2012

Analysis of Wavelet Based Alternatives for OFDM

Tassniem Rashed

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

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

[1] *The 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

[2] Quran 7:43

Table of Contents

List of Tables

List of Figures

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 $x[n], n = 0, 1, \ldots, N-1$, is defined as

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

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

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

The operation of the DFT in matrix representation is

$$X = Qx, \tag{2.1.3}$$

where $X = [X_0, \ldots, X_{N-1}]^T$, $x = [x_0, \ldots, x_{N-1}]^T$, 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

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

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

$$x = Q^{-1}X = Q^H X, \tag{2.1.5}$$

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, $X = [X_0, X_1, \ldots, X_{N-1}]^T$. 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,

$$x = Q^{-1}X, \tag{2.1.6}$$

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

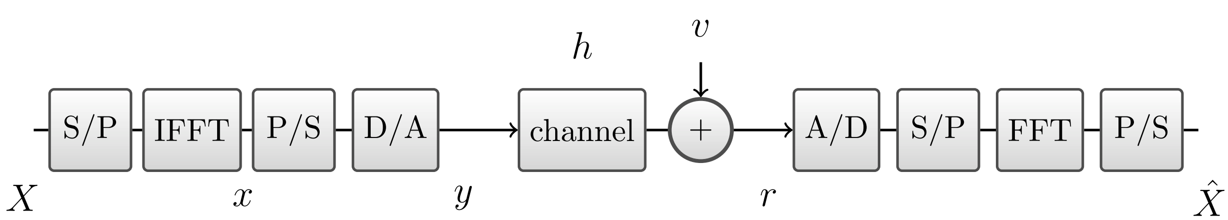


Figure 2.1. FFT-based OFDM system model.

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

$$r = hx + v, \tag{2.1.7}$$

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

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

$$r = x + v. \tag{2.1.8}$$

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

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

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

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

$$= \mathcal{N}_0 B_n. \tag{2.1.12}$$

Similarly, the subchannel noise power is

$$\sigma_k^2 = \mathcal{N}_0 B_k. \tag{2.1.13}$$

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

$$\sigma_k^2 = \sigma_n^2 / N, \tag{2.1.14}$$

Accordingly, each individual subchannel has one N^{th} the in-channel noise power σ_n^2. As a result, the covariance matrix[1] of the vector v_Q can be represented as

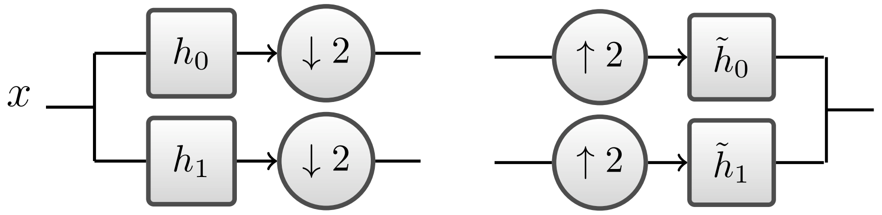
$$\begin{aligned}
\mathrm{cov}(v_Q) &= \frac{1}{N}\begin{bmatrix} \mathrm{Var}(\sum_{k=0}^{N-1} v_k) & \dots & \mathrm{Cov}(\sum_{k=0}^{N-1} v_k, \sum_{k=0}^{N-1} v_k W_N^{(N-1)k}) \\ \vdots & \ddots & \vdots \\ \mathrm{Cov}(\sum_{k=0}^{N-1} v_k, \sum_{k=0}^{N-1} v_k W_N^{(N-1)k}) & \dots & \mathrm{Var}(\sum_{k=0}^{N-1}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

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

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



(a) Analysis filter bank (b) Synthesis filter bank

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 family[2] 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

[2]wavelet and scaling functions.

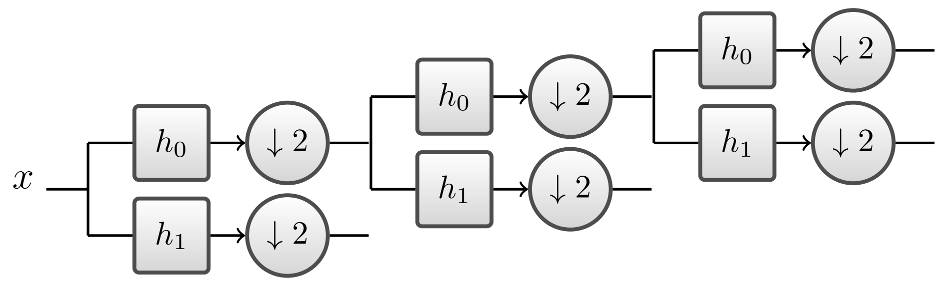


Figure 2.3. Three stage DWT tree structure, analysis.

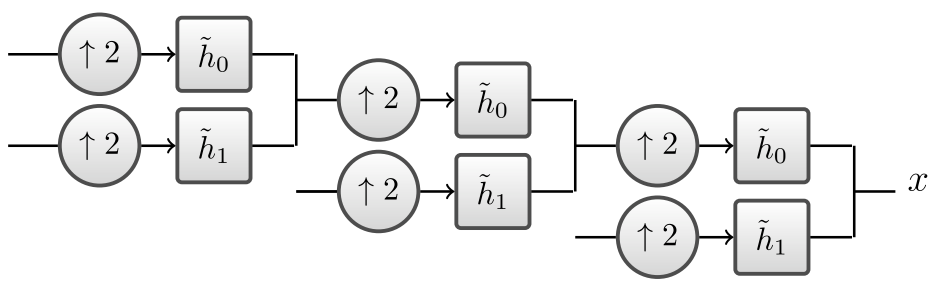


Figure 2.4. Three stage IDWT tree structure, synthesis.

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

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

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

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

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, $X = [X_0, X_1, \ldots, X_{N-1}]^T$, 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

$$X = W_N x, \tag{2.2.3}$$

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

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

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

$$W_R = \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} L_J \\ H_J \end{bmatrix} & L_{J-1} \\ H_{J-1} & \end{bmatrix} & \ddots \\ \ddots & \end{bmatrix} & L_2 \\ H_2 & \end{bmatrix} & L_1 \\ H_1 & \end{bmatrix}, \tag{2.2.5}$$

and

$$\tilde{W}_R^{-1} = \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} \tilde{L}_1 \\ \tilde{H}_1 \end{bmatrix} & \tilde{L}_2 \\ \tilde{H}_2 & \end{bmatrix} & \ddots \\ \ddots & \end{bmatrix} & \tilde{L}_{J-1} \\ \tilde{H}_{J-1} & \end{bmatrix} & \tilde{L}_J \\ \tilde{H}_J & \end{bmatrix}^{-1}. \tag{2.2.6}$$

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

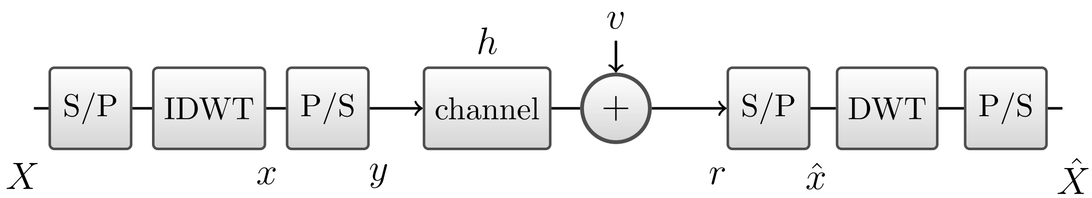


Figure 2.5. DWT-based OFDM system model.

