(L,2) = 3*sign_reconstructed_data1(L,2);
83 end
84 //—————
85 if ( (reconstructed_data(L,1) ~= data_before_norm(L,1)) | (
        reconstructed_data(L,2) ~= data_before_norm(L,2)) ) then
86     counter1(1,signal_to_noise_ratio+1) = counter1(1,
        signal_to_noise_ratio+1) + 1;
87 end //if data1
88 end // for L
89 //—————
90 end // for no_of_times_to_avg_on
91 end // for signal_to_noise_ratio
92 bit_error_rate1 = counter1/(4*data.length*number_of_times_to_avg_on);
93 signal_to_noise_ratio = 0:8
94 // the theoretical results
95 Q = QFUNCTION(sqrt(8/10*(10.^(signal_to_noise_ratio/10))));
96 //sigma^2 = 5/(4*SNR_b)
97 QAM16 = (3*Q - 9/4*Q^2)/4;
98 // Plot using Tikz:
99 fd = mopen('Users\Tassniem\Documents\TikZ\figure_1\figure_1.tex', 'wt');
100 mfprintf(fd, '%s\n', '\documentclass{article}');

```

```

101 mfprintf(fd, '%s\n', '\usepackage{tikz}');
102 mfprintf(fd, '%s\n', '\usepackage{pgfplots}');
103 mfprintf(fd, '%s\n', '\begin{document}');
104 mfprintf(fd, '%s\n', '\begin{center}');
105 //-----
106 mfprintf(fd, '%s\n', '\begin{tikzpicture}');
107 mfprintf(fd, '%s\n', '\begin{semilogyaxis}[');
108 mfprintf(fd, '%s\n', 'xlabel=SNR(dB),');
109 mfprintf(fd, '%s\n', 'ylabel={BEP}]');
110 mfprintf(fd, '%s\n', '\addplot[ color=black, mark=triangle]');
111 mfprintf(fd, '%s\n', 'coordinates{');
112 for signal_to_noise_ratio = 0:8,
113     mfprintf(fd, '%s%d%s%.6f%s\n', '(', signal_to_noise_ratio, ', ', QAM16
114         (1, signal_to_noise_ratio+1), ')');
115 end
116 mfprintf(fd, '%s\n', ');');
117 mfprintf(fd, '%s\n', '\addplot[ color=black, mark=o]');
118 mfprintf(fd, '%s\n', 'coordinates{');
119 for signal_to_noise_ratio = 0:8,
120     mfprintf(fd, '%s%d%s%.6f%s\n', '(', signal_to_noise_ratio, ', ',
121         bit_error_rate1(1, signal_to_noise_ratio+1), ')');
122 end
123 mfprintf(fd, '%s\n', ');');
124 mfprintf(fd, '%s\n', '\legend{Numerical\\Simulated}\\}');
125 mfprintf(fd, '%s\n', '\end{semilogyaxis}');
126 mfprintf(fd, '%s\n', '\end{tikzpicture}');
127 //-----
128 mfprintf(fd, '%s\n', '\end{center}');
129 mfprintf(fd, '%s\n', '\end{document}');
130 mclose(fd);

```

A.1.2 DWT-based OFDM, with Haar filters.

```
1  clc
2  clear
3  //-----
4  //CIRCULAR_SHIFT_BY_TWO
5  function [shifted_sequence] = CIRCULAR_SHIFT_BY_TWO(
        to_be_shifted_sequence),
6
7  to_be_shifted_sequence_length = length(to_be_shifted_sequence);
8
9  shifted_sequence = [ to_be_shifted_sequence(1,
        to_be_shifted_sequence_length -1:to_be_shifted_sequence_length),
        to_be_shifted_sequence(1,1:to_be_shifted_sequence_length -2)];
10
11 endfunction
12 //-----
13 // MATRIX_EVALUATION
14 function [matrix_evaluation] = MATRIX_EVALUATION(transformation_length ,
        filters),
15 [r, c] = size(filters);
16 matrix_evaluation = zeros(transformation_length , transformation_length)
        ;
17
18 matrix_evaluation(1,1:c) = filters(1,1:c);
19 matrix_evaluation(1+transformation_length/2,1:c) = filters(2,1:c);
20
21 for row = 2:transformation_length/2,
22     matrix_evaluation(row,1:c) = CIRCULAR_SHIFT_BY_TWO(
        matrix_evaluation(row-1,1:c));
23     matrix_evaluation(row+transformation_length/2,1:c) =
        CIRCULAR_SHIFT_BY_TWO( matrix_evaluation(row-1+
        transformation_length/2,1:c));
24     end
25 endfunction
26 //-----
27 // DWT
28 function [dwt_matrix] = DWT(data_length , filter_length , no_stages , filters) ,
29
30 dwt_matrix = eye(data_length , data_length);
31
32 if no_stages > 1 then
33
```



```

34     dwt_matrix(1:data_length/2^no_stages,1:data_length/2^no_stages) =
        MATRIX_EVALUATION(data_length/2^no_stages,[filters(3:4,:), zeros
            (2,data_length/2^no_stages-filter_length)]);
35
36     if no_stages > 2 then
37         for stage = no_stages-1:-1:2,
38             matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters(3:4,:),
                zeros(2,data_length/2^stage-filter_length)]);
39             dwt_matrix(1:data_length/2^(stage+1),1:data_length/2^stage) =
                dwt_matrix(1:data_length/2^(stage+1),1:data_length/2^(stage+1)
                    )*matrix_i(1:data_length/2^(stage+1),1:data_length/2^stage);
40             dwt_matrix(data_length/2^stage/2+1:data_length/2^stage,1:
                data_length/2^stage) = matrix_i(data_length/2^stage/2+1:
                    data_length/2^stage,1:data_length/2^stage);
41         end
42     end
43 end
44     matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,
        data_length-filter_length)]);
45     dwt_matrix(1:data_length/2,1:data_length) = dwt_matrix(1:data_length
        /2,1:data_length/2)*matrix_i(1:data_length/2,1:data_length);
46     dwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(
        data_length/2+1:data_length,1:data_length);
47 endfunction
48 //-----
49 // IDWT
50 function [idwt_matrix] = IDWT(data_length,filter_length,no_stages,filters
    ),
51
52     idwt_matrix = eye(data_length,data_length);
53
54     if no_stages > 1 then
55
56         idwt_matrix(1:data_length/2^no_stages,1:data_length/2^no_stages) =
            MATRIX_EVALUATION(data_length/2^no_stages,[filters(3:4,:), zeros
                (2,data_length/2^no_stages-filter_length)]);
57
58         if no_stages > 2 then
59             for stage = no_stages-1:-1:2,
60                 matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters
                    (3:4,:),zeros(2,data_length/2^stage-filter_length)]);

```

```

61         idwt_matrix(1:data_length/2^(stage+1),1:data_length/2^stage) =
            idwt_matrix(1:data_length/2^(stage+1),1:data_length/2^(stage
+1))*matrix_i(1:data_length/2^(stage+1),1:data_length/2^
stage);
62         idwt_matrix(data_length/2^stage/2+1:data_length/2^stage,1:
            data_length/2^stage) = matrix_i(data_length/2^stage/2+1:
            data_length/2^stage,1:data_length/2^stage);
63     end
64     end
65 end
66
67     matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,
            data_length-filter_length)]);
68     idwt_matrix(1:data_length/2,1:data_length) = idwt_matrix(1:
            data_length/2,1:data_length/2)*matrix_i(1:data_length/2,1:
            data_length);
69     idwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(
            data_length/2+1:data_length,1:data_length);
70
71     idwt_matrix = idwt_matrix.';
72 endfunction
73 //-----
74 // Q_FUNCTION
75 function [q_function] = Q_FUNCTION(some_value),
76
77     q_function = 0.5*erfc(sqrt(0.5)*some_value);
78
79 endfunction
80 //-----
81 // AVERAGE
82 function [average] = AVERAGE(some_vector),
83
84     average = sum(some_vector)/length(some_vector);
85
86 endfunction
87 //-----
88 number_of_times_to_avg_on = 3*10^4;
89 data_length = 64;
90 number_channels = 64;
91 sqrt_no_channels = sqrt(number_channels);
92 data_before_norm = zeros(data_length,2);
93 SNR = 0:8;
94 SNR_range = length(SNR);

```

```

95 //-----
96 // Transmitter
97 //-----
98 // Data
99 data_before_norm = grand(data_length,2,'uin',0,1);
100 random = rand(data_before_norm,'normal');
101 for L = 1:data_length,
102     if random(L,1) > 0 then
103         data_before_norm(L,1) = 2*data_before_norm(L,1)+1;
104     else data_before_norm(L,1) = -1*(2*data_before_norm(L,1)+1);
105     end
106     if random(L,2) > 0 then
107         data_before_norm(L,2) = 2*data_before_norm(L,2)+1;
108     else data_before_norm(L,2) = -1*(2*data_before_norm(L,2)+1);
109     end
110 end
111 normalization = sqrt(10);
112 //-----
113 // DWT-OFDM, Haar filters
114 //-----
115 wavelet = dbwavf('db1'); // it is Haar
116 [low_analysis_filter, high_analysis_filter, low_synthesis_filter,
    high_synthesis_filter] = orthfilt(wavelet);
117 analysis_filter = [low_analysis_filter; high_analysis_filter];
118 First_analysis_filter = analysis_filter;
119 synthesis_filter = [low_synthesis_filter; high_synthesis_filter];
120 Final_synthesis_filter = synthesis_filter;
121 //-----
122 no_stages = 5;
123 filter_length = length(analysis_filter(1,:));
124 //-----
125 data = (data_before_norm(1:data_length,1) + %i*data_before_norm(1:
    data_length,2))/normalization; //normalized
126 //-----
127 // Filters
128 low_filter = 1;
129 high_filter = 2;
130
131 ANALYSIS_FILTERS = [First_analysis_filter(low_filter:high_filter,:);
    analysis_filter(low_filter:high_filter,:)];
132
133 SYNTHESIS_FILTERS = [Final_synthesis_filter(low_filter:high_filter,:);
    synthesis_filter(low_filter:high_filter,:)];

```

```

134 the_idwt_matrix_Haar = IDWT(data_length , filter_length , no_stages , flipdim
    (SYNTHESIS_FILTERS,2));
135 //-----
136 // Transformation
137 //-----
138 output_IDWT_Haar = the_idwt_matrix_Haar*data;
139 //-----
140 // Channel
141 //-----
142 counter1 = zeros(1,9);
143 for signal_to_noise_ratio = 0:8,
144     for no_of_times_to_avg_on = 1:number_of_times_to_avg_on ,
145
146         copy_output_IDWT_Haar = output_IDWT_Haar;
147         some_constants = 1/normalization*sqrt(5/4*10^(-0.1*
            signal_to_noise_ratio));
148         // both real and imag noise have same power as regular noise "one
            dimentinal noise" because nbar^2=nbar_c^2=nbar_s^2
149
150         real_noise = some_constants*rand(data_length ,1 , 'normal');
151         imag_noise = %i*some_constants*rand(data_length ,1 , 'normal');
152
153         copy_output_IDWT_Haar = copy_output_IDWT_Haar + real_noise +
            imag_noise;
154
155         the_dwt_matrix_Haar = DWT(data_length , filter_length , no_stages ,
            ANALYSIS_FILTERS);
156
157         output_DWT_Haar = the_dwt_matrix_Haar*copy_output_IDWT_Haar*
            normalization;
158         //-----
159         output_DWT_Haar1(1:data_length ,1) = real(output_DWT_Haar);
160         output_DWT_Haar1(1:data_length ,2) = imag(output_DWT_Haar);
161         //-----
162         sign_output_DWT_Haar = sign(output_DWT_Haar1);
163         abs_output_DWT_Haar = abs(output_DWT_Haar1);
164         //-----
165         for L = 1:data_length ,
166
167             if (abs_output_DWT_Haar(L,1) < 2 & abs_output_DWT_Haar(L,2) < 2)
                then
168                 output_DWT_Haar(L,1) = 1*sign_output_DWT_Haar(L,1);
169                 output_DWT_Haar(L,2) = 1*sign_output_DWT_Haar(L,2);

```

```

170
171     elseif (abs_output_DWT_Haar(L,1) >= 2 & abs_output_DWT_Haar(L
        ,2) < 2) then
172     output_DWT_Haar(L,1) = 3*sign_output_DWT_Haar(L,1);
173     output_DWT_Haar(L,2) = 1*sign_output_DWT_Haar(L,2);
174
175     elseif (abs_output_DWT_Haar(L,1) < 2 & abs_output_DWT_Haar(L,2)
        >= 2) then
176     output_DWT_Haar(L,1) = 1*sign_output_DWT_Haar(L,1);
177     output_DWT_Haar(L,2) = 3*sign_output_DWT_Haar(L,2);
178
179     elseif (abs_output_DWT_Haar(L,1) >= 2 & abs_output_DWT_Haar(L
        ,2) >= 2) then
180     output_DWT_Haar(L,1) = 3*sign_output_DWT_Haar(L,1);
181     output_DWT_Haar(L,2) = 3*sign_output_DWT_Haar(L,2);
182     end
183     //—————
184     if ( (output_DWT_Haar(L,1) ~= data_before_norm(L,1)) | (
        output_DWT_Haar(L,2) ~= data_before_norm(L,2)) ) then
185     counter1(1,signal_to_noise_ratio+1) = counter1(1,
        signal_to_noise_ratio+1) + 1;
186     end //if data1
187     end // for L
188     //—————
189 end // for no_of_times_to_avg_on
190 end // for signal_to_noise_ratio
191
192 bit_error_rate1 = counter1/(4*data_length*number_of_times_to_avg_on);
193 signal_to_noise_ratio = 0:8
194 // the theoretical results
195 Q = QFUNCTION(sqrt(8/10*(10.^(signal_to_noise_ratio/10))));
196 //sigma^2 = 5/(4*SNR_b)
197 QAM16 = (3*Q - 9/4*Q^2)/4;
198 // Plot using Tikz:
199 fd = mopen('\Users\Tassniem\Documents\TikZ\figure_2\figure2.tex', 'wt');
200 mfprintf(fd, '%s_\n', '\documentclass{article}');
201 mfprintf(fd, '%s_\n', '\usepackage{tikz}');
202 mfprintf(fd, '%s_\n', '\usepackage{pgfplots}');
203 mfprintf(fd, '%s_\n', '\begin{document}');
204 //—————
205 mfprintf(fd, '%s_\n', '\begin{center}');
206 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
207 mfprintf(fd, '%s_\n', '\begin{semilogyaxis}[');

```

```

208 mfprintf(fd, '%s\n', 'xlabel=SNR(dB), ');
209 mfprintf(fd, '%s\n', 'ylabel={BEP} ');
210 mfprintf(fd, '%s\n', '\addplot [ color=black ,mark=triangle ] ');
211 mfprintf(fd, '%s\n', 'coordinates{ ');
212 for signal_to_noise_ratio = 0:8,
213   mfprintf(fd, '%s%d%s%.6f%s\n', '(' , signal_to_noise_ratio , ', ', QAM16
      (1, signal_to_noise_ratio+1), ') ');
214 end
215 mfprintf(fd, '%s\n', '); ');
216 mfprintf(fd, '%s\n', '\addplot [ color=black ,mark=o ] ');
217 mfprintf(fd, '%s\n', 'coordinates{ ');
218 for signal_to_noise_ratio = 0:8,
219   mfprintf(fd, '%s%d%s%.6f%s\n', '(' , signal_to_noise_ratio , ', ',
      bit_error_rate1(1, signal_to_noise_ratio+1), ') ');
220 end
221 mfprintf(fd, '%s\n', '); ');
222 mfprintf(fd, '%s\n', '\legend{Numerical\\Simulated}\\ ');
223 mfprintf(fd, '%s\n', '\end{semilogyaxis} ');
224 mfprintf(fd, '%s\n', '\end{tikzpicture} ');
225 mfprintf(fd, '%s\n', '\end{center} ');
226 mfprintf(fd, '%s\n', '\end{document} ');
227 mclose(fd);

```

A.1.3 DWT-based OFDM, with D-6 filters.