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-branchs 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

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

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

$$r = hx + v, \tag{2.2.9}$$

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

$$r = x + v. \tag{2.2.10}$$

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

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

where

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

and

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

The covariance matrix of the vector v_W can be represented as

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

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 other[3]. 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.

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

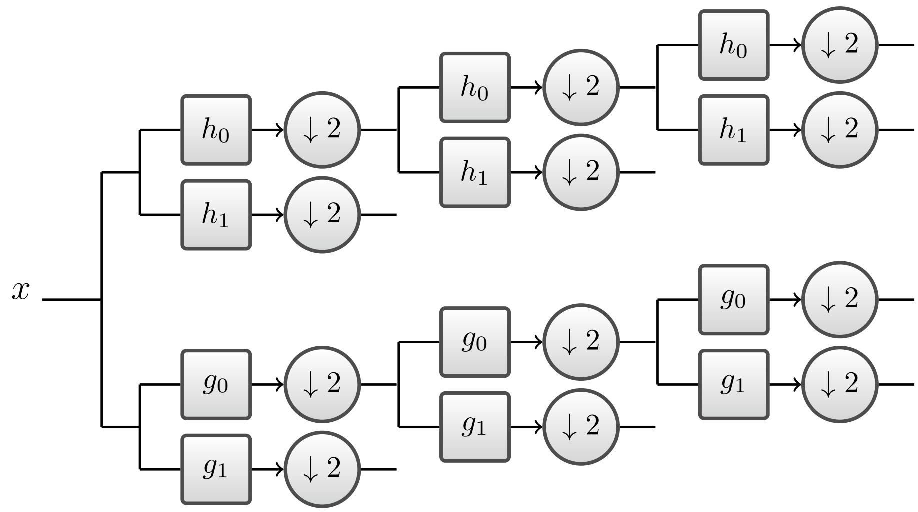


Figure 2.6. Three stage DT-CWT tree structure, analysis.

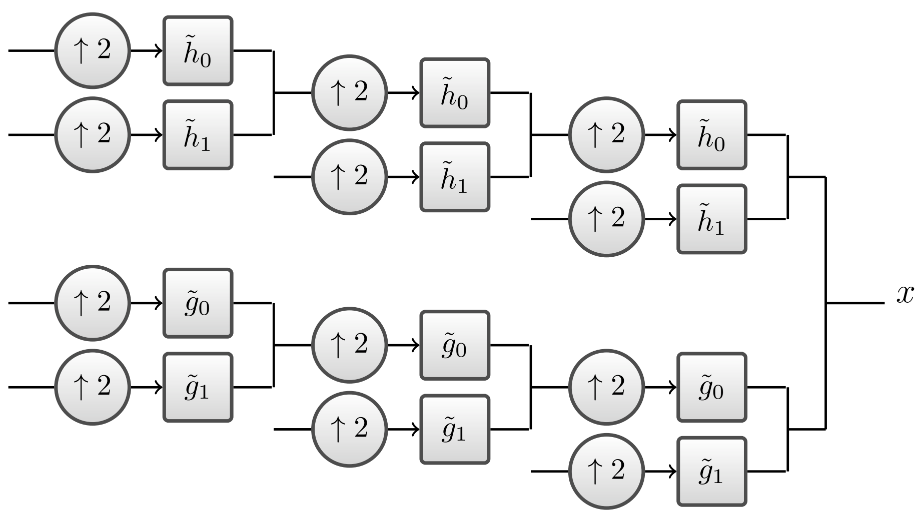


Figure 2.7. Three stage IDT-CWT tree structure, synthesis.

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

$$
Q_{hN} = \begin{bmatrix}
h_0(0) & h_0(1) & h_0(2) & h_0(3) & \ldots & 0 & 0 \\
0 & 0 & h_0(0) & h_0(1) & \ldots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
h_0(L-2) & h_0(L-1) & 0 & 0 & \ldots & h_0(L-4) & h_0(L-3) \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
h_1(0) & h_1(1) & h_1(2) & h_1(3) & \ldots & 0 & 0 \\
0 & 0 & h_1(0) & h_1(1) & \ldots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
h_1(L-2) & h_1(L-1) & 0 & 0 & \ldots & h_1(L-4) & h_1(L-3) \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots
\end{bmatrix} = \begin{bmatrix} L_{hj} \\ H_{hj} \end{bmatrix}, \tag{2.3.1}
$$

where L is the filter length, and L_{hj} and H_{hj} 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

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

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

$$W_R = \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} L_J \\ H_J \end{bmatrix} L_{J-1} \\ H_{J-1} \end{bmatrix} \ddots \\ \ddots \end{bmatrix} L_2 \\ H_2 \end{bmatrix} L_1 \\ H_1 \end{bmatrix}. \tag{2.3.3}$$

The synthesis FB for the upper tree can be written as

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

where L is the filter length, and $\tilde{L}_{hj}$ and $\tilde{H}_{hj}$ 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

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

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

$$\tilde{W}_R^{-1} = \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} \tilde{L}_1 \\ \tilde{H}_1 \end{bmatrix} \tilde{L}_2 \\ \tilde{H}_2 \end{bmatrix} \ddots \\ \ddots \end{bmatrix} \tilde{L}_{J-1} \\ \tilde{H}_{J-1} \end{bmatrix} \tilde{L}_J \\ \tilde{H}_J \end{bmatrix}^{-1}. \tag{2.3.6}$$

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:

$$h_0(n) = h_0(L-1-n) \tag{2.3.7}$$

$$g_0(n) = g_0(L-n). \tag{2.3.8}$$

This procedure gives biorthogonal set of filters.

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

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

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

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

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

$$h_0(n) = f(n) * d(n), \tag{2.3.11}$$

and

$$g_0(n) = f(n) * d(L-n), \tag{2.3.12}$$

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

$$H_0(z) = F(z)D(z), \tag{2.3.13}$$

and

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

Let $A(z)$ be an all-pass transfer function[4] defined as

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

then

$$G_0(z) = H_0(z)A(z), \tag{2.3.16}$$

accordingly

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

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

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

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].

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

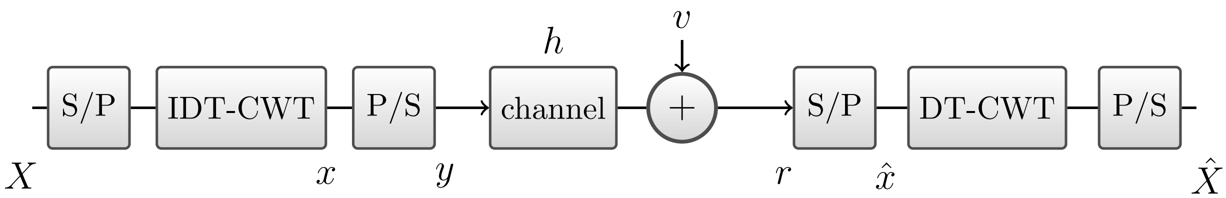


Figure 2.8. DT-CWT-based OFDM System Model.

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

$$x = \begin{bmatrix} x_h \\ x_g \end{bmatrix} = \begin{bmatrix} \tilde{W}_{h_N}^{-1} \\ \tilde{W}_{g_N}^{-1} \end{bmatrix} X, \tag{2.3.19}$$

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, $\tilde{W}_{h_N}^{-1}$ and $\tilde{W}_{g_N}^{-1}$ are given in equation (2.3.6).