```
1  clc
2  clear
3  //-----
4  //CIRCULAR_SHIFT_BY_TWO
5  function [shifted_sequence] = CIRCULAR_SHIFT_BY_TWO(
        to_be_shifted_sequence),
6
7  to_be_shifted_sequence_length = length(to_be_shifted_sequence);
8
9  shifted_sequence = [ to_be_shifted_sequence(1,
        to_be_shifted_sequence_length -1:to_be_shifted_sequence_length),
        to_be_shifted_sequence(1,1:to_be_shifted_sequence_length -2)];
10
11 endfunction
12 //-----
13 // MATRIX_EVALUATION
14 function [matrix_evaluation] = MATRIX_EVALUATION(transformation_length ,
        filters),
15 [r, c] = size(filters);
16 matrix_evaluation = zeros(transformation_length , transformation_length)
        ;
17
18 matrix_evaluation(1,1:c) = filters(1,1:c);
19 matrix_evaluation(1+transformation_length/2,1:c) = filters(2,1:c);
20
21 for row = 2:transformation_length/2,
22     matrix_evaluation(row,1:c) = CIRCULAR_SHIFT_BY_TWO(
        matrix_evaluation(row-1,1:c));
23     matrix_evaluation(row+transformation_length/2,1:c) =
        CIRCULAR_SHIFT_BY_TWO( matrix_evaluation(row-1+
        transformation_length/2,1:c));
24     end
25 endfunction
26 //-----
27 // DWT
28 function [dwt_matrix] = DWT(data_length , filter_length , no_stages , filters) ,
29
30 dwt_matrix = eye(data_length , data_length);
31
32 if no_stages > 1 then
33
```

```

34     dwt_matrix(1:data_length/2^no_stages,1:data_length/2^no_stages) =
        MATRIX_EVALUATION(data_length/2^no_stages,[filters(3:4,:), zeros
            (2,data_length/2^no_stages-filter_length)]);
35
36     if no_stages > 2 then
37         for stage = no_stages-1:-1:2,
38             matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters(3:4,:),
                zeros(2,data_length/2^stage-filter_length)]);
39             dwt_matrix(1:data_length/2^(stage+1),1:data_length/2^stage) =
                dwt_matrix(1:data_length/2^(stage+1),1:data_length/2^(stage+1)
                    )*matrix_i(1:data_length/2^(stage+1),1:data_length/2^stage);
40             dwt_matrix(data_length/2^stage/2+1:data_length/2^stage,1:
                data_length/2^stage) = matrix_i(data_length/2^stage/2+1:
                    data_length/2^stage,1:data_length/2^stage);
41         end
42     end
43 end
44     matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,
        data_length-filter_length)]);
45     dwt_matrix(1:data_length/2,1:data_length) = dwt_matrix(1:data_length
        /2,1:data_length/2)*matrix_i(1:data_length/2,1:data_length);
46     dwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(
        data_length/2+1:data_length,1:data_length);
47 endfunction
48 //-----
49 // IDWT
50 function [idwt_matrix] = IDWT(data_length,filter_length,no_stages,filters
    ),
51
52     idwt_matrix = eye(data_length,data_length);
53
54     if no_stages > 1 then
55         idwt_matrix(1:data_length/2^no_stages,1:data_length/2^no_stages) =
            MATRIX_EVALUATION(data_length/2^no_stages,[filters(3:4,:), zeros
                (2,data_length/2^no_stages-filter_length)]);
56
57         if no_stages > 2 then
58             for stage = no_stages-1:-1:2,
59
60                 matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters
                    (3:4,:),zeros(2,data_length/2^stage-filter_length)]);

```



```

61         idwt_matrix(1:data_length/2^(stage+1),1:data_length/2^stage) =
            idwt_matrix(1:data_length/2^(stage+1),1:data_length/2^(stage
+1))*matrix_i(1:data_length/2^(stage+1),1:data_length/2^
stage);
62         idwt_matrix(data_length/2^stage/2+1:data_length/2^stage,1:
            data_length/2^stage) = matrix_i(data_length/2^stage/2+1:
            data_length/2^stage,1:data_length/2^stage);
63     end
64 end
65 end
66 matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,
            data_length-filter_length)]);
67 idwt_matrix(1:data_length/2,1:data_length) = idwt_matrix(1:
            data_length/2,1:data_length/2)*matrix_i(1:data_length/2,1:
            data_length);
68 idwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(
            data_length/2+1:data_length,1:data_length);
69
70 idwt_matrix = idwt_matrix.';
71 endfunction
72 //-----
73 // Q_FUNCTION
74 function [q_function] = Q_FUNCTION(some_value),
75
76     q_function = 0.5*erfc(sqrt(0.5)*some_value);
77
78 endfunction
79 //-----
80 // AVERAGE
81 function [average] = AVERAGE(some_vector),
82
83     average = sum(some_vector)/length(some_vector);
84
85 endfunction
86 //-----
87 number_of_times_to_avg_on = 3*10^4;
88 data_length = 64;
89 number_channels = 64;
90 sqrt_no_channels = sqrt(number_channels);
91 data_before_norm = zeros(data_length,2);
92 SNR = 0:8;
93 SNR_range = length(SNR);
94 //-----

```

```

95 // Transmitter
96 //-----
97 // Data
98 data_before_norm = grand(data_length,2,'uin',0,1);
99 random = rand(data_before_norm,'normal');
100 for L = 1:data_length,
101     if random(L,1) > 0 then
102         data_before_norm(L,1) = 2*data_before_norm(L,1)+1;
103     else data_before_norm(L,1) = -1*(2*data_before_norm(L,1)+1);
104     end
105     if random(L,2) > 0 then
106         data_before_norm(L,2) = 2*data_before_norm(L,2)+1;
107     else data_before_norm(L,2) = -1*(2*data_before_norm(L,2)+1);
108     end
109 end
110 normalization = sqrt(10);
111 //-----
112 // DWT-OFDM, D-6 filters
113 //-----
114 wavelet = dbwavf('db3'); // it is D-6
115 [low_analysis_filter, high_analysis_filter, low_synthesis_filter,
    high_synthesis_filter] = orthfilt(wavelet);
116 analysis_filter = [low_analysis_filter; high_analysis_filter];
117 First_analysis_filter = analysis_filter;
118 synthesis_filter = [low_synthesis_filter; high_synthesis_filter];
119 Final_synthesis_filter = synthesis_filter;
120 //-----
121 no_stages = 3;
122 filter_length = length(analysis_filter(1,:));
123 //-----
124 data = (data_before_norm(1:data_length,1) + %i*data_before_norm(1:
    data_length,2))/normalization; //normalized
125 //-----
126 // Filters
127 low_filter = 1;
128 high_filter = 2;
129
130 ANALYSIS_FILTERS = [First_analysis_filter(low_filter:high_filter,:);
    analysis_filter(low_filter:high_filter,:)];
131
132 SYNTHESIS_FILTERS = [Final_synthesis_filter(low_filter:high_filter,:);
    synthesis_filter(low_filter:high_filter,:)];

```

```

133 the_idwt_matrix_D6 = IDWT(data_length , filter_length , no_stages , flipdim (
      SYNTHESIS_FILTERS,2) );
134 //-----
135 // Transformation
136 //-----
137 output_IDWT_D6 = the_idwt_matrix_D6*data;
138 //-----
139 // Channel
140 //-----
141 counter1 = zeros(1,9);
142
143 for signal_to_noise_ratio = 0:8,
144     for no_of_times_to_avg_on = 1:number_of_times_to_avg_on ,
145
146         copy_output_IDWT_D6 = output_IDWT_D6;
147         some_constants = 1/normalization*sqrt(5/4*10^(-0.1*
            signal_to_noise_ratio));
148         // both real and imag noise have same power as regular noise "one
            dimentional noise" because nbar^2=nbar_c^2=nbar_s^2
149         real_noise = some_constants*rand(data_length,1,'normal');
150         imag_noise = %i*some_constants*rand(data_length,1,'normal');
151
152         copy_output_IDWT_D6 = copy_output_IDWT_D6 + real_noise + imag_noise
            ;
153
154         the_dwt_matrix_D6 = DWT(data_length , filter_length , no_stages ,
            ANALYSIS_FILTERS);
155
156         output_DWT_D6 = the_dwt_matrix_D6*copy_output_IDWT_D6*normalization
            ;
157         //-----
158         output_DWT_D61(1:data_length,1) = real(output_DWT_D6);
159         output_DWT_D61(1:data_length,2) = imag(output_DWT_D6);
160         //-----
161         sign_output_DWT_D6 = sign(output_DWT_D61);
162         abs_output_DWT_D6 = abs(output_DWT_D61);
163         //-----
164         for L = 1:data_length ,
165
166             if (abs_output_DWT_D6(L,1) < 2 & abs_output_DWT_D6(L,2) < 2) then
167                 output_DWT_D6(L,1) = 1*sign_output_DWT_D6(L,1);
168                 output_DWT_D6(L,2) = 1*sign_output_DWT_D6(L,2);
169

```

```

170         elseif (abs_output_DWT_D6(L,1) >= 2 & abs_output_DWT_D6(L,2) <
171             2) then
172             output_DWT_D6(L,1) = 3*sign_output_DWT_D6(L,1);
173             output_DWT_D6(L,2) = 1*sign_output_DWT_D6(L,2);
174
175         elseif (abs_output_DWT_D6(L,1) < 2 & abs_output_DWT_D6(L,2) >=
176             2) then
177             output_DWT_D6(L,1) = 1*sign_output_DWT_D6(L,1);
178             output_DWT_D6(L,2) = 3*sign_output_DWT_D6(L,2);
179
180         elseif (abs_output_DWT_D6(L,1) >= 2 & abs_output_DWT_D6(L,2) >=
181             2) then
182             output_DWT_D6(L,1) = 3*sign_output_DWT_D6(L,1);
183             output_DWT_D6(L,2) = 3*sign_output_DWT_D6(L,2);
184         end
185         //—————
186         if ( (output_DWT_D6(L,1) ~= data_before_norm(L,1)) | (
187             output_DWT_D6(L,2) ~= data_before_norm(L,2)) ) then
188             counter1(1, signal_to_noise_ratio+1) = counter1(1,
189                 signal_to_noise_ratio+1) + 1;
190         end //if data1
191     end // for L
192 //—————
193 end // for no_of_times_to_avg_on
194 end // for signal_to_noise_ratio
195
196 bit_error_rate1 = counter1/(4*data_length*number_of_times_to_avg_on);
197 signal_to_noise_ratio = 0:8
198 // the theoretical results
199 Q = QFUNCTION(sqrt(8/10*(10.^(signal_to_noise_ratio/10))));
200 //sigma^2 = 5/(4*SNR_b)
201 QAM16 = (3*Q - 9/4*Q^2)/4;
202 // Plot using Tikz:
203 fd = mopen('\Users\Tassniem\Documents\TikZ\figure_3\figure3.tex', 'wt');
204 mfprintf(fd, '%s_\n', '\documentclass{article}');
205 mfprintf(fd, '%s_\n', '\usepackage{tikz}');
206 mfprintf(fd, '%s_\n', '\usepackage{pgfplots}');
207 mfprintf(fd, '%s_\n', '\begin{document}');
208 //—————
209 mfprintf(fd, '%s_\n', '\begin{center}');
210 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
211 mfprintf(fd, '%s_\n', '\begin{semilogyaxis}[');
212 mfprintf(fd, '%s_\n', 'xlabel=SNR(dB), ');

```

```

208 mfprintf(fd, '%s\n', 'ylabel={BEP} ');
209 mfprintf(fd, '%s\n', '\addplot [ color=black ,mark=triangle ] ');
210 mfprintf(fd, '%s\n', 'coordinates{ ');
211 for signal_to_noise_ratio = 0:8,
212     mfprintf(fd, '%s%d%s%.6f%s\n', '(' , signal_to_noise_ratio , ',' , QAM16
                (1, signal_to_noise_ratio+1), ') ');
213 end
214 mfprintf(fd, '%s\n', '}; ');
215 mfprintf(fd, '%s\n', '\addplot [ color=black ,mark=o ] ');
216 mfprintf(fd, '%s\n', 'coordinates{ ');
217 for signal_to_noise_ratio = 0:8,
218     mfprintf(fd, '%s%d%s%.6f%s\n', '(' , signal_to_noise_ratio , ',' ,
                bit_error_rate1(1, signal_to_noise_ratio+1), ') ');
219 end
220 mfprintf(fd, '%s\n', '}; ');
221 mfprintf(fd, '%s\n', '\legend{Numerical\\Simulated}\\} ');
222 mfprintf(fd, '%s\n', '\end{semilogyaxis} ');
223 mfprintf(fd, '%s\n', '\end{tikzpicture} ');
224 mfprintf(fd, '%s\n', '\end{center} ');
225 mfprintf(fd, '%s\n', '\end{document} ');
226 mclose(fd);

```

A.1.4 DT-CWT-based OFDM, with q-shift filters.