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

$$r = hx + v, \tag{2.3.20}$$

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

$$r = x + v. \tag{2.3.21}$$

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

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

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

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

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

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

and

$$W_{g_N} v_g = \begin{bmatrix} \frac{1}{2\sqrt{2}} \sum_{k=1}^{8} v_k \\ \frac{1}{2\sqrt{2}} \sum_{k=1}^{8} v_k \\ \frac{1}{2} \sum_{k=1}^{4} v_k \\ \frac{1}{2} \sum_{k=1}^{4} v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^{2} v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^{2} v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^{2} v_k \\ \frac{1}{\sqrt{2}} \sum_{k=1}^{2} v_k \end{bmatrix}. \tag{2.3.25}$$

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

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

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 forth 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]

$$\begin{aligned} P_{s_k} &= \left[4\left(\frac{\sqrt{M}-1}{\sqrt{M}}\right) Q\left(\sqrt{\frac{3\gamma_{s_k}}{(M-1)}}\right)\right. \\ &- \left. 4\left(\frac{\sqrt{M}-1}{\sqrt{M}}\right)^2 Q^2\left(\sqrt{\frac{3\gamma_{s_k}}{(M-1)}}\right)\right], \end{aligned} \tag{3.1.1}$$

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

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

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.

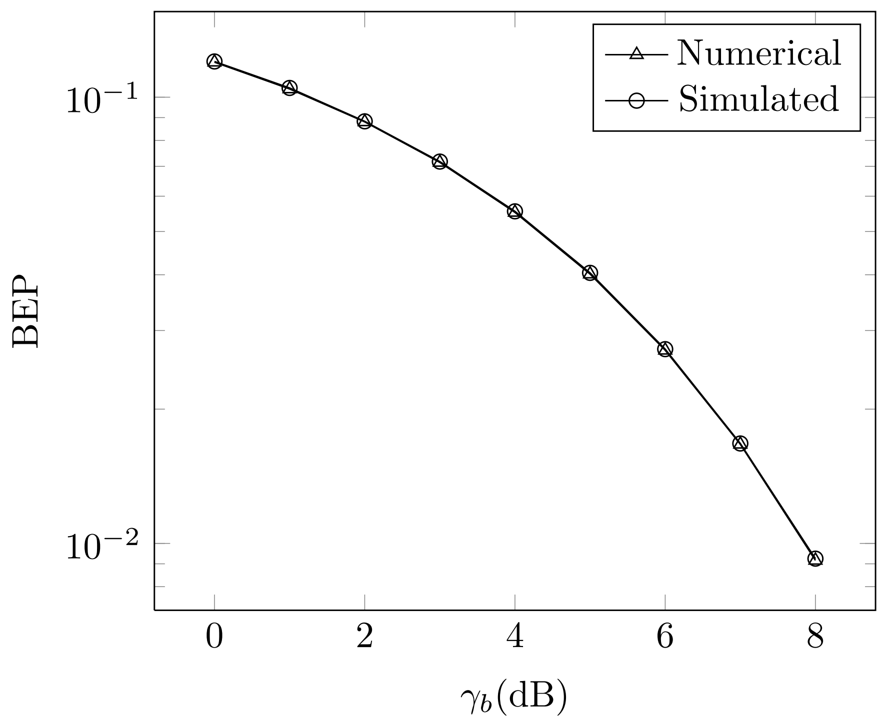


Figure 3.1. BEP, 16-QAM, DFT-based OFDM, Rectangular Pulse Shaping, program in A.1.1.

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]

$$\begin{aligned} P_{s_k} &= \Bigg[4\left(\frac{\sqrt{M}-1}{\sqrt{M}}\right) Q\left(\sqrt{\frac{3\gamma_{s_k}}{(M-1)}}\right) \\ &- 4\left(\frac{\sqrt{M}-1}{\sqrt{M}}\right)^2 Q^2\left(\sqrt{\frac{3\gamma_{s_k}}{(M-1)}}\right)\Bigg], \end{aligned} \tag{3.1.4}$$

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

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

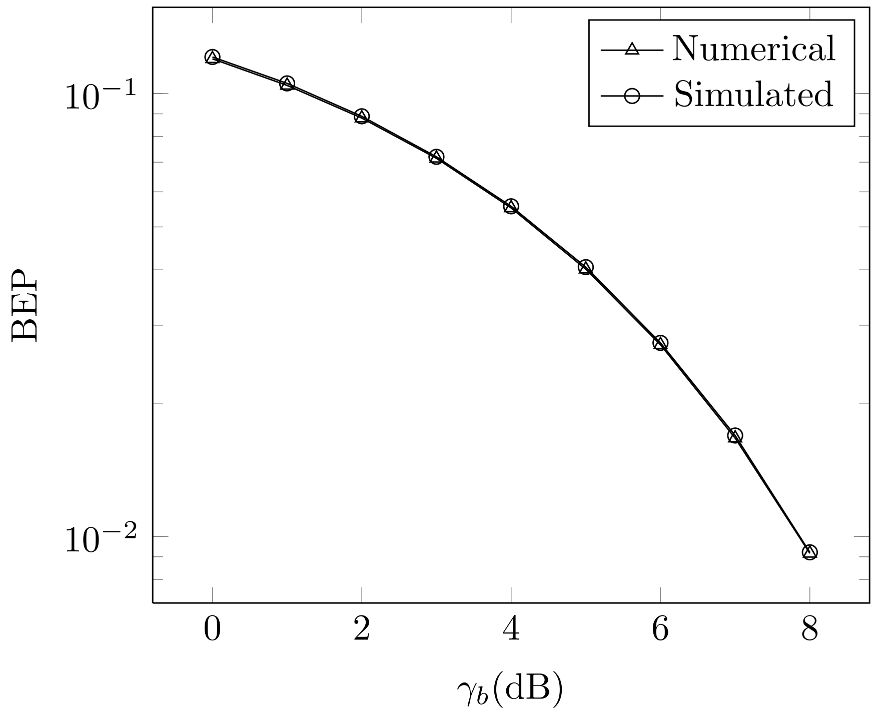


Figure 3.2. BEP, 16-QAM, DWT-based OFDM, with Haar filters, and 5 stages, program in A.1.2.

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.

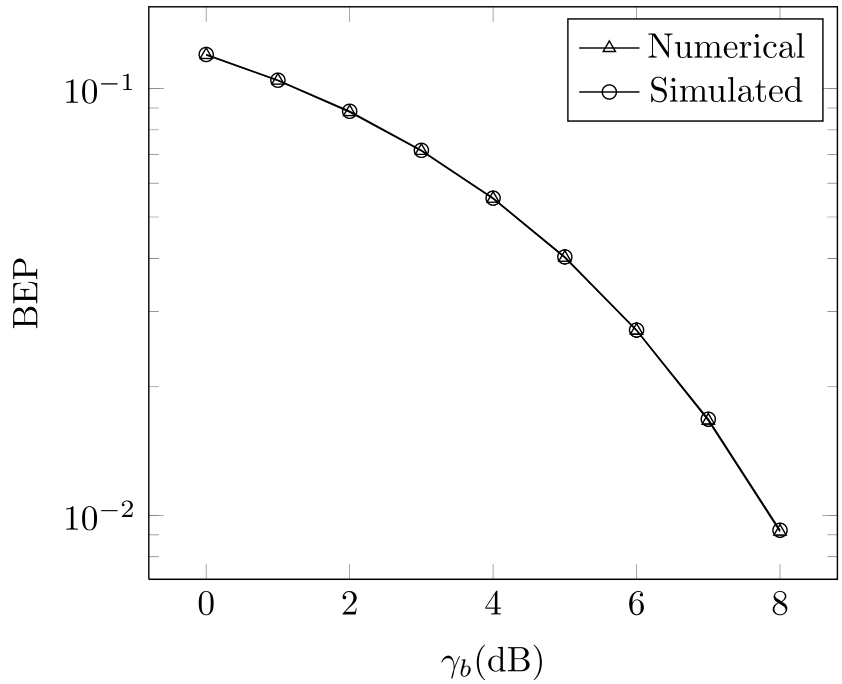


Figure 3.3. BEP, 16-QAM, DWT-based OFDM, with D-6, and 3 stages, program in A.1.3.

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× 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]

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

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

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

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

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

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,

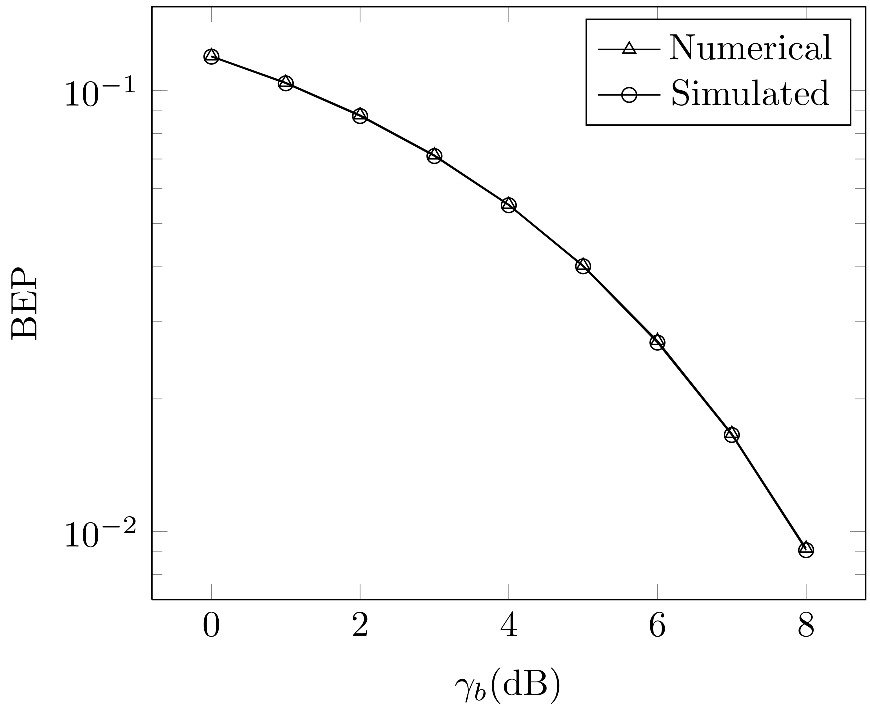


Figure 3.4. BEP, 16-QAM, DT-CWT-based OFDM, with q-shift filters, and 2 stages, program in A.1.4.

both trees employe 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 20^{th} 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.

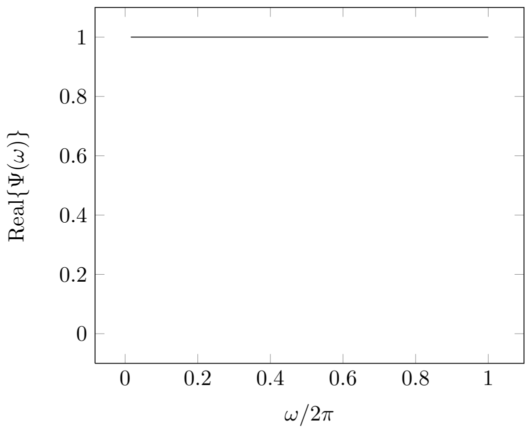
In Figure 3.6, the DWT-based OFDM system response with Haar filters is illustrated. The system's frequency response is illustrated in Figures 3.6(a) and 3.6(b). It is double sided. In Figures 3.6(c) and 3.6(d), the system's impulse response is illustrated. Figure 3.6(e) shows the amplitude of the system's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

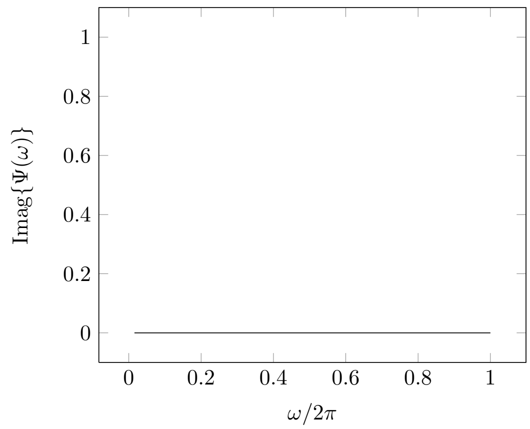
In Figure 3.8, the DT-CWT-based OFDM system frequency response with q-shift filters is illustrated. The system's upper tree frequency response is illustrated in Figures 3.8(a) and Figure 3.8(b). It is double sided. The system's lower tree frequency response is illustrated in Figures 3.8(c) and Figure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

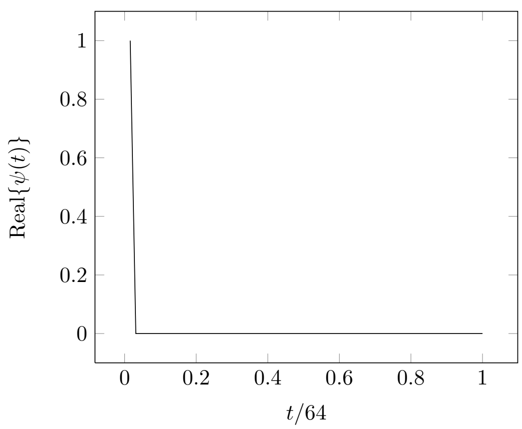
In Figure 3.10, the amplitude of the system's upper tree and lower tree impulse response are illustrated in Figures 3.10(a) and 3.10(b), respectively. Figure 3.10(c) shows the amplitude of the system's dual tree impulse response.



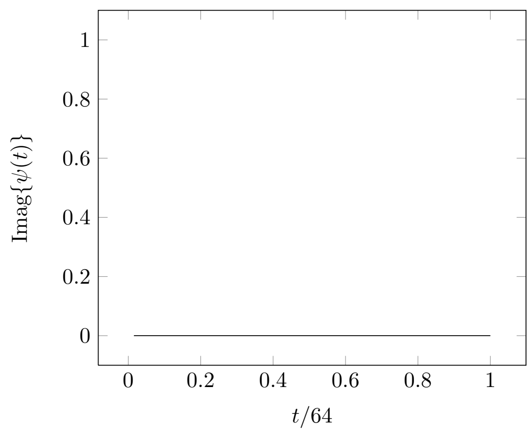
(a) System's frequency response, real



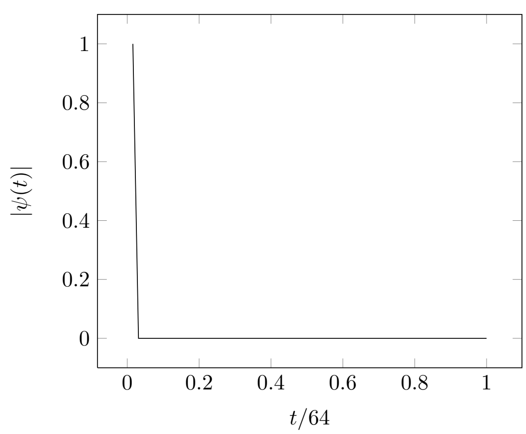
(b) System's frequency response, imaginary