```
1  clc
2  clear
3  //-----
4  //CIRCULAR_SHIFT_BY_TWO
5  function [shifted_sequence] = CIRCULAR_SHIFT_BY_TWO(
        to_be_shifted_sequence),
6
7  to_be_shifted_sequence_length = length(to_be_shifted_sequence);
8
9  shifted_sequence = [ to_be_shifted_sequence(1,
        to_be_shifted_sequence_length -1:to_be_shifted_sequence_length),
        to_be_shifted_sequence(1,1:to_be_shifted_sequence_length -2)];
10
11 endfunction
12 //-----
13 // MATRIX_EVALUATION
14 function [matrix_evaluation] = MATRIX_EVALUATION(transformation_length ,
        filters),
15 [r, c] = size(filters);
16 matrix_evaluation = zeros(transformation_length , transformation_length)
        ;
17
18 matrix_evaluation(1,1:c) = filters(1,1:c);
19 matrix_evaluation(1+transformation_length/2,1:c) = filters(2,1:c);
20
21 for row = 2:transformation_length/2,
22     matrix_evaluation(row,1:c) = CIRCULAR_SHIFT_BY_TWO(
        matrix_evaluation(row-1,1:c));
23     matrix_evaluation(row+transformation_length/2,1:c) =
        CIRCULAR_SHIFT_BY_TWO( matrix_evaluation(row-1+
        transformation_length/2,1:c));
24     end
25 endfunction
26 //-----
27 // DWT
28 function [dwt_matrix] = DWT(data_length , filter_length , no_stages , filters) ,
29
30 dwt_matrix = eye(data_length , data_length);
31
32 if no_stages > 1 then
```

```

33     dwt_matrix(1:data_length/2^no_stages,1:data_length/2^no_stages) =
        MATRIX_EVALUATION(data_length/2^no_stages,[filters(3:4,:), zeros
            (2,data_length/2^no_stages-filter_length)]);
34
35     if no_stages > 2 then
36         for stage = no_stages-1:-1:2,
37             matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters(3:4,:),
                zeros(2,data_length/2^stage-filter_length)]);
38             dwt_matrix(1:data_length/2^(stage+1),1:data_length/2^stage) =
                dwt_matrix(1:data_length/2^(stage+1),1:data_length/2^(stage+1)
                    )*matrix_i(1:data_length/2^(stage+1),1:data_length/2^stage);
39             dwt_matrix(data_length/2^stage/2+1:data_length/2^stage,1:
                data_length/2^stage) = matrix_i(data_length/2^stage/2+1:
                    data_length/2^stage,1:data_length/2^stage);
40         end
41     end
42 end
43 matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:), zeros(2,
    data_length-filter_length)]);
44 dwt_matrix(1:data_length/2,1:data_length) = dwt_matrix(1:data_length
    /2,1:data_length/2)*matrix_i(1:data_length/2,1:data_length);
45 dwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(
    data_length/2+1:data_length,1:data_length);
46
47 endfunction
48 //-----
49 // IDWT
50 function [idwt_matrix] = IDWT(data_length,filter_length,no_stages,filters
    ),
51
52 idwt_matrix = eye(data_length,data_length);
53
54     if no_stages > 1 then
55
56         idwt_matrix(1:data_length/2^no_stages,1:data_length/2^no_stages) =
            MATRIX_EVALUATION(data_length/2^no_stages,[filters(3:4,:), zeros
                (2,data_length/2^no_stages-filter_length)]);
57
58         if no_stages > 2 then
59             for stage = no_stages-1:-1:2,
60                 matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters
                    (3:4,:), zeros(2,data_length/2^stage-filter_length)]);

```

```

61         idwt_matrix(1:data_length/2^(stage+1),1:data_length/2^stage) =
            idwt_matrix(1:data_length/2^(stage+1),1:data_length/2^(stage
+1))*matrix_i(1:data_length/2^(stage+1),1:data_length/2^
stage);
62         idwt_matrix(data_length/2^stage/2+1:data_length/2^stage,1:
            data_length/2^stage) = matrix_i(data_length/2^stage/2+1:
            data_length/2^stage,1:data_length/2^stage);
63     end
64 end
65 end
66 matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,
            data_length-filter_length)]);
67 idwt_matrix(1:data_length/2,1:data_length) = idwt_matrix(1:
            data_length/2,1:data_length/2)*matrix_i(1:data_length/2,1:
            data_length);
68 idwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(
            data_length/2+1:data_length,1:data_length);
69
70 idwt_matrix = idwt_matrix.';
71 endfunction
72 //-----
73 // Q_FUNCTION
74 function [q_function] = Q_FUNCTION(some_value),
75
76     q_function = 0.5*erfc(sqrt(0.5)*some_value);
77
78 endfunction
79 //-----
80 // AVERAGE
81 function [average] = AVERAGE(some_vector),
82
83     average = sum(some_vector)/length(some_vector);
84
85 endfunction
86 //-----
87 number_of_times_to_avg_on = 3*10^4;
88 data_length = 64;
89 number_channels = 64;
90 sqrt_no_channels = sqrt(number_channels);
91 data_before_norm = zeros(data_length,2);
92 SNR = 0:8;
93 SNR_range = length(SNR);
94 //-----

```



```

95 // Transmitter
96 //-----
97 // Data
98 data_before_norm = grand(data_length,2,'uin',0,1);
99 random = rand(data_before_norm,'normal');
100 for L = 1:data_length,
101     if random(L,1) > 0 then
102         data_before_norm(L,1) = 2*data_before_norm(L,1)+1;
103     else data_before_norm(L,1) = -1*(2*data_before_norm(L,1)+1);
104     end
105     if random(L,2) > 0 then
106         data_before_norm(L,2) = 2*data_before_norm(L,2)+1;
107     else data_before_norm(L,2) = -1*(2*data_before_norm(L,2)+1);
108     end
109 end
110 normalization = sqrt(10);
111 //-----
112 // DT-CWF-OFDM, q-shift filters
113 //-----
114 [First_analysis_filter, Final_synthesis_filter] = FSfarras('f');
115 [analysis_filter, synthesis_filter] = dualfilt1('f');
116 //-----
117 no_stages = 2;
118 filter_length = length(analysis_filter(1,:));
119 //-----
120 data = (data_before_norm(1:data_length,1) + %i*data_before_norm(1:
    data_length,2))/normalization; //normalized
121 //-----
122 // Filters
123 //-----
124 for tree_no = 1:2,
125     if tree_no == 1 then
126         low_filter = 1;
127         high_filter = 2;
128     elseif tree_no == 2 then
129         low_filter = 3;
130         high_filter = 4;
131     end // for elseif
132     //-----
133
134 SYNTHESIS_FILTERS = [Final_synthesis_filter(low_filter:high_filter,:);
    synthesis_filter(low_filter:high_filter,:)];
135

```

```

136     the_idwt_matrix = IDWT(data_length , filter_length , no_stages , flipdim (
        SYNTHESIS_FILTERS,2));
137
138     //-----
139     // Transformation
140     //-----
141     output_IDWT = the_idwt_matrix*data;
142     //-----
143     // to store output of both trees
144     if tree_no == 1 then
145         output_IDWT_upper = output_IDWT;
146     elseif tree_no == 2 then
147         output_IDWT_lower = output_IDWT;
148     end // if tree_no
149 end // for tree_no
150     //-----
151     // Channel
152     //-----
153     counter = zeros(1,9);
154
155     for signal_to_noise_ratio = 0:8,
156         for no_of_times_to_avg_on = 1:number_of_times_to_avg_on ,
157
158             for tree_no = 1:2,
159                 if tree_no == 1 then
160                     low_filter = 1;
161                     high_filter = 2;
162                     copy_output_IDWT = output_IDWT_upper;
163                 elseif tree_no == 2 then
164                     low_filter = 3;
165                     high_filter = 4;
166                     copy_output_IDWT = output_IDWT_lower;
167                 end // for elseif
168
169                 ANALYSIS_FILTERS = [ First_analysis_filter(low_filter : high_filter
                    ,:); analysis_filter(low_filter : high_filter ,:) ];
170
171
172                 some_constants = sqrt(1/4*10^(-0.1*signal_to_noise_ratio));
173                 // both real and imag noise have same power as regular noise "one
                    dimentinal noise" because  $\bar{n}^2 = \bar{n}_c^2 = \bar{n}_s^2$ 
174                 real_noise = some_constants*rand(data_length ,1 , 'normal');
175                 imag_noise = %i*some_constants*rand(data_length ,1 , 'normal');

```

```

176
177     copy_output_IDWT = copy_output_IDWT + real_noise + imag_noise;
178
179     the_dwt_matrix = DWT(data_length , filter_length , no_stages ,
        ANALYSIS_FILTERS);
180
181     output_DWT = the_dwt_matrix*copy_output_IDWT;
182
183     if tree_no == 1 then
184         output_DWT_upper = output_DWT;
185     elseif tree_no == 2 then
186         output_DWT_lower = output_DWT;
187     end // if tree_no
188 end // for tree_no
189     reconstructed_data_both = zeros(data_length ,1);
190     reconstructed_data_both = .5*(output_DWT_upper + output_DWT_lower)*
        normalization;
191     //—————
192     output_DWT_both = zeros(data_length ,2);
193     output_DWT_both(1:data_length ,1) = real(reconstructed_data_both);
194     output_DWT_both(1:data_length ,2) = imag(reconstructed_data_both);
195     //—————
196     sign_output_DWT = zeros(data_length ,2);
197     abs_output_DWT = zeros(data_length ,2);
198
199     sign_output_DWT = sign(output_DWT_both);
200     abs_output_DWT = abs(output_DWT_both);
201     //—————
202     for L = 1:data_length ,
203
204         if (abs_output_DWT(L,1) < 2 & abs_output_DWT(L,2) < 2) then
205             output_DWT_both(L,1) = 1*sign_output_DWT(L,1);
206             output_DWT_both(L,2) = 1*sign_output_DWT(L,2);
207
208             elseif (abs_output_DWT(L,1) >= 2 & abs_output_DWT(L,2) < 2)
                then
209                 output_DWT_both(L,1) = 3*sign_output_DWT(L,1);
210                 output_DWT_both(L,2) = 1*sign_output_DWT(L,2);
211
212             elseif (abs_output_DWT(L,1) < 2 & abs_output_DWT(L,2) >= 2)
                then
213                 output_DWT_both(L,1) = 1*sign_output_DWT(L,1);
214                 output_DWT_both(L,2) = 3*sign_output_DWT(L,2);

```

```

215
216         elseif (abs_output_DWT(L,1) >= 2 & abs_output_DWT(L,2) >= 2)
                then
217             output_DWT_both(L,1) = 3*sign_output_DWT(L,1);
218             output_DWT_both(L,2) = 3*sign_output_DWT(L,2);
219         end
220         //—————
221         if ( (output_DWT_both(L,1) ~= data_before_norm(L,1)) | (
                output_DWT_both(L,2) ~= data_before_norm(L,2)) ) then
222             counter(1,signal_to_noise_ratio+1) = counter(1,
                signal_to_noise_ratio+1) + 1;
223         end //if
224     end // for L
225     //—————
226 end // for no_of_times_to_avg_on
227 end // for signal_to_noise_ratio
228
229 bit_error_rate = counter/(4*data_length*number_of_times_to_avg_on);
230
231 signal_to_noise_ratio = 0:8
232
233 // the theoretical results
234 Q = QFUNCTION(sqrt(8/10*(10.^(signal_to_noise_ratio/10))));
235 //sigma^2 = 5/(4*SNR_b)
236 QAM16 = (3*Q - 9/4*Q^2)/4;
237 // Plot using Tikz:
238 fd = mopen('\Users\Tassniem\Documents\TikZ\figure_4\figure4.tex', 'wt');
239 mfprintf(fd, '%s_\n', '\documentclass{article}');
240 mfprintf(fd, '%s_\n', '\usepackage{tikz}');
241 mfprintf(fd, '%s_\n', '\usepackage{pgfplots}');
242 mfprintf(fd, '%s_\n', '\begin{document}');
243 //—————
244 mfprintf(fd, '%s_\n', '\begin{center}');
245 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
246 mfprintf(fd, '%s_\n', '\begin{semilogyaxis}[');
247 mfprintf(fd, '%s_\n', 'xlabel=SNR(dB), ');
248 mfprintf(fd, '%s_\n', 'ylabel={BEP}]');
249 mfprintf(fd, '%s_\n', '\addplot [color=black, mark=triangle]');
250 mfprintf(fd, '%s_\n', 'coordinates{');
251 for signal_to_noise_ratio = 0:8,
252     mfprintf(fd, '%s_\%d_\%s_\%.6f_\%s_\n', '(', signal_to_noise_ratio, ', ', QAM16
                (1, signal_to_noise_ratio+1), ')');
253 end

```

```

254 mfprintf(fd, '%s\n', '};');
255 mfprintf(fd, '%s\n', '\addplot [ color=black , mark=o] ');
256 mfprintf(fd, '%s\n', 'coordinates{');
257 for signal_to_noise_ratio = 0:8,
258     mfprintf(fd, '%s%d%s%.6f%s\n', '(' , signal_to_noise_ratio , ', ',
                bit_error_rate(1, signal_to_noise_ratio+1), ')');
259 end
260 mfprintf(fd, '%s\n', '};');
261 mfprintf(fd, '%s\n', '\legend{Numerical\\Simulated}\\}');
262 mfprintf(fd, '%s\n', '\end{semilogyaxis}');
263 mfprintf(fd, '%s\n', '\end{tikzpicture}');
264 mfprintf(fd, '%s\n', '\end{center}');
265 mfprintf(fd, '%s\n', '\end{document}');
266 mclose(fd);

```

A.2 Impulse Response, Frequency Response, and Energy Spectral Density (ESD) Code for OFDM Alternatives

A.2.1 FFT-based OFDM.

```
1  clc
2  clear
3
4  //-----
5  // Q_FUNCTION
6  function [q_function] = Q_FUNCTION(some_value),
7
8     q_function = 0.5*erfc(sqrt(0.5)*some_value);
9
10 endfunction
11
12 //-----
13
14 //-----
15 //-----
16 // the transmitter side
17 //-----
18 //-----
19 // data_length := data is in symbols
20 data_length = 64;
21 number_channels = 64;
22 det_fft = number_channels;
23
24 data_in_freq = ones(data_length,1);
25
26 //-----
27 // Data
28 //-----
29
30 //-----
31 // DFT-OFDM
32 //-----
33
34 output_DFT = ifft(data_in_freq); //in time
35
```

```

36
37 //-----
38 // FFTs
39 //-----
40 data_in_time = zeros(64,1);
41 data_in_time(1,1) = 1;
42 fft_output = fft(data_in_time);
43 //fft_output = fft(output_DFT)/det_fft; //in freq
44
45
46 //-----
47 // Frequency Response
48 //-----
49
50 //-----
51 // Plot using Tikz:
52 fd = mopen('\Users\Tassniem\Documents\TikZ\IR\FFT_IR\FFT_IR.tex', 'wt');
53 fprintf(fd, '%s\n', '\documentclass{article}');
54 fprintf(fd, '%s\n', '\usepackage{tikz}');
55 fprintf(fd, '%s\n', '\usepackage{pgfplots}');
56 fprintf(fd, '%s\n', '\begin{document}');
57
58
59
60 fprintf(fd, '%s\n', '\begin{center}');
61
62 normalization_fft_output = max(abs(fft_output))
63
64 normalization_time_impulse = max(abs(output_DFT))
65
66 //-----
67 // Frequency Response
68 //-----
69
70 fprintf(fd, '%s\n', '\begin{tikzpicture}');
71 fprintf(fd, '%s\n', '\begin{axis}[ymax=1.1, ymin=-0.1,');
72 fprintf(fd, '%s\n', 'xlabel=$\omega/2\pi$,');
73 fprintf(fd, '%s\n', 'ylabel=_$\mathrm{Real}\{\Psi(\omega)\}$');
74
75 fprintf(fd, '%s\n', '\addplot[ color=black]');
76 fprintf(fd, '%s\n', 'coordinates{');
77
78 for n = 1:data.length,

```