(c) System's impulse response, real

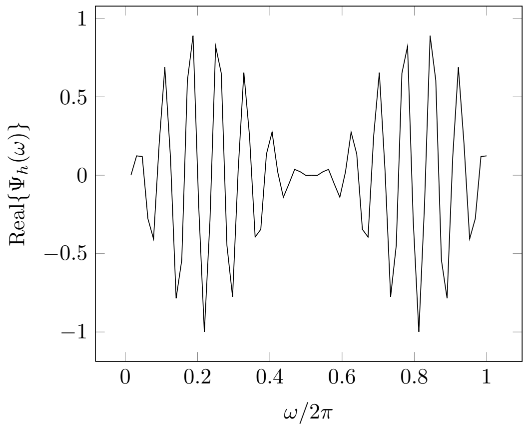


(d) System's impulse response, imaginary

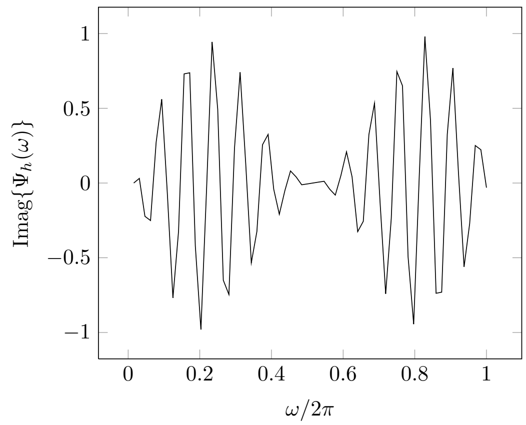


(e) Amplitude of system's impulse response

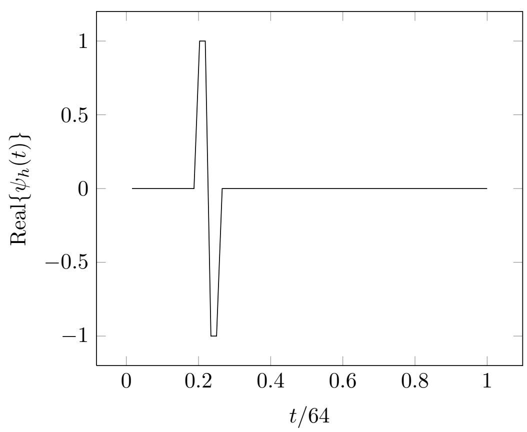
Figure 3.5. Normalized DFT-based OFDM system response, program in A.2.1.



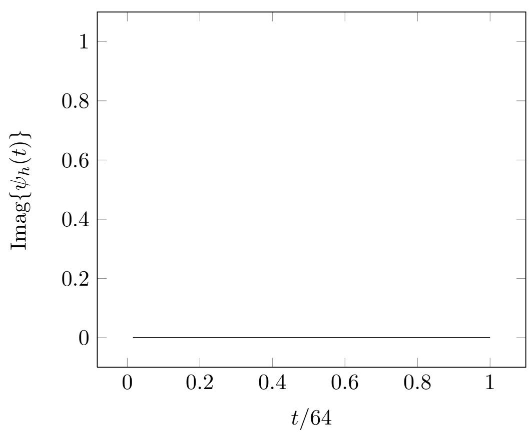
(a) System's frequency response, real



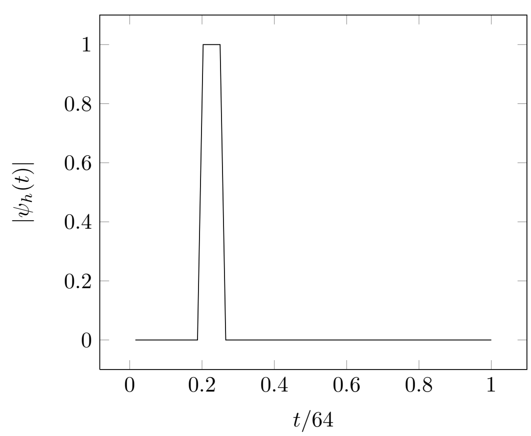
(b) System's frequency response, imaginary



(c) System's impulse response, real

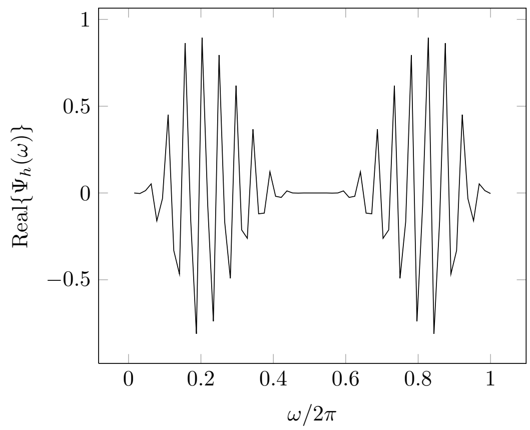


(d) System's impulse response, imaginary

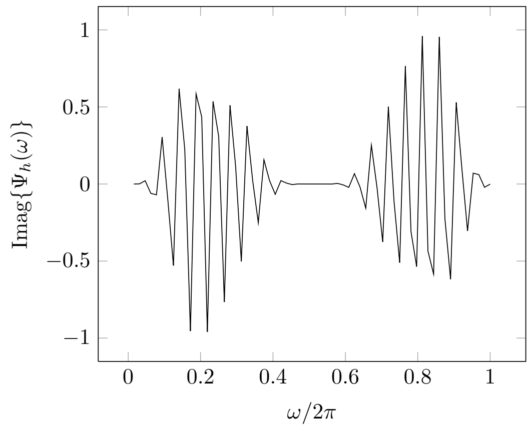


(e) Amplitude of system's impulse response

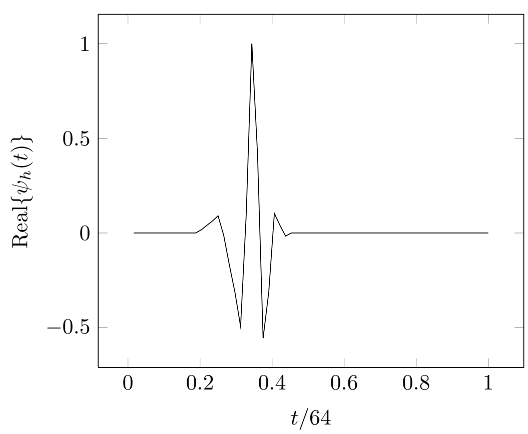
Figure 3.6. Normalized DWT-based OFDM system response, with Haar filters, program in A.2.2.



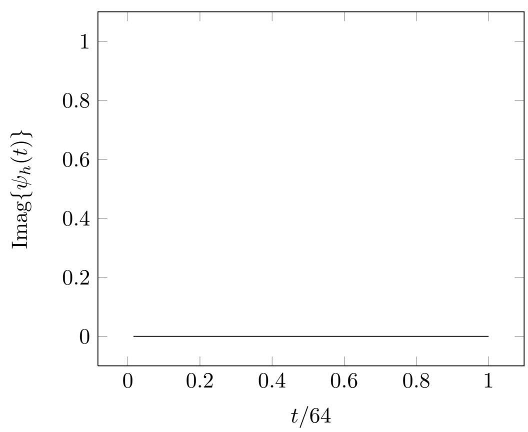
(a) System's frequency response, real



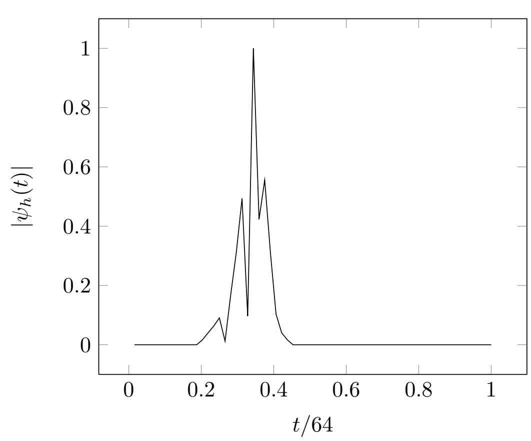
(b) System's frequency response, imaginary



(c) System's impulse response, real

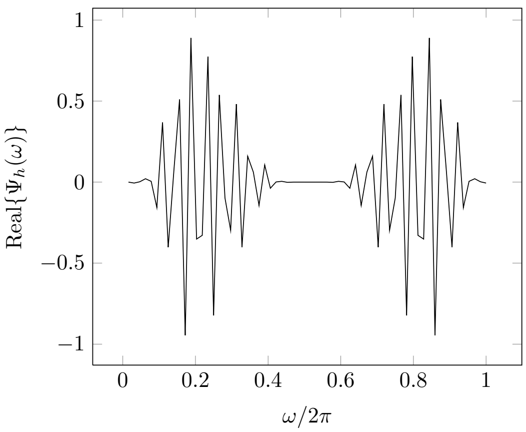


(d) System's impulse response, imaginary

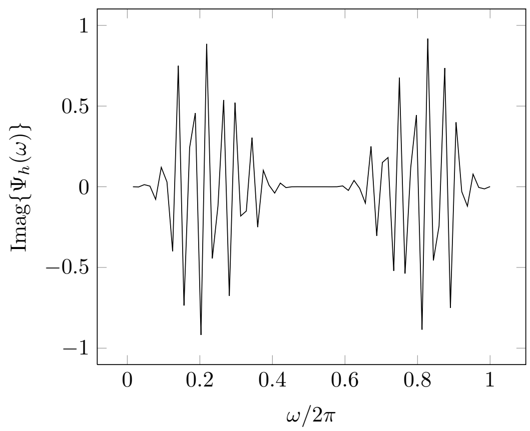


(e) Amplitude of system's impulse response

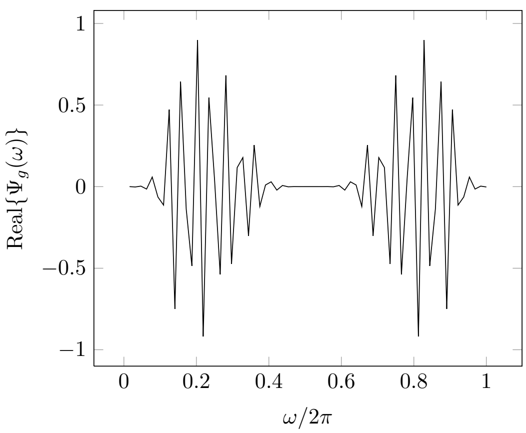
Figure 3.7. Normalized DWT-based OFDM system response, with D-6 filters, program in A.2.3.



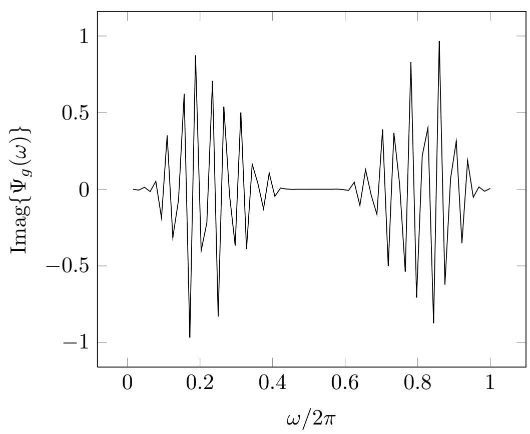
(a) System's upper tree frequency response, real



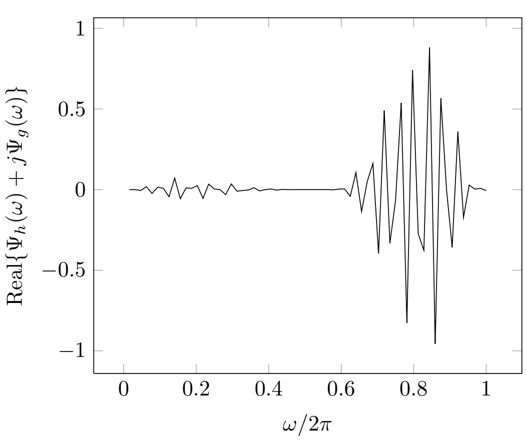
(b) System's upper tree frequency response, imaginary



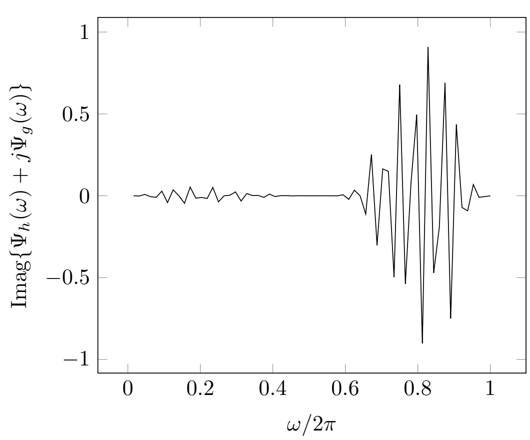
(c) System's lower tree frequency response, real



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

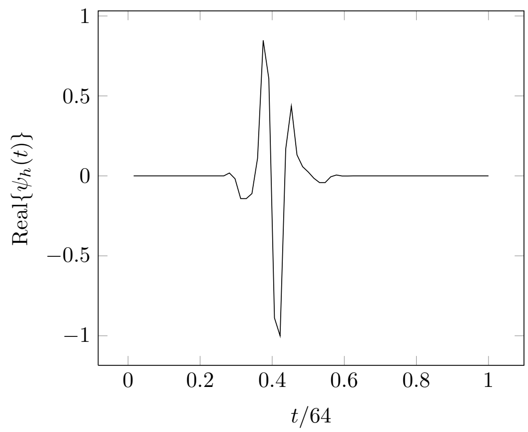


(e) System's dual tree frequency response, real

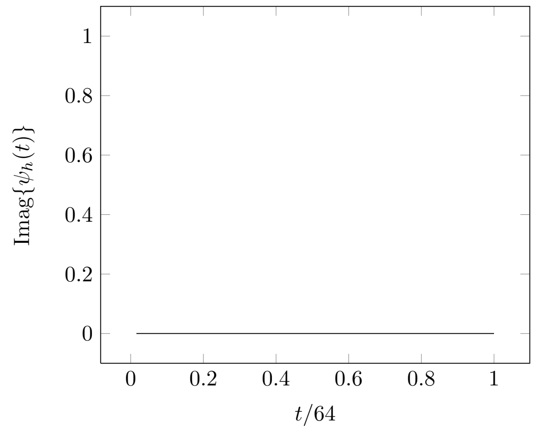


(f) System's dual tree frequency response, imaginary

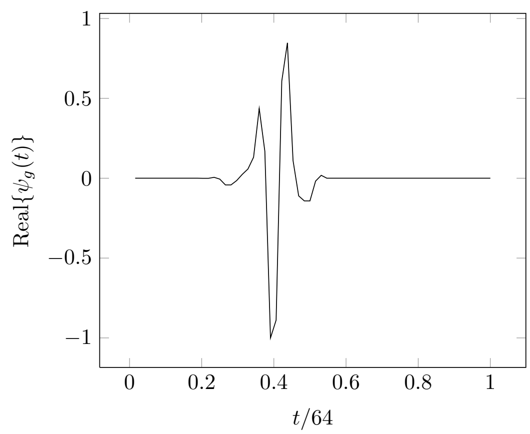
Figure 3.8. Normalized DT-CWT-based OFDM frequency response, with q-shift filters, program in A.2.4.