```

79     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n/data_length , ', ', real(
        fft_output(n,1))/normalization_fft_output , ')');
80 end
81
82 mfprintf(fd, '%s\n', '};');
83
84 mfprintf(fd, '%s\n', '\end{axis}');
85 mfprintf(fd, '%s\n', '\end{tikzpicture}');
86 //-----
87 mfprintf(fd, '%s\n', '\begin{tikzpicture}');
88 mfprintf(fd, '%s\n', '\begin{axis}[ymax=1.1, ymin=-0.1,');
89 mfprintf(fd, '%s\n', 'xlabel=$\omega/2\pi$,');
90 mfprintf(fd, '%s\n', 'ylabel=_$\mathrm{Imag}\{\Psi(\omega)\}$');
91
92 mfprintf(fd, '%s\n', '\addplot[ color=black]');
93 mfprintf(fd, '%s\n', 'coordinates{');
94
95 for n = 1:data_length ,
96     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n/data_length , ', ', imag(
        fft_output(n,1))/normalization_fft_output , ')');
97 end
98
99 mfprintf(fd, '%s\n', '};');
100
101 mfprintf(fd, '%s\n', '\end{axis}');
102 mfprintf(fd, '%s\n', '\end{tikzpicture}');
103 //-----
104 //-----
105 mfprintf(fd, '%s\n', '\begin{tikzpicture}');
106 mfprintf(fd, '%s\n', '\begin{axis}[ymax=1.1, ymin=-0.1,');
107 mfprintf(fd, '%s\n', 'xlabel=$\omega/2\pi$,');
108 mfprintf(fd, '%s\n', 'ylabel=_$|\Psi(\omega)|^2$');
109
110 mfprintf(fd, '%s\n', '\addplot[ color=black]');
111 mfprintf(fd, '%s\n', 'coordinates{');
112
113 for n = 1:data_length ,
114     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n/data_length , ', ', fft_output(
        n,1)*conj(fft_output(n,1))/normalization_fft_output^2,')');
115 end
116
117 mfprintf(fd, '%s\n', '};');
118

```



```

119 mfprintf(fd , '%s\n' , '\end{ axis }');
120 mfprintf(fd , '%s\n' , '\end{ tikzpicture }');
121 //-----
122 // Impulse Response
123 //-----
124 mfprintf(fd , '%s\n' , '\newpage ');
125 mfprintf(fd , '%s\n' , '\begin{ tikzpicture }');
126 mfprintf(fd , '%s\n' , '\begin{ axis }[ymax=1.1 , ymin=-0.1, ');
127 mfprintf(fd , '%s\n' , 'xlabel=$t/64$, ');
128 mfprintf(fd , '%s\n' , 'ylabel=_$\mathrm{Real}\{\psi(t)\}$ ');
129
130 mfprintf(fd , '%s\n' , '\addplot [ color=black ] ');
131 mfprintf(fd , '%s\n' , 'coordinates { ');
132
133 for n = 1:data_length ,
134     mfprintf(fd , '%s%.6f%s%.6f%s\n' , '(' , n/data_length , ', ' , real(
        output_DFT(n,1))/normalization_time_impulse , ')');
135 end
136
137 mfprintf(fd , '%s\n' , '); ');
138
139 mfprintf(fd , '%s\n' , '\end{ axis }');
140 mfprintf(fd , '%s\n' , '\end{ tikzpicture }');
141 //-----
142 mfprintf(fd , '%s\n' , '\begin{ tikzpicture }');
143 mfprintf(fd , '%s\n' , '\begin{ axis }[ymax=1.1 , ymin=-0.1, ');
144 mfprintf(fd , '%s\n' , 'xlabel=$t/64$, ');
145 mfprintf(fd , '%s\n' , 'ylabel=_$\mathrm{Imag}\{\psi(t)\}$ ');
146
147 mfprintf(fd , '%s\n' , '\addplot [ color=black ] ');
148 mfprintf(fd , '%s\n' , 'coordinates { ');
149
150 for n = 1:data_length ,
151     mfprintf(fd , '%s%.6f%s%.6f%s\n' , '(' , n/data_length , ', ' , imag(
        output_DFT(n,1))/normalization_time_impulse , ')');
152 end
153
154 mfprintf(fd , '%s\n' , '); ');
155
156 mfprintf(fd , '%s\n' , '\end{ axis }');
157 mfprintf(fd , '%s\n' , '\end{ tikzpicture }');
158 //-----
159 //-----

```

```

160 mfprintf(fd , '%s\n' , '\begin{tikzpicture}');
161 mfprintf(fd , '%s\n' , '\begin{axis}[ymax=1.1, ymin=-0.1, ');
162 mfprintf(fd , '%s\n' , 'xlabel=$t/64$, ');
163 mfprintf(fd , '%s\n' , 'ylabel=|psi(t)|$]');
164
165 mfprintf(fd , '%s\n' , '\addplot[ color=black]');
166 mfprintf(fd , '%s\n' , 'coordinates{');
167
168 for n = 1:data_length ,
169     mfprintf(fd , '%s%.6f%s%.6f%s\n' , '(' , n/data_length , ', ' , abs(
        output_DFT(n,1))/normalization_time_impulse , ')');
170 end
171
172 mfprintf(fd , '%s\n' , '};');
173
174 mfprintf(fd , '%s\n' , '\end{axis}');
175 mfprintf(fd , '%s\n' , '\end{tikzpicture}');
176 //-----
177
178
179
180
181 mfprintf(fd , '%s\n' , '\end{center}');
182 mfprintf(fd , '%s\n' , '\end{document}');
183 mclose(fd);

```

A.2.2 DWT-based OFDM, with Haar filters.

```
1  clc
2  clear
3
4  //-----
5  //CIRCULAR_SHIFT_BY_TWO
6  function [shifted_sequence] = CIRCULAR_SHIFT_BY_TWO(
       to_be_shifted_sequence),
7
8     to_be_shifted_sequence_length = length(to_be_shifted_sequence);
9
10    shifted_sequence = [ to_be_shifted_sequence(1,
        to_be_shifted_sequence_length - 1:to_be_shifted_sequence_length),
        to_be_shifted_sequence(1,1:to_be_shifted_sequence_length - 2)];
11
12  endfunction
13
14  //-----
15  // DISCRETE_WAVELET_TRANSFORM_MATRIX
16  function [discrete_wavelet_transform_matrix] =
        DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter , High_filter , zero_padding
        ),
17
18     Length = length(Low_filter) + zero_padding;
19
20     discrete_wavelet_transform_matrix = zeros(Length, Length);
21
22     discrete_wavelet_transform_matrix(1,1:Length) = [flipdim(Low_filter ,2),
        zeros(1, zero_padding)];
23     discrete_wavelet_transform_matrix(1+Length/2,1:Length) = [flipdim(
        High_filter ,2), zeros(1, zero_padding)];
24
25     for row = 2:Length/2,
26         discrete_wavelet_transform_matrix(row,1:Length) =
            CIRCULAR_SHIFT_BY_TWO( discrete_wavelet_transform_matrix(row
            -1,1:Length));
27         discrete_wavelet_transform_matrix(row+Length/2,1:Length) =
            CIRCULAR_SHIFT_BY_TWO( discrete_wavelet_transform_matrix(row-1+
            Length/2,1:Length));
28     end,
29
30  endfunction
```

```

31
32 //-----
33 // INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX
34 function [inverse_discrete_wavelet_transform_matrix] =
    INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter , High_filter ,
    zero_padding) ,
35
36     inverse_discrete_wavelet_transform_matrix =
        DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter , High_filter ,
        zero_padding) ' ;
37
38 endfunction
39
40 //-----
41 // Q_FUNCTION
42 function [q_function] = Q_FUNCTION(some_value) ,
43
44     q_function = 0.5*erfc(sqrt(0.5)*some_value) ;
45
46 endfunction
47
48 //-----
49 wavelet = dbwavf('db1') ; // it is db6
50 [low_analysis_filter , high_analysis_filter , low_synthesis_filter ,
    high_synthesis_filter] = orthfilt(wavelet) ;
51 analysis_filter = [low_analysis_filter ; high_analysis_filter] ;
52 First_analysis_filter = analysis_filter ;
53 synthesis_filter = [low_synthesis_filter ; high_synthesis_filter] ;
54 Final_synthesis_filter = synthesis_filter ;
55 //-----
56 // [First_analysis_filter , Final_synthesis_filter] = FSfarras('f') ;
57 // [analysis_filter , synthesis_filter] = dualfilt1('f') ;
58 //-----
59 // square_root_half = sqrt(.5) ;
60 // analysis_filter = [square_root_half , square_root_half ; -
    square_root_half , square_root_half] ;
61 // First_analysis_filter = analysis_filter ;
62 // synthesis_filter = [square_root_half , square_root_half ;
    square_root_half , -square_root_half] ;
63 // Final_synthesis_filter = synthesis_filter ;
64 //-----
65 no_stages = 4 ;
66 filter_length = length(analysis_filter(1,:)) ;

```

```

67 //-----
68
69 data_length = 64//filter_length*2^no_stages;
70
71 //-----
72 // Data
73 //-----
74 data = zeros(data_length,1);
75 data(20, 1) = 1;
76 normalization = sqrt(10);
77 data_before_norm = data;
78
79 //-----
80 //-----
81 zero_padding = abs(data_length - filter_length);
82
83
84 //-----
85 // Number of data per branch
86 //-----
87 branch_rate = zeros(1,no_stages+1);// to store # of ele. per branch
88 branch_rate(1,1) = data_length/2^no_stages;
89 for k = no_stages:-1:1,
90     branch_rate(1,no_stages+2-k) = data_length/2^k;
91 end
92 cumsum_branch_rate = cumsum(branch_rate);// matrix to store the
    cumulative sum up to kth elem.
93 //-----
94     data_copy = zeros(data_length,1);
95     constructed_data = zeros(data_length,1);
96
97
98 //-----
99 //-----
100 // the transmitter side
101 //-----
102 //-----
103
104
105 //-----
106 for tree_no = 1:1,
107
108     if tree_no == 1 then

```

```

109     low_filter = 1;
110     high_filter = 2;
111     elseif tree_no == 2 then
112         low_filter = 3;
113         high_filter = 4;
114     end // for elseif
115     //-----
116
117
118
119     data_copy = data; // data has to be of even length
120
121     //-----
122     // Stages from 1:no_stages-1
123     //-----
124     constructed_data(1:cumsum_branch_rate(1,2),1) = data_copy(1:
        cumsum_branch_rate(1,2),1);
125
126
127     for stage = 1:no_stages-1,
128
129         // data_length = cumsum_branch_rate(1,stage+1)
130         zero_padding = abs(cumsum_branch_rate(1,stage+1) - filter_length);
131
132
133         constructed_data(1:cumsum_branch_rate(1,stage+1),1) =
            INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX( flipdim(
                synthesis_filter(low_filter,:),2), flipdim(synthesis_filter(
                    high_filter,:),2), zero_padding)*data_copy(1:cumsum_branch_rate(1,
                    stage+1),1);
134     // filters are flipped, due to using same function of the DWT, and they
        // have to be arranged in an inverse order.
135
136     data_copy(1:cumsum_branch_rate(1,stage+2),1) = [constructed_data(1:
        cumsum_branch_rate(1,stage+1),1); data_copy(cumsum_branch_rate(1,
        stage+1)+1:cumsum_branch_rate(1,stage+2),1)];
137
138     end
139
140     //-----
141     // Stage number no_stages, "last stage"
142     //-----
143

```

```

144 zero_padding = abs(cumsum_branch_rate(1,no_stages+1) - filter_length);
145
146
147 constructed_data(1:cumsum_branch_rate(1,no_stages+1),1) =
    INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX( flipdim(
        Final_synthesis_filter(low_filter,:),2), flipdim(
        Final_synthesis_filter(high_filter,:),2), zero_padding)*data_copy(1:
        cumsum_branch_rate(1,no_stages+1),1);
148 // filters are flipped, due to using same function of the DWT, and they
    have to be arranged in an inverse order.
149
150
151
152
153
154
155
156 //-----
157 // to store output of both trees
158 if tree_no == 1 then
159     output_tree1 = constructed_data
160
161     elseif tree_no == 2 then
162         output_tree2 = constructed_data
163
164     end // if tree_no
165
166
167 end // for tree_no
168
169
170 //-----
171 // FFTs
172 //-----
173
174 fft_output_tree1 = fft(output_tree1);
175
176 //-----
177
178
179 // Plot using Tikz:
180 fd = mopen('Users\Tassniem\Documents\TikZ\IR\DWT_Haar_IR\DWT_Haar_IR.tex
    ', 'wt');

```

```

181 mfprintf(fd, '%s\n', '\documentclass{article}');
182 mfprintf(fd, '%s\n', '\usepackage{tikz}');
183 mfprintf(fd, '%s\n', '\usepackage{pgfplots}');
184 mfprintf(fd, '%s\n', '\begin{document}');
185
186
187 mfprintf(fd, '%s\n', '\begin{center}');
188
189
190 normalization_time1 = max(abs(output_tree1))
191
192 normalization_freq1 = max(abs(fft_output_tree1))
193
194 //—————
195 // impulse_response in Frequency
196 //—————
197
198 mfprintf(fd, '%s\n', '\begin{tikzpicture}');
199 mfprintf(fd, '%s\n', '\begin{axis}[');
200 mfprintf(fd, '%s\n', 'xlabel=$\omega/2\pi$, ');
201 mfprintf(fd, '%s\n', 'ylabel=_$\mathrm{Real}\{\Psi_h(\omega)\}$');
202
203 mfprintf(fd, '%s\n', '\addplot[ color=black]');
204 mfprintf(fd, '%s\n', 'coordinates{');
205
206 for n = 1:data_length,
207     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(', n/data_length, ', ', real(
208         fft_output_tree1(n,1))/normalization_freq1, ')');
209
210 mfprintf(fd, '%s\n', '};');
211
212 mfprintf(fd, '%s\n', '\end{axis}');
213 mfprintf(fd, '%s\n', '\end{tikzpicture}');
214 //—————
215 mfprintf(fd, '%s\n', '\begin{tikzpicture}');
216 mfprintf(fd, '%s\n', '\begin{axis}[');
217 mfprintf(fd, '%s\n', 'xlabel=$\omega/2\pi$, ');
218 mfprintf(fd, '%s\n', 'ylabel=_$\mathrm{Imag}\{\Psi_h(\omega)\}$');
219
220 mfprintf(fd, '%s\n', '\addplot[ color=black]');
221 mfprintf(fd, '%s\n', 'coordinates{');
222

```



```

223 for n = 1:data_length ,
224     mfprintf(fd , '%s_%.6f_%s_%.6f_%s_\n' , '(' , n/data_length , ',' , imag(
        fft_output_tree1(n,1))/normalization_freq1 , ')');
225 end
226
227 mfprintf(fd , '%s_\n' , '};');
228
229 mfprintf(fd , '%s_\n' , '\end{axis}');
230 mfprintf(fd , '%s_\n' , '\end{tikzpicture}');
231
232 //-----
233 mfprintf(fd , '%s_\n' , '\begin{tikzpicture}');
234 mfprintf(fd , '%s_\n' , '\begin{axis}[');
235 mfprintf(fd , '%s_\n' , 'xlabel=\omega/2\pi$ , ');
236 mfprintf(fd , '%s_\n' , 'ylabel=_$|\Psi_h(\omega)|^2$]');
237
238 mfprintf(fd , '%s_\n' , '\addplot[ color=black]');
239 mfprintf(fd , '%s_\n' , 'coordinates{');
240
241 for n = 1:data_length ,
242     mfprintf(fd , '%s_%.6f_%s_%.6f_%s_\n' , '(' , n/data_length , ',' , (abs(
        fft_output_tree1(n,1)))^2/normalization_freq1^2 , ')');
243 end
244
245 mfprintf(fd , '%s_\n' , '};');
246
247 mfprintf(fd , '%s_\n' , '\end{axis}');
248 mfprintf(fd , '%s_\n' , '\end{tikzpicture}');
249 //-----
250
251 // impulse_response in Time
252 //-----
253 // Upper
254 //-----
255 mfprintf(fd , '%s_\n' , '\newpage');
256 mfprintf(fd , '%s_\n' , '\begin{tikzpicture}');
257 mfprintf(fd , '%s_\n' , '\begin{axis}[');
258 mfprintf(fd , '%s_\n' , 'xlabel=_$t/64$ , ');
259 mfprintf(fd , '%s_\n' , 'ylabel=_$\mathrm{Real}\{\Psi_h(t)\}$]');
260
261 mfprintf(fd , '%s_\n' , '\addplot[ color=black]');
262 mfprintf(fd , '%s_\n' , 'coordinates{');
263

```

```

264 for n = 1:data_length ,
265     mfprintf(fd , '%s%.6f%s%.6f%s\n' , '(' , n/data_length , ',' , real(
        output_tree1(n,1))/normalization_time1 , ')');
266 end
267
268 mfprintf(fd , '%s\n' , '};');
269
270 mfprintf(fd , '%s\n' , '\end{axis}');
271 mfprintf(fd , '%s\n' , '\end{tikzpicture}');
272 //-----
273 mfprintf(fd , '%s\n' , '\begin{tikzpicture}');
274 mfprintf(fd , '%s\n' , '\begin{axis}[ymax=1.1, ymin=-0.1,');
275 mfprintf(fd , '%s\n' , 'xlabel=$t/64$,');
276 mfprintf(fd , '%s\n' , 'ylabel=_$\mathrm{Imag}\{\psi_h(t)\}$');
277
278 mfprintf(fd , '%s\n' , '\addplot[ color=black]');
279 mfprintf(fd , '%s\n' , 'coordinates{');
280
281 for n = 1:data_length ,
282     mfprintf(fd , '%s%.6f%s%.6f%s\n' , '(' , n/data_length , ',' , imag(
        output_tree1(n,1))/normalization_time1 , ')');
283 end
284
285 mfprintf(fd , '%s\n' , '};');
286
287 mfprintf(fd , '%s\n' , '\end{axis}');
288 mfprintf(fd , '%s\n' , '\end{tikzpicture}');
289 //-----
290 mfprintf(fd , '%s\n' , '\begin{tikzpicture}');
291 mfprintf(fd , '%s\n' , '\begin{axis}[');
292 mfprintf(fd , '%s\n' , 'xlabel=$t/64$,');
293 mfprintf(fd , '%s\n' , 'ylabel=_$|\psi_h(t)|$');
294
295 mfprintf(fd , '%s\n' , '\addplot[ color=black]');
296 mfprintf(fd , '%s\n' , 'coordinates{');
297
298 for n = 1:data_length ,
299     mfprintf(fd , '%s%.6f%s%.6f%s\n' , '(' , n/data_length , ',' , abs(
        output_tree1(n,1))/normalization_time1 , ')');
300 end
301
302 mfprintf(fd , '%s\n' , '};');
303 mfprintf(fd , '%s\n' , '\end{axis}');

```

```
304 mfprintf(fd, '%s\\n', '\\end{tikzpicture}');
305 //—————
306
307
308 mfprintf(fd, '%s\\n', '\\end{center}');
309 mfprintf(fd, '%s\\n', '\\end{document}');
310 fclose(fd);
```

A.2.3 DWT-based OFDM, with D-6 filters.

```
1  clc
2  clear
3
4  //-----
5  //CIRCULAR_SHIFT_BY_TWO
6  function [shifted_sequence] = CIRCULAR_SHIFT_BY_TWO(
       to_be_shifted_sequence),
7
8     to_be_shifted_sequence_length = length(to_be_shifted_sequence);
9
10    shifted_sequence = [ to_be_shifted_sequence(1,
        to_be_shifted_sequence_length-1:to_be_shifted_sequence_length),
        to_be_shifted_sequence(1,1:to_be_shifted_sequence_length-2)];
11
12  endfunction
13
14  //-----
15  // DISCRETE_WAVELET_TRANSFORMMATRIX
16  function [discrete_wavelet_transform_matrix] =
        DISCRETE_WAVELET_TRANSFORMMATRIX( Low_filter , High_filter , zero_padding
        ),
17
18     Length = length(Low_filter) + zero_padding;
19
20     discrete_wavelet_transform_matrix = zeros(Length, Length);
21
22     discrete_wavelet_transform_matrix(1,1:Length) = [flipdim(Low_filter,2),
        zeros(1,zero_padding)];
23     discrete_wavelet_transform_matrix(1+Length/2,1:Length) = [flipdim(
        High_filter,2),zeros(1,zero_padding)];
24
25     for row = 2:Length/2,
26         discrete_wavelet_transform_matrix(row,1:Length) =
            CIRCULAR_SHIFT_BY_TWO( discrete_wavelet_transform_matrix(row
            -1,1:Length));
27         discrete_wavelet_transform_matrix(row+Length/2,1:Length) =
            CIRCULAR_SHIFT_BY_TWO( discrete_wavelet_transform_matrix(row-1+
            Length/2,1:Length));
28     end,
29
30  endfunction
```

```

31
32 //-----
33 // INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX
34 function [inverse_discrete_wavelet_transform_matrix] =
    INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter , High_filter ,
    zero_padding) ,
35
36     inverse_discrete_wavelet_transform_matrix =
        DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter , High_filter ,
        zero_padding) ' ;
37
38 endfunction
39
40 //-----
41 // Q_FUNCTION
42 function [q_function] = Q_FUNCTION(some_value) ,
43
44     q_function = 0.5*erfc(sqrt(0.5)*some_value) ;
45
46 endfunction
47
48 //-----
49 wavelet = dbwavf('db3') ; // it is db6
50 [low_analysis_filter , high_analysis_filter , low_synthesis_filter ,
    high_synthesis_filter] = orthfilt(wavelet) ;
51 analysis_filter = [low_analysis_filter ; high_analysis_filter] ;
52 First_analysis_filter = analysis_filter ;
53 synthesis_filter = [low_synthesis_filter ; high_synthesis_filter] ;
54 Final_synthesis_filter = synthesis_filter ;
55 //-----
56 // [First_analysis_filter , Final_synthesis_filter] = FSfarras('f') ;
57 // [analysis_filter , synthesis_filter] = dualfilt1('f') ;
58 //-----
59 // square_root_half = sqrt(.5) ;
60 // analysis_filter = [square_root_half , square_root_half ; -
    square_root_half , square_root_half] ;
61 // First_analysis_filter = analysis_filter ;
62 // synthesis_filter = [square_root_half , square_root_half ;
    square_root_half , -square_root_half] ;
63 // Final_synthesis_filter = synthesis_filter ;
64 //-----
65 no_stages = 4 ;
66 filter_length = length(analysis_filter(1,:)) ;

```

```

67 //-----
68
69 data_length = 64//filter_length*2^no_stages;
70
71 //-----
72 // Data
73 //-----
74 data = zeros(data_length,1);
75 data(20, 1) = 1;
76 normalization = sqrt(10);
77 data_before_norm = data;
78
79 //-----
80 //-----
81 zero_padding = abs(data_length - filter_length);
82
83
84 //-----
85 // Number of data per branch
86 //-----
87 branch_rate = zeros(1,no_stages+1);// to store # of ele. per branch
88 branch_rate(1,1) = data_length/2^no_stages;
89 for k = no_stages:-1:1,
90     branch_rate(1,no_stages+2-k) = data_length/2^k;
91 end
92 cumsum_branch_rate = cumsum(branch_rate);// matrix to store the
    cumulative sum up to kth elem.
93 //-----
94     data_copy = zeros(data_length,1);
95     constructed_data = zeros(data_length,1);
96
97
98 //-----
99 //-----
100 // the transmitter side
101 //-----
102 //-----
103
104
105 //-----
106 for tree_no = 1:1,
107
108     if tree_no == 1 then

```

```

109     low_filter = 1;
110     high_filter = 2;
111     elseif tree_no == 2 then
112         low_filter = 3;
113         high_filter = 4;
114     end // for elseif
115     //-----
116
117
118
119     data_copy = data; // data has to be of even length
120
121     //-----
122     // Stages from 1:no_stages-1
123     //-----
124     constructed_data(1:cumsum_branch_rate(1,2),1) = data_copy(1:
        cumsum_branch_rate(1,2),1);
125
126
127     for stage = 1:no_stages-1,
128
129         // data_length = cumsum_branch_rate(1,stage+1)
130         zero_padding = abs(cumsum_branch_rate(1,stage+1) - filter_length);
131
132
133         constructed_data(1:cumsum_branch_rate(1,stage+1),1) =
            INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX( flipdim(
                synthesis_filter(low_filter,:),2), flipdim(synthesis_filter(
                    high_filter,:),2), zero_padding)*data_copy(1:cumsum_branch_rate(1,
                    stage+1),1);
134     // filters are flipped, due to using same function of the DWT, and they
        // have to be arranged in an inverse order.
135
136     data_copy(1:cumsum_branch_rate(1,stage+2),1) = [constructed_data(1:
        cumsum_branch_rate(1,stage+1),1); data_copy(cumsum_branch_rate(1,
        stage+1)+1:cumsum_branch_rate(1,stage+2),1)];
137
138     end
139
140     //-----
141     // Stage number no_stages, "last stage"
142     //-----
143

```

```

144 zero_padding = abs(cumsum_branch_rate(1,no_stages+1) - filter_length);
145
146
147 constructed_data(1:cumsum_branch_rate(1,no_stages+1),1) =
    INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX( flipdim(
        Final_synthesis_filter(low_filter,:),2), flipdim(
        Final_synthesis_filter(high_filter,:),2), zero_padding)*data_copy(1:
        cumsum_branch_rate(1,no_stages+1),1);
148 // filters are flipped, due to using same function of the DWT, and they
    have to be arranged in an inverse order.

149
150
151
152
153
154
155
156 //—————
157 // to store output of both trees
158 if tree_no == 1 then
159     output_tree1 = constructed_data
160
161     elseif tree_no == 2 then
162         output_tree2 = constructed_data
163
164     end // if tree_no
165
166
167
168
169
170
171 end // for tree_no
172
173
174
175 //—————
176 // FFTs
177 //—————
178
179 fft_output_tree1 = fft(constructed_data);
180
181 //—————

```



```

182 // ESD
183 //-----
184
185 energy_spectral_density_tree1 = fft_output_tree1.*conj(fft_output_tree1)/
    data_length;
186
187 //-----
188 // Impulse Response in frequency
189 //-----
190
191 freq_impulse_response_tree1 = sqrt(fft_output_tree1.*conj(
    fft_output_tree1));
192
193 //-----
194
195 //-----
196 // FFTs
197 //-----
198
199 fft_output_tree1 = fft(output_tree1);
200
201 //-----
202
203
204 // Plot using Tikz:
205 fd = mopen('\Users\Tassniem\Documents\TikZ\IR\DWT_D6_IR\DWT_D6_IR.tex', '
    wt');
206 mfprintf(fd, '%s\n', '\documentclass{article}');
207 mfprintf(fd, '%s\n', '\usepackage{tikz}');
208 mfprintf(fd, '%s\n', '\usepackage{pgfplots}');
209 mfprintf(fd, '%s\n', '\begin{document}');
210
211
212
213 mfprintf(fd, '%s\n', '\begin{center}');
214
215
216 normalization_time1 = max(abs(output_tree1))
217
218 normalization_freq1 = max(abs(fft_output_tree1))
219
220 //-----
221 // impulse_response in Frequency

```

```

222 //-----
223
224 mfprintf(fd, '%s\n', '\begin{tikzpicture}');
225 mfprintf(fd, '%s\n', '\begin{axis}[');
226 mfprintf(fd, '%s\n', 'xlabel=$\omega/2\pi$, ');
227 mfprintf(fd, '%s\n', 'ylabel=_{\mathrm{Re}}\{\Psi_h(\omega)\}$');
228
229 mfprintf(fd, '%s\n', '\addplot[ color=black]');
230 mfprintf(fd, '%s\n', 'coordinates{');
231
232 for n = 1:data_length,
233     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n/data_length, ', ', real(
        fft_output_tree1(n,1))/normalization_freq1, ')');
234 end
235
236 mfprintf(fd, '%s\n', '};');
237
238 mfprintf(fd, '%s\n', '\end{axis}');
239 mfprintf(fd, '%s\n', '\end{tikzpicture}');
240 //-----
241 mfprintf(fd, '%s\n', '\begin{tikzpicture}');
242 mfprintf(fd, '%s\n', '\begin{axis}[');
243 mfprintf(fd, '%s\n', 'xlabel=$\omega/2\pi$, ');
244 mfprintf(fd, '%s\n', 'ylabel=_{\mathrm{Im}}\{\Psi_h(\omega)\}$');
245
246 mfprintf(fd, '%s\n', '\addplot[ color=black]');
247 mfprintf(fd, '%s\n', 'coordinates{');
248
249 for n = 1:data_length,
250     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n/data_length, ', ', imag(
        fft_output_tree1(n,1))/normalization_freq1, ')');
251 end
252
253 mfprintf(fd, '%s\n', '};');
254
255 mfprintf(fd, '%s\n', '\end{axis}');
256 mfprintf(fd, '%s\n', '\end{tikzpicture}');
257
258 //-----
259 mfprintf(fd, '%s\n', '\begin{tikzpicture}');
260 mfprintf(fd, '%s\n', '\begin{axis}[');
261 mfprintf(fd, '%s\n', 'xlabel=$\omega/2\pi$, ');
262 mfprintf(fd, '%s\n', 'ylabel=_{|\Psi_h(\omega)|^2}$');

```

```

263
264 mfprintf(fd, '%s_\n', '\addplot [ color=black] ');
265 mfprintf(fd, '%s_\n', 'coordinates{');
266
267 for n = 1:data_length,
268     mfprintf(fd, '%s_%.6f_%s_%.6f_%s_\n', '(' , n/data_length , ', ', (abs(
        fft_output_tree1(n,1)))^2/normalization_freq1^2, ')');
269 end
270
271 mfprintf(fd, '%s_\n', '};');
272
273 mfprintf(fd, '%s_\n', '\end{axis}');
274 mfprintf(fd, '%s_\n', '\end{tikzpicture}');
275 //—————
276
277 // impulse_response in Time
278 //—————
279 // Upper
280 //—————
281 mfprintf(fd, '%s_\n', '\newpage');
282 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
283 mfprintf(fd, '%s_\n', '\begin{axis}[');
284 mfprintf(fd, '%s_\n', 'xlabel=_$t/64$, ');
285 mfprintf(fd, '%s_\n', 'ylabel=_$\mathrm{Real}\{\psi_h(t)\}$');
286
287 mfprintf(fd, '%s_\n', '\addplot [ color=black] ');
288 mfprintf(fd, '%s_\n', 'coordinates{');
289
290 for n = 1:data_length,
291     mfprintf(fd, '%s_%.6f_%s_%.6f_%s_\n', '(' , n/data_length , ', ', real(
        output_tree1(n,1))/normalization_time1, ')');
292 end
293
294 mfprintf(fd, '%s_\n', '};');
295
296 mfprintf(fd, '%s_\n', '\end{axis}');
297 mfprintf(fd, '%s_\n', '\end{tikzpicture}');
298 //—————
299 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
300 mfprintf(fd, '%s_\n', '\begin{axis}[ymax=1.1, ymin=-0.1, ');
301 mfprintf(fd, '%s_\n', 'xlabel=_$t/64$, ');
302 mfprintf(fd, '%s_\n', 'ylabel=_$\mathrm{Imag}\{\psi_h(t)\}$');
303

```

```

304 mfprintf(fd, '%s\n', '\addplot [ color=black] ');
305 mfprintf(fd, '%s\n', 'coordinates{');
306
307 for n = 1:data_length,
308     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n/data_length , ', ', imag(
        output_tree1(n,1))/normalization_time1 , ')');
309 end
310
311 mfprintf(fd, '%s\n', '};');
312
313 mfprintf(fd, '%s\n', '\end{axis}');
314 mfprintf(fd, '%s\n', '\end{tikzpicture}');
315 //-----
316 mfprintf(fd, '%s\n', '\begin{tikzpicture}');
317 mfprintf(fd, '%s\n', '\begin{axis}[');
318 mfprintf(fd, '%s\n', 'xlabel=$t/64$, ');
319 mfprintf(fd, '%s\n', 'ylabel=_$|\psi_h(t)|$');
320
321 mfprintf(fd, '%s\n', '\addplot [ color=black] ');
322 mfprintf(fd, '%s\n', 'coordinates{');
323
324 for n = 1:data_length,
325     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n/data_length , ', ', abs(
        output_tree1(n,1))/normalization_time1 , ')');
326 end
327
328 mfprintf(fd, '%s\n', '};');
329 mfprintf(fd, '%s\n', '\end{axis}');
330 mfprintf(fd, '%s\n', '\end{tikzpicture}');
331 //-----
332
333
334 mfprintf(fd, '%s\n', '\end{center}');
335 mfprintf(fd, '%s\n', '\end{document}');
336 mclose(fd);

```

A.2.4 DT-CWT-based OFDM, with q-shift filters.