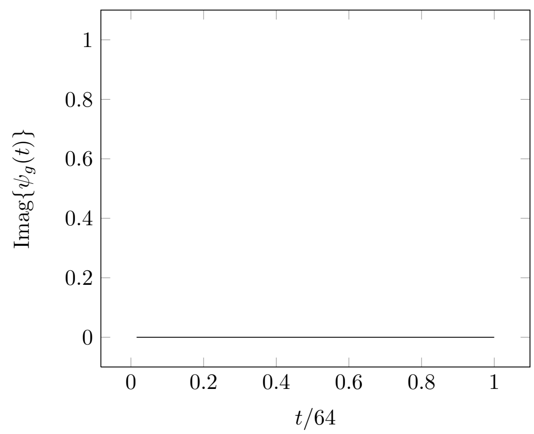
(a) System's upper tree impulse response, real



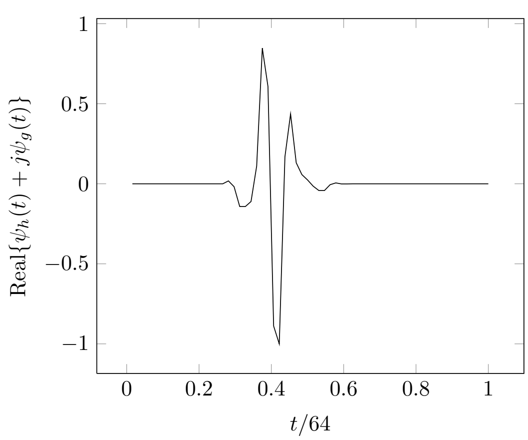
(b) System's upper tree impulse response, imaginary



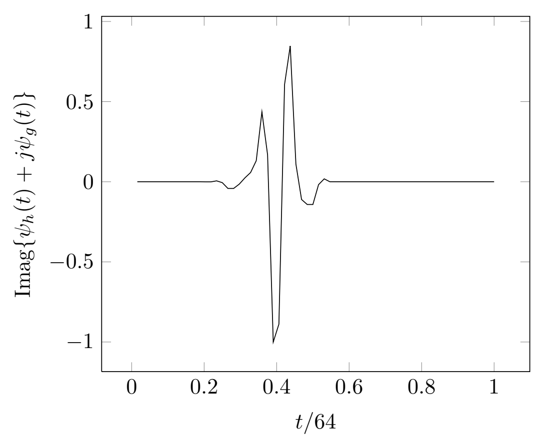
(c) System's lower tree impulse response, real



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

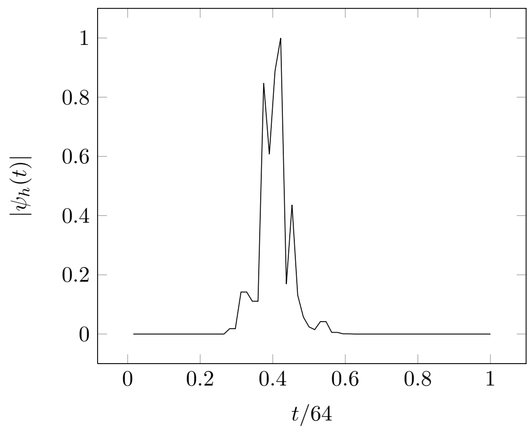


(e) System's dual tree impulse response, real

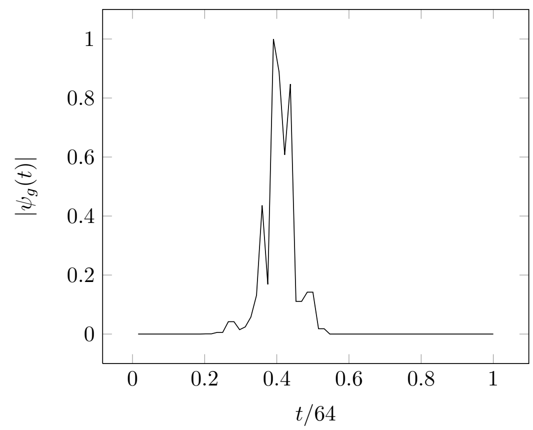


(f) System's dual tree impulse response, imaginary

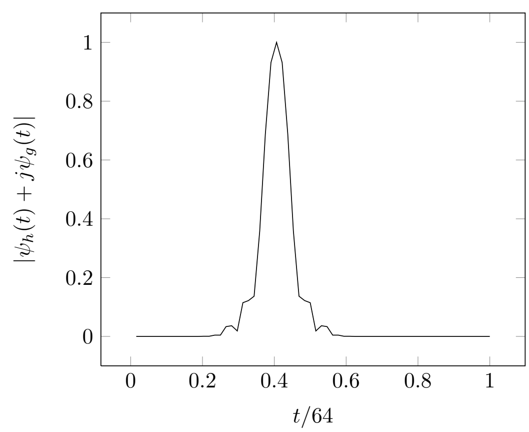
Figure 3.9. Normalized DT-CWT-based OFDM impulse response, with q-shift filters, program in A.2.4.



(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)

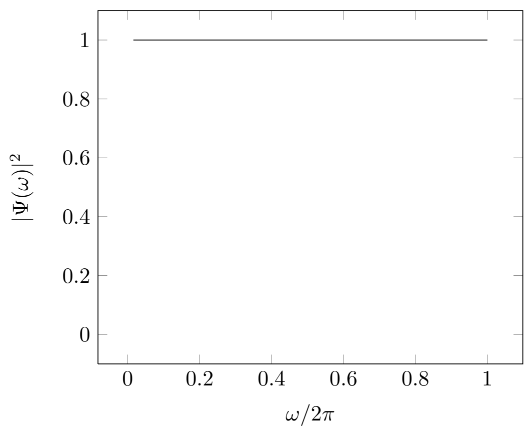
The ESD shows the energies for all frequency components of the signal. It is defined for a signal $f(t)$ as [28]

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

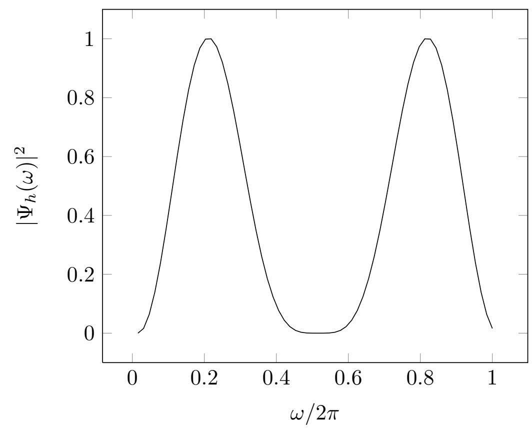
where $|F(\omega)|$ is the absolute value of the Fourier transform of $f(t)$.

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.

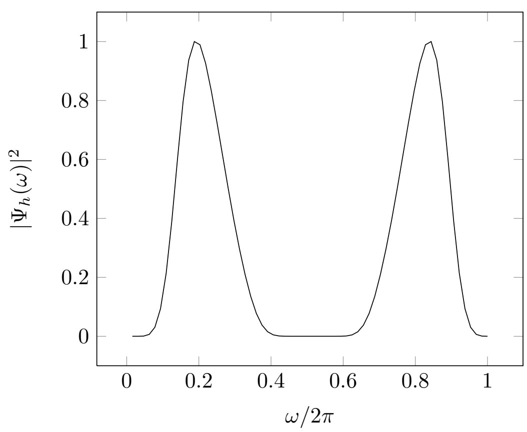
In Figure 3.11(a), the ESD for the DFT-based system is flat. In Figures 3.11(b), 3.11(c), 3.11(d) and 3.11(e), the DWT with Haar filters, the DWT with the D-6 filters, the upper tree of the DT-CWT, and the lower tree of the DT-CWT, respectively, give a double sided ESD. When the previous four ESD Figures 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× redundancy, where one is Hilbert transform of the other.