```
1  clc
2  clear
3
4  //-----
5  //CIRCULAR_SHIFT_BY_TWO
6  function [shifted_sequence] = CIRCULAR_SHIFT_BY_TWO(
       to_be_shifted_sequence),
7
8  to_be_shifted_sequence_length = length(to_be_shifted_sequence);
9
10 shifted_sequence = [ to_be_shifted_sequence(1,
       to_be_shifted_sequence_length-1:to_be_shifted_sequence_length),
       to_be_shifted_sequence(1,1:to_be_shifted_sequence_length-2)];
11
12 endfunction
13
14 //-----
15 // DISCRETE_WAVELET_TRANSFORM_MATRIX
16 function [discrete_wavelet_transform_matrix] =
       DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter , High_filter , zero_padding
       ),
17
18 Length = length(Low_filter) + zero_padding;
19
20 discrete_wavelet_transform_matrix = zeros(Length, Length);
21
22 discrete_wavelet_transform_matrix(1,1:Length) = [(Low_filter), zeros(1,
       zero_padding)];
23 discrete_wavelet_transform_matrix(1+Length/2,1:Length) = [(High_filter)
       , zeros(1, zero_padding)];
24
25 for row = 2:Length/2,
26     discrete_wavelet_transform_matrix(row,1:Length) =
           CIRCULAR_SHIFT_BY_TWO( discrete_wavelet_transform_matrix(row
           -1,1:Length));
27     discrete_wavelet_transform_matrix(row+Length/2,1:Length) =
           CIRCULAR_SHIFT_BY_TWO( discrete_wavelet_transform_matrix(row-1+
           Length/2,1:Length));
28     end,
29
30 endfunction
```

```

31
32 //-----
33 // INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX
34 function [inverse_discrete_wavelet_transform_matrix] =
    INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter , High_filter ,
    zero_padding) ,
35
36 inverse_discrete_wavelet_transform_matrix = (
    DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter , High_filter ,
    zero_padding)).';
37
38 endfunction
39
40 //-----
41 // Q_FUNCTION
42 function [q_function] = Q_FUNCTION(some_value) ,
43
44 q_function = 0.5*erfc(sqrt(0.5)*some_value);
45
46 endfunction
47
48 //-----
49 // AVERAGE
50 function [average] = AVERAGE(some_vector) ,
51
52 average = sum(some_vector)/length(some_vector);
53
54 endfunction
55
56 //-----
57 //wavelet = dbwavf('db1');// it is db6
58 //[low_analysis_filter , high_analysis_filter , low_synthesis_filter ,
    high_synthesis_filter] = orthfilt(wavelet);
59 //analysis_filter = [low_analysis_filter; high_analysis_filter];
60 //First_analysis_filter = analysis_filter;
61 //synthesis_filter = [low_synthesis_filter; high_synthesis_filter];
62 //Final_synthesis_filter = synthesis_filter;
63 //-----
64 [First_analysis_filter , Final_synthesis_filter] = FSfarras('f');
65 [analysis_filter , synthesis_filter] = dualfilt1('f');
66 //-----
67 //square_root_half = sqrt(.5);

```

```

68 //analysis_filter = [square_root_half, square_root_half ; -
    square_root_half, square_root_half];
69 //First_analysis_filter = analysis_filter;
70 //synthesis_filter = [square_root_half, square_root_half;
    square_root_half, -square_root_half];
71 //Final_synthesis_filter = synthesis_filter;
72 //-----
73 no_stages = 2;
74 filter_length = length(analysis_filter(1,:));
75 //-----
76 // data_length := data is in symbols
77 data_length = 64//filter_length*2^no_stages;//64
78
79 //multi_output_tree1 = zeros(data_length,1);
80 //multi_output_tree2 = zeros(data_length,1);
81
82 number_of_times_to_avg_on = 1;
83
84
85 //-----
86 // Data
87 //-----
88 data = zeros(data_length,1);
89 data(20, 1) = 1;
90 normalization = sqrt(10);
91 data_before_norm = data;
92 //data_before_norm = grand(data_length,2, 'uin',0,1);
93 //random = rand(data_before_norm, 'normal');
94 //for L = 1:data_length,
95 //   if random(L,1) > 0 then
96 //       data_before_norm(L,1) = 2*data_before_norm(L,1)+1;
97 //   else data_before_norm(L,1) = -1*(2*data_before_norm(L,1)+1);
98 //   end
99 //   if random(L,2) > 0 then
100 //       data_before_norm(L,2) = 2*data_before_norm(L,2)+1;
101 //   else data_before_norm(L,2) = -1*(2*data_before_norm(L,2)+1);
102 //   end
103
104 // end
105
106 //normalization = sqrt(10);
107 //data = data_before_norm; //normalized
108

```

```

109 //-----
110 zero_padding = abs(data_length - filter_length);
111
112
113 //-----
114 // Number of data per branch
115 //-----
116 branch_rate = zeros(1, no_stages+1); // to store # of ele. per branch
117 branch_rate(1,1) = data_length/2^no_stages;
118 for k = no_stages:-1:1,
119     branch_rate(1, no_stages+2-k) = data_length/2^k;
120 end
121 cumsum_branch_rate = cumsum(branch_rate); // matrix to store the
    cumulative sum up to kth elem.
122 //-----
123     data_copy = zeros(data_length,1);
124     constructed_data = zeros(data_length,1);
125
126
127 //-----
128 //-----
129 // the transmitter side
130 //-----
131 //-----
132
133
134 //-----
135 for tree_no = 1:2,
136
137     if tree_no == 1 then
138         low_filter = 1;
139         high_filter = 2;
140     elseif tree_no == 2 then
141         low_filter = 3;
142         high_filter = 4;
143     end // for elseif
144     //-----
145
146
147
148     data_copy = data; // data has to be of even length
149
150 //-----

```



```

151 // Stages from 1:no_stages-1
152 //-----
153 constructed_data(1:cumsum_branch_rate(1,2),1) = data_copy(1:
      cumsum_branch_rate(1,2),1);
154
155
156 for stage = 1:no_stages-1,
157
158     // data_length = cumsum_branch_rate(1,stage+1)
159     zero_padding = abs(cumsum_branch_rate(1,stage+1) - filter_length);
160
161
162     constructed_data(1:cumsum_branch_rate(1,stage+1),1) =
      INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX( flipdim(
      synthesis_filter(low_filter,:),2), flipdim(synthesis_filter(
      high_filter,:),2), zero_padding)*data_copy(1:cumsum_branch_rate(1,
      stage+1),1);
163 // filters are flipped, due to using same function of the DWT, and they
      have to be arranged in an inverse order.
164
165     data_copy(1:cumsum_branch_rate(1,stage+2),1) = [constructed_data(1:
      cumsum_branch_rate(1,stage+1),1); data_copy(cumsum_branch_rate(1,
      stage+1)+1:cumsum_branch_rate(1,stage+2),1)];
166
167 end
168
169 //-----
170 // Stage number no_stages, "last stage"
171 //-----
172
173 zero_padding = abs(cumsum_branch_rate(1,no_stages+1) - filter_length);
174
175
176 constructed_data(1:cumsum_branch_rate(1,no_stages+1),1) =
      INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX( flipdim(
      Final_synthesis_filter(low_filter,:),2), flipdim(
      Final_synthesis_filter(high_filter,:),2), zero_padding)*data_copy(1:
      cumsum_branch_rate(1,no_stages+1),1);
177 // filters are flipped, due to using same function of the DWT, and they
      have to be arranged in an inverse order.
178
179
180

```

```

181
182
183
184
185 //-----
186 // to store output of both trees
187 if tree_no == 1 then
188     output_tree1 = constructed_data
189
190 elseif tree_no == 2 then
191     output_tree2 = constructed_data
192
193 end // if tree_no
194
195
196
197
198
199
200 end // for tree_no
201
202 both_trees_output = output_tree1 + %i*output_tree2;
203
204
205
206
207
208 //-----
209
210
211
212 //-----
213 // FFTs
214 //-----
215
216
217
218 fft_output_tree1 = fft(output_tree1);
219 fft_output_tree2 = fft(output_tree2);
220 fft_output_both_trees = fft(both_trees_output);
221
222 //-----
223

```

```

224
225 // Plot using Tikz:
226 fd = mopen( '\Users\Tassniem\Documents\TikZ\IR\DTCWT_IR\DTCWT_IR.tex ', 'wt'
      );
227 mfprintf(fd, '%s\n', '\documentclass{article}');
228 mfprintf(fd, '%s\n', '\usepackage{tikz}');
229 mfprintf(fd, '%s\n', '\usepackage{pgfplots}');
230 mfprintf(fd, '%s\n', '\begin{document}');
231
232
233
234 mfprintf(fd, '%s\n', '\begin{center}');
235
236
237 normalization_time1 = max(abs(output_tree1))
238 normalization_time2 = max(abs(output_tree2))
239 normalization_time_DT = max(abs(both_trees_output))
240
241
242 normalization_freq1 = max(abs(fft_output_tree1))
243 normalization_freq2 = max(abs(fft_output_tree2))
244 normalization_freq_DT = max(abs(fft_output_both_trees))
245
246 //-----
247 // impulse_response in Frequency
248 //-----
249
250 mfprintf(fd, '%s\n', '\begin{tikzpicture}');
251 mfprintf(fd, '%s\n', '\begin{axis}[');
252 mfprintf(fd, '%s\n', 'xlabel=\omega/2\pi$, ');
253 mfprintf(fd, '%s\n', 'ylabel=_\mathrm{Real}\{\Psi_h(\omega)\}$ ');
254
255 mfprintf(fd, '%s\n', '\addplot[ color=black]');
256 mfprintf(fd, '%s\n', 'coordinates{');
257
258 for n = 1:data_length,
259     mfprintf(fd, '%s%.6f_%s%.6f_%s\n', '(' , n/data_length , ', ', real(
      fft_output_tree1(n,1))/normalization_freq1 , ')');
260 end
261
262 mfprintf(fd, '%s\n', '};');
263
264 mfprintf(fd, '%s\n', '\end{axis}');

```

```

265 mfprintf(fd, '%s_\n', '\end{tikzpicture}');
266 //-----
267 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
268 mfprintf(fd, '%s_\n', '\begin{axis}[');
269 mfprintf(fd, '%s_\n', 'xlabel=\omega/2\pi$, ');
270 mfprintf(fd, '%s_\n', 'ylabel=_$\mathrm{Imag}\{\Psi_h(\omega)\}$');
271
272 mfprintf(fd, '%s_\n', '\addplot [color=black]');
273 mfprintf(fd, '%s_\n', 'coordinates{');
274
275 for n = 1:data_length,
276     mfprintf(fd, '%s_%.6f_%s_%.6f_%s_\n', '(' , n/data_length , ', ', imag(
        fft_output_tree1(n,1))/normalization_freq1 , ')');
277 end
278
279 mfprintf(fd, '%s_\n', '};');
280
281 mfprintf(fd, '%s_\n', '\end{axis}');
282 mfprintf(fd, '%s_\n', '\end{tikzpicture}');
283
284 //-----
285 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
286 mfprintf(fd, '%s_\n', '\begin{axis}[');
287 mfprintf(fd, '%s_\n', 'xlabel=\omega/2\pi$, ');
288 mfprintf(fd, '%s_\n', 'ylabel=_$|\Psi_h(\omega)|^2$');
289
290 mfprintf(fd, '%s_\n', '\addplot [color=black]');
291 mfprintf(fd, '%s_\n', 'coordinates{');
292
293 for n = 1:data_length,
294     mfprintf(fd, '%s_%.6f_%s_%.6f_%s_\n', '(' , n/data_length , ', ', (abs(
        fft_output_tree1(n,1)))^2/normalization_freq1^2 , ')');
295 end
296
297 mfprintf(fd, '%s_\n', '};');
298
299 mfprintf(fd, '%s_\n', '\end{axis}');
300 mfprintf(fd, '%s_\n', '\end{tikzpicture}');
301 //-----
302 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
303 mfprintf(fd, '%s_\n', '\begin{axis}[');
304 mfprintf(fd, '%s_\n', 'xlabel=\omega/2\pi$, ');
305 mfprintf(fd, '%s_\n', 'ylabel=_$\mathrm{Real}\{\Psi_g(\omega)\}$');

```

```

306
307 mfprintf(fd, '%s_\n', '\addplot [ color=black] ');
308 mfprintf(fd, '%s_\n', 'coordinates{');
309
310 for n = 1:data_length,
311     mfprintf(fd, '%s_%.6f_%s_%.6f_%s_\n', '(' , n/data_length , ', ', real(
        fft_output_tree2(n,1))/normalization_freq2 , ')');
312 end
313
314 mfprintf(fd, '%s_\n', '};');
315 mfprintf(fd, '%s_\n', '\end{axis}');
316 mfprintf(fd, '%s_\n', '\end{tikzpicture}');
317 //-----
318 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
319 mfprintf(fd, '%s_\n', '\begin{axis}[');
320 mfprintf(fd, '%s_\n', 'xlabel=$\omega/2\pi$, ');
321 mfprintf(fd, '%s_\n', 'ylabel=_$\mathrm{Imag}\{\Psi_g(\omega)\}$');
322
323 mfprintf(fd, '%s_\n', '\addplot [ color=black] ');
324 mfprintf(fd, '%s_\n', 'coordinates{');
325
326 for n = 1:data_length,
327     mfprintf(fd, '%s_%.6f_%s_%.6f_%s_\n', '(' , n/data_length , ', ', imag(
        fft_output_tree2(n,1))/normalization_freq2 , ')');
328 end
329
330 mfprintf(fd, '%s_\n', '};');
331 mfprintf(fd, '%s_\n', '\end{axis}');
332 mfprintf(fd, '%s_\n', '\end{tikzpicture}');
333
334 //-----
335 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
336 mfprintf(fd, '%s_\n', '\begin{axis}[');
337 mfprintf(fd, '%s_\n', 'xlabel=$\omega/2\pi$, ');
338 mfprintf(fd, '%s_\n', 'ylabel=_$|\Psi_g(\omega)|^2$');
339
340 mfprintf(fd, '%s_\n', '\addplot [ color=black] ');
341 mfprintf(fd, '%s_\n', 'coordinates{');
342
343 for n = 1:data_length,
344     mfprintf(fd, '%s_%.6f_%s_%.6f_%s_\n', '(' , n/data_length , ', ', (abs(
        fft_output_tree2(n,1)))^2/normalization_freq2^2 , ')');
345 end

```

```

346
347 mfprintf(fd , '%s_\n' , '};');
348
349 mfprintf(fd , '%s_\n' , '\end{axis}');
350 mfprintf(fd , '%s_\n' , '\end{tikzpicture}');
351 //—————
352
353 mfprintf(fd , '%s_\n' , '\begin{tikzpicture}');
354 mfprintf(fd , '%s_\n' , '\begin{axis}[');
355 mfprintf(fd , '%s_\n' , 'xlabel=$\omega/2\pi$ , ');
356 mfprintf(fd , '%s_\n' , 'ylabel=_$\mathrm{Real}\{\Psi_h(\omega)_{-+j}\Psi_g(\omega)\}$');
357
358 mfprintf(fd , '%s_\n' , '\addplot [ color=black]');
359 mfprintf(fd , '%s_\n' , 'coordinates{');
360
361 for n = 1:data_length ,
362     mfprintf(fd , '%s_%.6f_%.6f_%.6f_\n' , '(' , n/data_length , ', , real(
        fft_output_both_trees(n,1))/normalization_freq_DT , ')');
363 end
364
365 mfprintf(fd , '%s_\n' , '};');
366 mfprintf(fd , '%s_\n' , '\end{axis}');
367 mfprintf(fd , '%s_\n' , '\end{tikzpicture}');
368 //—————
369
370 mfprintf(fd , '%s_\n' , '\begin{tikzpicture}');
371 mfprintf(fd , '%s_\n' , '\begin{axis}[');
372 mfprintf(fd , '%s_\n' , 'xlabel=$\omega/2\pi$ , ');
373 mfprintf(fd , '%s_\n' , 'ylabel=_$\mathrm{Imag}\{\Psi_h(\omega)_{-+j}\Psi_g(\omega)\}$');
374
375 mfprintf(fd , '%s_\n' , '\addplot [ color=black]');
376 mfprintf(fd , '%s_\n' , 'coordinates{');
377
378 for n = 1:data_length ,
379     mfprintf(fd , '%s_%.6f_%.6f_%.6f_\n' , '(' , n/data_length , ', , imag(
        fft_output_both_trees(n,1))/normalization_freq_DT , ')');
380 end
381
382 mfprintf(fd , '%s_\n' , '};');
383 mfprintf(fd , '%s_\n' , '\end{axis}');
384 mfprintf(fd , '%s_\n' , '\end{tikzpicture}');

```

```

385 //-----
386 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
387 mfprintf(fd, '%s_\n', '\begin{axis}[');
388 mfprintf(fd, '%s_\n', 'xlabel=$\omega/2\pi$, ');
389 mfprintf(fd, '%s_\n', 'ylabel=_$\Psi_h(\omega)_{+_{j}}\Psi_g(\omega)^2$');
390
391 mfprintf(fd, '%s_\n', '\addplot[ color=black]');
392 mfprintf(fd, '%s_\n', 'coordinates{');
393
394 for n = 1:data_length,
395     mfprintf(fd, '%s_%.6f_%s_%.6f_%s_\n', '(' , n/data_length , ', ', (abs(
        fft_output_both_trees(n,1)))^2/normalization_freq_DT^2, ')');
396 end
397
398 mfprintf(fd, '%s_\n', '};');
399 mfprintf(fd, '%s_\n', '\end{axis}');
400 mfprintf(fd, '%s_\n', '\end{tikzpicture}');
401 //-----
402 // impulse_response in Time
403 //-----
404 // Upper
405 //-----
406 mfprintf(fd, '%s_\n', '\newpage');
407 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
408 mfprintf(fd, '%s_\n', '\begin{axis}[');
409 mfprintf(fd, '%s_\n', 'xlabel=_$t/64$, ');
410 mfprintf(fd, '%s_\n', 'ylabel=_$\mathrm{Real}\{\psi_h(t)\}$');
411
412 mfprintf(fd, '%s_\n', '\addplot[ color=black]');
413 mfprintf(fd, '%s_\n', 'coordinates{');
414
415 for n = 1:data_length,
416     mfprintf(fd, '%s_%.6f_%s_%.6f_%s_\n', '(' , n/data_length , ', ', real(
        output_tree1(n,1))/normalization_time1, ')');
417 end
418
419 mfprintf(fd, '%s_\n', '};');
420
421 mfprintf(fd, '%s_\n', '\end{axis}');
422 mfprintf(fd, '%s_\n', '\end{tikzpicture}');
423 //-----
424 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
425 mfprintf(fd, '%s_\n', '\begin{axis}[ymax=1.1, ymin=-0.1, ');

```

```

426 mfprintf(fd, '%s_\n', 'xlabel=_$t/64$, ');
427 mfprintf(fd, '%s_\n', 'ylabel=_$\mathrm{Imag}\{\psi_h(t)\}$ ');
428
429 mfprintf(fd, '%s_\n', '\addplot [ color=black] ');
430 mfprintf(fd, '%s_\n', 'coordinates{ ');
431
432 for n = 1:data_length,
433     mfprintf(fd, '%s_%.6f_%s_%.6f_%s_\n', '( ', n/data_length, ', ', imag(
        output_tree1(n,1))/normalization_time1, ') ');
434 end
435
436 mfprintf(fd, '%s_\n', '}; ');
437
438 mfprintf(fd, '%s_\n', '\end{axis} ');
439 mfprintf(fd, '%s_\n', '\end{tikzpicture} ');
440 //—————
441 mfprintf(fd, '%s_\n', '\begin{tikzpicture} ');
442 mfprintf(fd, '%s_\n', '\begin{axis} [ ');
443 mfprintf(fd, '%s_\n', 'xlabel=$t/64$, ');
444 mfprintf(fd, '%s_\n', 'ylabel=_$|\psi_h(t)|$ ');
445
446 mfprintf(fd, '%s_\n', '\addplot [ color=black] ');
447 mfprintf(fd, '%s_\n', 'coordinates{ ');
448
449 for n = 1:data_length,
450     mfprintf(fd, '%s_%.6f_%s_%.6f_%s_\n', '( ', n/data_length, ', ', abs(
        output_tree1(n,1))/normalization_time1, ') ');
451 end
452
453 mfprintf(fd, '%s_\n', '}; ');
454 mfprintf(fd, '%s_\n', '\end{axis} ');
455 mfprintf(fd, '%s_\n', '\end{tikzpicture} ');
456 //—————
457 //—————
458 // Lower
459 //—————
460 mfprintf(fd, '%s_\n', '\newpage ');
461 mfprintf(fd, '%s_\n', '\begin{tikzpicture} ');
462 mfprintf(fd, '%s_\n', '\begin{axis} [ ');
463 mfprintf(fd, '%s_\n', 'xlabel=_$t/64$, ');
464 mfprintf(fd, '%s_\n', 'ylabel=_$\mathrm{Real}\{\psi_g(t)\}$ ');
465
466 mfprintf(fd, '%s_\n', '\addplot [ color=black] ');

```



```

467 mfprintf(fd, '%s\n', 'coordinates{');
468
469 for n = 1:data_length,
470     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n/data_length, ', ', real(
        output_tree2(n,1))/normalization_time2, ')');
471 end
472
473 mfprintf(fd, '%s\n', '};');
474 mfprintf(fd, '%s\n', '\end{axis}');
475 mfprintf(fd, '%s\n', '\end{tikzpicture}');
476 //—————
477 mfprintf(fd, '%s\n', '\begin{tikzpicture}');
478 mfprintf(fd, '%s\n', '\begin{axis}[ymax=1.1, ymin=-0.1,');
479 mfprintf(fd, '%s\n', 'xlabel=$t/64$,');
480 mfprintf(fd, '%s\n', 'ylabel=$_{\mathrm{Imag}}\{\psi_g(t)\}$');
481
482 mfprintf(fd, '%s\n', '\addplot[ color=black]');
483 mfprintf(fd, '%s\n', 'coordinates{');
484
485 for n = 1:data_length,
486     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n/data_length, ', ', imag(
        output_tree2(n,1))/normalization_time2, ')');
487 end
488
489 mfprintf(fd, '%s\n', '};');
490 mfprintf(fd, '%s\n', '\end{axis}');
491 mfprintf(fd, '%s\n', '\end{tikzpicture}');
492 //—————
493
494 //—————
495
496 mfprintf(fd, '%s\n', '\begin{tikzpicture}');
497 mfprintf(fd, '%s\n', '\begin{axis}[');
498 mfprintf(fd, '%s\n', 'xlabel=$t/64$,');
499 mfprintf(fd, '%s\n', 'ylabel=$_{|\psi_g(t)|}$');
500
501 mfprintf(fd, '%s\n', '\addplot[ color=black]');
502 mfprintf(fd, '%s\n', 'coordinates{');
503
504 for n = 1:data_length,
505     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n/data_length, ', ', abs(
        output_tree2(n,1))/normalization_time2, ')');
506 end

```

```

507
508 mfprintf(fd, '%s_\n', '};');
509 mfprintf(fd, '%s_\n', '\end{axis}');
510 mfprintf(fd, '%s_\n', '\end{tikzpicture}');
511 //-----
512 // Both
513 //-----
514 mfprintf(fd, '%s_\n', '\newpage');
515 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
516 mfprintf(fd, '%s_\n', '\begin{axis}[');
517 mfprintf(fd, '%s_\n', 'xlabel=$t/64$, ');
518 mfprintf(fd, '%s_\n', 'ylabel=_$\mathrm{Re}\{\psi_h(t)_{-+j}\psi_g(t)\}$]')
    ;
519
520 mfprintf(fd, '%s_\n', '\addplot[ color=black]');
521 mfprintf(fd, '%s_\n', 'coordinates{');
522
523 for n = 1:data_length,
524     mfprintf(fd, '%s_%.6f_%s_%.6f_%s_\n', '(' , n/data_length , ', ', real(
        both_trees_output(n,1))/normalization_time1 , ')');
525 end
526
527 mfprintf(fd, '%s_\n', '};');
528 mfprintf(fd, '%s_\n', '\end{axis}');
529 //-----
530 mfprintf(fd, '%s_\n', '\end{tikzpicture}');
531 mfprintf(fd, '%s_\n', '\begin{tikzpicture}');
532 mfprintf(fd, '%s_\n', '\begin{axis}[');
533 mfprintf(fd, '%s_\n', 'xlabel=$t/64$, ');
534 mfprintf(fd, '%s_\n', 'ylabel=_$\mathrm{Im}\{\psi_h(t)_{-+j}\psi_g(t)\}$]')
    ;
535
536 mfprintf(fd, '%s_\n', '\addplot[ color=black]');
537 mfprintf(fd, '%s_\n', 'coordinates{');
538
539 for n = 1:data_length,
540     mfprintf(fd, '%s_%.6f_%s_%.6f_%s_\n', '(' , n/data_length , ', ', imag(
        both_trees_output(n,1))/normalization_time2 , ')');
541 end
542
543 mfprintf(fd, '%s_\n', '};');
544 mfprintf(fd, '%s_\n', '\end{axis}');
545 mfprintf(fd, '%s_\n', '\end{tikzpicture}');

```

```

546 //-----
547
548 //-----
549
550 mfprintf(fd, '%s\n', '\begin{tikzpicture}');
551 mfprintf(fd, '%s\n', '\begin{axis}[');
552 mfprintf(fd, '%s\n', 'xlabel=$t/64$, ');
553 mfprintf(fd, '%s\n', 'ylabel=_{\psi_h(t)}+_{\psi_g(t)}|$\ ');
554
555 mfprintf(fd, '%s\n', '\addplot[ color=black] ');
556 mfprintf(fd, '%s\n', 'coordinates{');
557
558 for n = 1:data_length,
559     mfprintf(fd, '%s%.6f%s%.6f%s\n', '( ', n/data_length, ', ', abs(
        both_trees_output(n,1))/normalization_time_DT, ')');
560 end
561
562 mfprintf(fd, '%s\n', '}; ');
563 mfprintf(fd, '%s\n', '\end{axis}');
564 mfprintf(fd, '%s\n', '\end{tikzpicture}');
565 //-----
566
567
568 mfprintf(fd, '%s\n', '\end{center}');
569 mfprintf(fd, '%s\n', '\end{document}');
570 mclose(fd);

```

A.3 Peak to Average Power Ratio (PAPR) Code for OFDM Alternatives

```

1  clc
2  clear
3  //-----
4  //CIRCULAR_SHIFT_BY_TWO
5  function [shifted_sequence] = CIRCULAR_SHIFT_BY_TWO(
        to_be_shifted_sequence),
6
7  to_be_shifted_sequence_length = length(to_be_shifted_sequence);
8
9  shifted_sequence = [ to_be_shifted_sequence(1,
        to_be_shifted_sequence_length - 1:to_be_shifted_sequence_length),
        to_be_shifted_sequence(1,1:to_be_shifted_sequence_length - 2)];
10
11 endfunction
12 //-----
13 // MATRIX_EVALUATION
14 function [matrix_evaluation] = MATRIX_EVALUATION(transformation_length ,
        filters),
15 [r, c] = size(filters);
16 matrix_evaluation = zeros(transformation_length , transformation_length)
        ;
17
18 matrix_evaluation(1,1:c) = filters(1,1:c);
19 matrix_evaluation(1+transformation_length/2,1:c) = filters(2,1:c);
20
21 for row = 2:transformation_length/2,
22     matrix_evaluation(row,1:c) = CIRCULAR_SHIFT_BY_TWO(
        matrix_evaluation(row-1,1:c));
23     matrix_evaluation(row+transformation_length/2,1:c) =
        CIRCULAR_SHIFT_BY_TWO( matrix_evaluation(row-1+
        transformation_length/2,1:c));
24     end
25 endfunction
26 //-----
27 // DWT
28 function [dwt_matrix] = DWT(data_length , filter_length , no_stages , filters) ,
29
30     dwt_matrix = eye(data_length , data_length);
31

```

```

32  if no_stages > 1 then
33
34      dwt_matrix(1:data_length/2^no_stages,1:data_length/2^no_stages) =
          MATRIX_EVALUATION(data_length/2^no_stages,[filters(3:4,:), zeros
          (2,data_length/2^no_stages-filter_length)]);
35
36  if no_stages > 2 then
37      for stage = no_stages-1:-1:2,
38          matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters(3:4,:),
          zeros(2,data_length/2^stage-filter_length)]);
39      dwt_matrix(1:data_length/2^(stage+1),1:data_length/2^stage) =
          dwt_matrix(1:data_length/2^(stage+1),1:data_length/2^(stage+1))
          *matrix_i(1:data_length/2^(stage+1),1:data_length/2^stage);
40      dwt_matrix(data_length/2^stage/2+1:data_length/2^stage,1:
          data_length/2^stage) = matrix_i(data_length/2^stage/2+1:
          data_length/2^stage,1:data_length/2^stage);
41  end
42  end
43  end
44  matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,
          data_length-filter_length)]);
45  dwt_matrix(1:data_length/2,1:data_length) = dwt_matrix(1:data_length
          /2,1:data_length/2)*matrix_i(1:data_length/2,1:data_length);
46  dwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(
          data_length/2+1:data_length,1:data_length);
47
48  endfunction
49  //-----
50  // IDWT
51  function [idwt_matrix] = IDWT(data_length,filter_length,no_stages,filters
          ),
52
53      idwt_matrix = eye(data_length,data_length);
54
55      if no_stages > 1 then
56          idwt_matrix(1:data_length/2^no_stages,1:data_length/2^no_stages) =
          MATRIX_EVALUATION(data_length/2^no_stages,[filters(3:4,:), zeros
          (2,data_length/2^no_stages-filter_length)]);
57
58      if no_stages > 2 then
59          for stage = no_stages-1:-1:2,
60              matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters
          (3:4,:),zeros(2,data_length/2^stage-filter_length)]);

```

```

61         idwt_matrix(1:data_length/2^(stage+1),1:data_length/2^stage) =
            idwt_matrix(1:data_length/2^(stage+1),1:data_length/2^(stage
+1))*matrix_i(1:data_length/2^(stage+1),1:data_length/2^
stage);
62         idwt_matrix(data_length/2^stage/2+1:data_length/2^stage,1:
            data_length/2^stage) = matrix_i(data_length/2^stage/2+1:
            data_length/2^stage,1:data_length/2^stage);
63     end
64     end
65     end
66     matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,
            data_length-filter_length)]);
67     idwt_matrix(1:data_length/2,1:data_length) = idwt_matrix(1:
            data_length/2,1:data_length/2)*matrix_i(1:data_length/2,1:
            data_length);
68     idwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(
            data_length/2+1:data_length,1:data_length);
69
70     idwt_matrix = idwt_matrix.';
71     endfunction
72     //-----
73     // Q_FUNCTION
74     function [q_function] = Q_FUNCTION(some_value),
75
76     q_function = 0.5*erfc(sqrt(0.5)*some_value);
77
78     endfunction
79     //-----
80     // AVERAGE
81     function [average] = AVERAGE(some_vector),
82
83     average = sum(some_vector)/length(some_vector);
84
85     endfunction
86     //-----
87     stacksize(800000)
88     number_of_OFDM_symbols = 3*10^5;
89     data_length = 64;
90     number_channels = 64;
91     sqrt_no_channels = sqrt(number_channels);
92     data_before_norm = zeros(data_length,2);
93     PAPR_range = 0:.1:10;
94     length_PAPR_range = length(PAPR_range);

```

```

95 //-----
96 // counter:
97 counter = zeros(length_PAPR_range,6);
98 // transform
99 output = zeros(data_length,1);
100 // power:
101 p_power_output = zeros(data_length,1);
102 // Average power:
103 average_power_output = zeros(1,1);
104 //-----
105 for no_of_OFDM_symbols = 1:number_of_OFDM_symbols
106 //-----
107 // Data
108 //-----
109 data_before_norm = grand(data_length,2,'uin',0,1);
110 random = rand(data_before_norm,'normal');
111 for L = 1:data_length,
112     if random(L,1) > 0 then
113         data_before_norm(L,1) = 2*data_before_norm(L,1)+1;
114     else data_before_norm(L,1) = -1*(2*data_before_norm(L,1)+1);
115     end
116     if random(L,2) > 0 then
117         data_before_norm(L,2) = 2*data_before_norm(L,2)+1;
118     else data_before_norm(L,2) = -1*(2*data_before_norm(L,2)+1);
119     end
120 end
121 normalization = sqrt(10);
122 data = (data_before_norm(1:data_length,1) + %i*data_before_norm(1:
123     data_length,2))/normalization; //normalized
124 //-----
125 for transformation = 1:6,
126 // 1:DFT
127 // 2:DWF-Haar
128 // 3:DWF-D6
129 // 4:Upper-DT-CWT
130 // 5:Lower-DT-CWT
131 //-----
132 if transformation == 1 then // 1:DFT
133     output = sqrt_no_channels*ifft(data);
134 end
135 //-----
136 if transformation == 2 then // 2:DWF-Haar

```

```

137     no_stages = 4 ;
138     low_filter = 1;
139     high_filter = 2;
140     wavelet = dbwavf('db1'); // it is Haar
141     [low_analysis_filter , high_analysis_filter , low_synthesis_filter ,
        high_synthesis_filter] = orthfilt(wavelet);
142     analysis_filter = [low_analysis_filter; high_analysis_filter];
143     First_analysis_filter = analysis_filter;
144     synthesis_filter = [low_synthesis_filter; high_synthesis_filter];
145     Final_synthesis_filter = synthesis_filter;
146
147     elseif transformation == 3 then // 3:DWT-D6
148         no_stages = 3 ;
149         low_filter = 1;
150         high_filter = 2;
151         wavelet = dbwavf('db3'); // it is D-6
152         [low_analysis_filter , high_analysis_filter , low_synthesis_filter ,
            high_synthesis_filter] = orthfilt(wavelet);
153         analysis_filter = [low_analysis_filter; high_analysis_filter];
154         First_analysis_filter = analysis_filter;
155         synthesis_filter = [low_synthesis_filter; high_synthesis_filter];
156         Final_synthesis_filter = synthesis_filter;
157
158     elseif transformation == 4 then // 4:Upper-DT-CWT
159         no_stages = 2;
160         low_filter = 1;
161         high_filter = 2;
162         [First_analysis_filter , Final_synthesis_filter] = FSfarras('f');
163         [analysis_filter , synthesis_filter] = dualfilt1('f');
164
165     elseif transformation == 5 then // 5:Lower-DT-CWT
166         no_stages = 2;
167         low_filter = 3;
168         high_filter = 4;
169         [First_analysis_filter , Final_synthesis_filter] = FSfarras('f');
170         [analysis_filter , synthesis_filter] = dualfilt1('f');
171
172     end
173     //-----
174     if transformation ~ = 1 & transformation ~ = 6 then
175
176         ANALYSIS_FILTERS = [First_analysis_filter(low_filter:high_filter,:)
            ; analysis_filter(low_filter:high_filter,:)];

```



```

177
178     SYNTHESIS_FILTERS = [ Final_synthesis_filter(low_filter:high_filter
179         ,:); synthesis_filter(low_filter:high_filter ,: )];
180
181     [row, filter_length] = size(analysis_filter(1,:));
182
183     output = IDWT(data_length, filter_length, no_stages, flipdim(
184         SYNTHESIS_FILTERS,2))*data;
185
186     end
187
188     if transformation == 6 then
189         power_output = abs(output.*conj(output))/data_length;
190
191         average_power_output = AVERAGE(power_output);
192         p_power_output = power_output/average_power_output;
193     end
194
195     if transformation == 4 then p_output_Upper = p_power_output;
196     elseif transformation == 5 then p_output_Lower = p_power_output;
197     end
198
199     if transformation == 6 then
200         power_output = p_output_Upper + p_output_Lower;
201         average_power_output = AVERAGE(power_output);
202         p_power_output = power_output/average_power_output;
203     end
204
205     for DL = 1:data_length
206         location = 1;
207         for PAPR_0 = PAPR_range
208             if p_power_output(data_length,1) > PAPR_0 then counter(location,
209                 transformation) = counter(location,transformation) + 1;
210             location = location + 1;
211         end
212     end
213     end
214
215     counter = counter/number_of_OFDM_symbols/data_length;
216     PAPR_range_sub = 0:.5:10;
217     // Plot using Tikz:

```

```

217 fd = mopen( '\Users\Tassniem\Documents\TikZ\PAPR\PAPR_figure.tex', 'wt' );
218 mfprintf( fd, '%s\n', '\documentclass{article}' );
219 mfprintf( fd, '%s\n', '\usepackage{tikz}' );
220 mfprintf( fd, '%s\n', '\usepackage{pgfplots}' );
221 mfprintf( fd, '%s\n', '\begin{document}' );
222 mfprintf( fd, '%s\n', '\begin{center}' );
223 //-----
224 mfprintf( fd, '%s\n', '\begin{tikzpicture}' );
225 mfprintf( fd, '%s\n', '\begin{semilogyaxis}[xmin=-0.5, xmax=10, width=12cm
, ' );
226 mfprintf( fd, '%s\n', 'xlabel=\mathrm{PAPR}_0$, ' );
227 mfprintf( fd, '%s\n', 'ylabel=\mathbf{P}_-\mathrm{PAPR}$ ] ' );
228 mfprintf( fd, '%s\n', '\pgfplotsset{every axis legend/.append style={at
={ (0.72,0.66) }, anchor=south }}, ' );
229 // "Marks only"
230 //-----
231 // DFT
232 //-----
233 mfprintf( fd, '%s\n', '\addplot[ color=black, mark=*, only marks ] ' );
234 mfprintf( fd, '%s\n', 'coordinates{ ' );
235 m = 1;
236 for n = PAPR_range_sub,
237     mfprintf( fd, '%s%.6f%s%.6f%s\n', '(', n, ', ', counter(m,1), ') ' );
238     m = m + 5;
239 end
240 mfprintf( fd, '%s\n', '}; ' );
241 //-----
242 mfprintf( fd, '%s\n', '\addlegendentry{DFT-Based} ' );
243 //-----
244 // DWT-Haar
245 //-----
246 mfprintf( fd, '%s\n', '\addplot[ color=black, mark=+, only marks ] ' );
247 mfprintf( fd, '%s\n', 'coordinates{ ' );
248 m = 1;
249 for n = PAPR_range_sub,
250     mfprintf( fd, '%s%.6f%s%.6f%s\n', '(', n, ', ', counter(m,2), ') ' );
251     m = m + 5;
252 end
253 mfprintf( fd, '%s\n', '}; ' );
254 //-----
255 mfprintf( fd, '%s\n', '\addlegendentry{DWT-Based, Haar filters} ' );
256 //-----
257 // DWT-D6

```

```

258 //-----
259 mfprintf(fd, '%s\n', '\addplot [ color=black , mark=x , only _marks ] ');
260 mfprintf(fd, '%s\n', 'coordinates{ ');
261 m = 1;
262 for n = PAPR_range_sub ,
263     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n , ', ' , counter(m,3) , ') ');
264     m = m + 5;
265 end
266 mfprintf(fd, '%s\n', '}; ');
267 //-----
268 mfprintf(fd, '%s\n', '\addlegentry {DWT-Based , _D-6_filters } ');
269 //-----
270 // DT-CWT
271 //-----
272 mfprintf(fd, '%s\n', '\addplot [ color=black , mark=triangle , only _marks ] ');
273 mfprintf(fd, '%s\n', 'coordinates{ ');
274 m = 1;
275 for n = PAPR_range_sub ,
276     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n , ', ' , counter(m,4) , ') ');
277     m = m + 5;
278 end
279 mfprintf(fd, '%s\n', '}; ');
280 //-----
281 mfprintf(fd, '%s\n', '\addlegentry {DT-CWT-Based , _Upper-tree } ');
282 //-----
283 //-----
284 mfprintf(fd, '%s\n', '\addplot [ color=black , mark=o , only _marks ] ');
285 mfprintf(fd, '%s\n', 'coordinates{ ');
286 m = 1;
287 for n = PAPR_range_sub ,
288     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n , ', ' , counter(m,5) , ') ');
289     m = m + 5;
290 end
291 mfprintf(fd, '%s\n', '}; ');
292 //-----
293 mfprintf(fd, '%s\n', '\addlegentry {DT-CWT-Based , _Lower-tree } ');
294 //-----
295 //-----
296 mfprintf(fd, '%s\n', '\addplot [ color=black , mark=square , only _marks ] ');
297 mfprintf(fd, '%s\n', 'coordinates{ ');
298 m = 1;
299 for n = PAPR_range_sub ,
300     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n , ', ' , counter(m,6) , ') ');

```

```

301     m = m + 5;
302 end
303 mfprintf(fd, '%s\n', '};');
304 //-----
305 mfprintf(fd, '%s\n', '\addlegendentry{DT-CWT-Based, Dual-tree}');
306 //” Without marks”
307 //-----
308 // DFT
309 //-----
310 mfprintf(fd, '%s\n', '\addplot [color=black]');
311 mfprintf(fd, '%s\n', 'coordinates{');
312 m = 0;
313 for n = PAPR_range,
314     m = m + 1;
315     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n, ', ', counter(m,1), ')');
316 end
317 mfprintf(fd, '%s\n', '};');
318 //-----
319 // DWT-Haar
320 //-----
321 mfprintf(fd, '%s\n', '\addplot [color=black]');
322 mfprintf(fd, '%s\n', 'coordinates{');
323 m = 0;
324 for n = PAPR_range,
325     m = m + 1;
326     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n, ', ', counter(m,2), ')');
327 end
328 mfprintf(fd, '%s\n', '};');
329 //-----
330 // DWT-D6
331 //-----
332 mfprintf(fd, '%s\n', '\addplot [color=black]');
333 mfprintf(fd, '%s\n', 'coordinates{');
334 m = 0;
335 for n = PAPR_range,
336     m = m + 1;
337     mfprintf(fd, '%s%.6f%s%.6f%s\n', '(' , n, ', ', counter(m,3), ')');
338 end
339 mfprintf(fd, '%s\n', '};');
340 //-----
341 // DT-CWT
342 //-----
343 mfprintf(fd, '%s\n', '\addplot [color=black]');

```

```

344 mfprintf(fd , '%s_\n' , 'coordinates{');
345 m = 0;
346 for n = PAPR_range ,
347     m = m + 1;
348     mfprintf(fd , '%s_%.6f_%s_%.6f_%s_\n' , '(' , n , ',' , counter(m,4) , ')');
349 end
350 mfprintf(fd , '%s_\n' , '};');
351 //—————
352 //—————
353 mfprintf(fd , '%s_\n' , '\addplot [ color=black]');
354 mfprintf(fd , '%s_\n' , 'coordinates{');
355 m = 0;
356 for n = PAPR_range ,
357     m = m + 1;
358     mfprintf(fd , '%s_%.6f_%s_%.6f_%s_\n' , '(' , n , ',' , counter(m,5) , ')');
359 end
360 mfprintf(fd , '%s_\n' , '};');
361 //—————
362 //—————
363 mfprintf(fd , '%s_\n' , '\addplot [ color=black]');
364 mfprintf(fd , '%s_\n' , 'coordinates{');
365 m = 0;
366 for n = PAPR_range ,
367     m = m + 1;
368     mfprintf(fd , '%s_%.6f_%s_%.6f_%s_\n' , '(' , n , ',' , counter(m,6) , ')');
369 end
370 mfprintf(fd , '%s_\n' , '};');
371 //—————
372 mfprintf(fd , '%s_\n' , '\end{semilogyaxis}');
373 mfprintf(fd , '%s_\n' , '\end{tikzpicture}');
374 //—————
375 mfprintf(fd , '%s_\n' , '\end{center}');
376 mfprintf(fd , '%s_\n' , '\end{document}');
377 mclose(fd);

```

VITA

Tassniem Rashed received her Bachelors of Science degree in Telecommunication Engineering from Mutah University, Jordan. She is seeking a graduate degree in the department of Electrical Engineering at the University of Mississippi. Her research interest includes communication systems modeling, digital signal processing, and wavelet applications in telecommunications and networks. From Aug 2008 to Dec 2011, she was mainly a Research Assistant at the Center for Wireless Communications at the University of Mississippi.