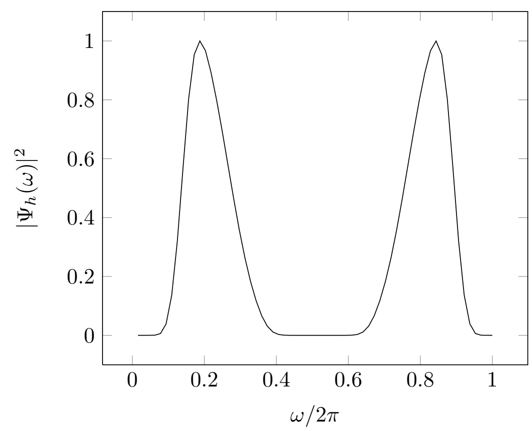
(a) ESD for DFT-based OFDM, program in A.2.1



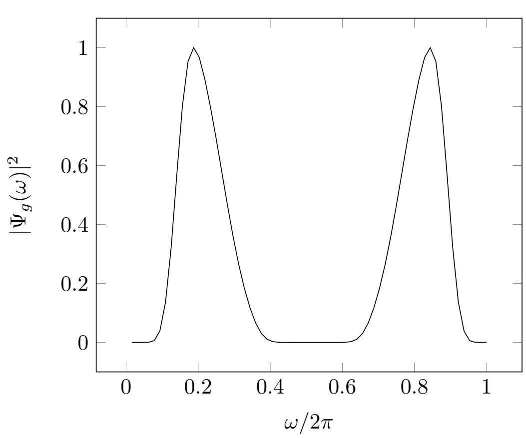
(b) ESD for DWT-based OFDM, Haar filters, program in A.2.2



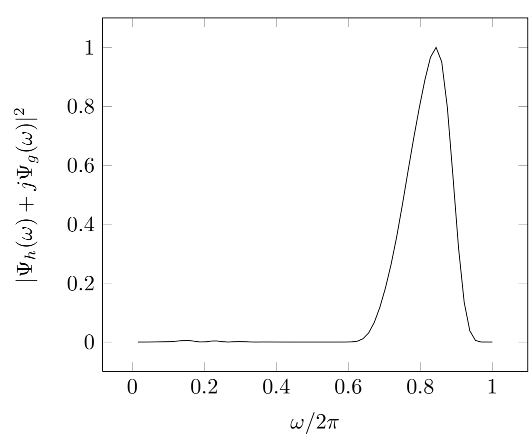
(c) ESD for DWT-based OFDM, D-6 filters, program in A.2.3



(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]

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

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;

$$P_{\mathrm{PAPR}} = P\{\mathrm{PAPR} > \mathrm{PAPR}_0\}, \tag{3.4.2}$$

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.

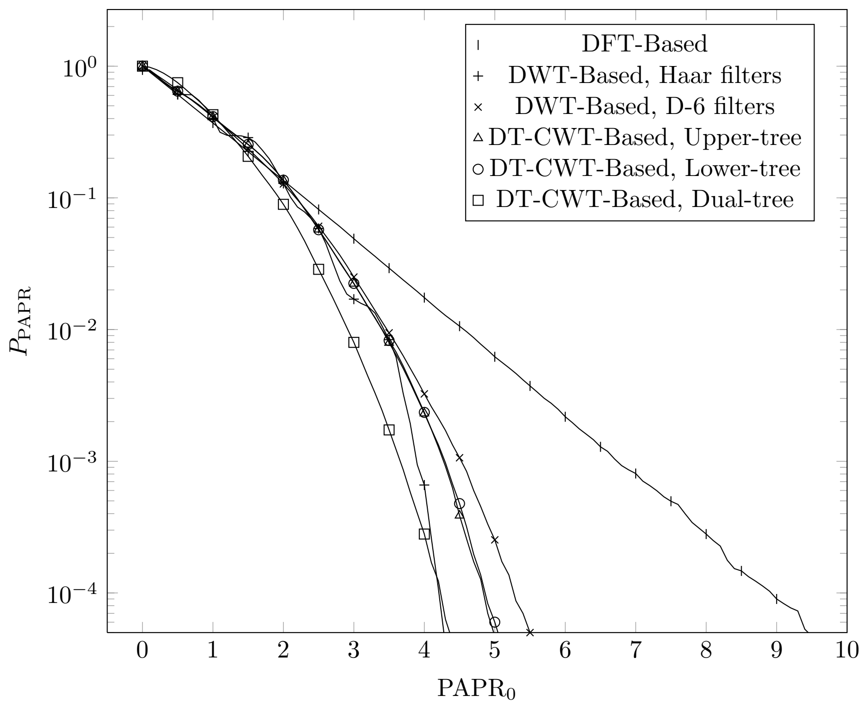


Figure 3.12. $P_{\mathrm{PAPR}} = P\{\mathrm{PAPR} > \mathrm{PAPR}_0\}$ for OFDM alternatives, program in A.3.

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 programing, and Latex programing 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] Himal 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] Athananthan, 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.

```
clc
clear
//—————————————————————————————————————
// Q_FUNCTION
function [q_function] = Q_FUNCTION(some_value),
  q_function = 0.5*erfc(sqrt(0.5)*some_value);
endfunction
//——————————————
// the transmitter side
//——————————————
data_length = 64;
number_channels = 64;
sqrt_no_channels = sqrt(number_channels);
data_before_norm = zeros(data_length,2);
data = zeros(data_length,1);
output = zeros(data_length,1);
output_copy = zeros(data_length,1);
constructed_data = zeros(data_length,1);
reconstructed_data1 = zeros(data_length,2);
reconstructed_data = zeros(data_length,2);
number_of_times_to_avg_on = 3*10^4;
//——————————————————
// data has to be of even length
```

```
data_before_norm = grand(data_length,2,'uin',0,1);
random = rand(data_before_norm,'normal');
for L = 1:data_length,
   if random(L,1) > 0 then
     data_before_norm(L,1) = 2*data_before_norm(L,1)+1;
   else data_before_norm(L,1) = -1*(2*data_before_norm(L,1)+1);
   end
    if random(L,2) > 0 then
     data_before_norm(L,2) = 2*data_before_norm(L,2)+1;
   else data_before_norm(L,2) = -1*(2*data_before_norm(L,2)+1);
   end
end
normalization = sqrt(10);
data = (data_before_norm(1:data_length,1)+%i*data_before_norm(1:data_length,2))/normalization; //normalized
//-----------------------------------
ifft_data = sqrt_no_channels*ifft(data);
output = ifft_data;
//----------------
// the channel
//----------------
counter1 = zeros(1,9);
for signal_to_noise_ratio = 0:8,
 for no_of_times_to_avg_on = 1:number_of_times_to_avg_on,
   output_copy = output;
   some_constants = 1/normalization*sqrt(10/8*10^(-0.1*signal_to_noise_ratio));
   // both real and imag noise have same power as regular noise "one dimentinal noise" because nbar^2=nbar_c^2=nbar_s^2
   real_noise = some_constants*rand(data_length,1,'normal');
   imag_noise = %i*some_constants*rand(data_length,1,'normal');
   output_copy = output_copy + real_noise + imag_noise;
//-----------------------
// the receiver side
//-----------------------
fft_data = normalization/sqrt_no_channels*fft(output_copy);
reconstructed_data1(1:data_length,1) = real(fft_data);
reconstructed_data1(1:data_length,2) = imag(fft_data);
//-----------------------
// error countor:
//-----------------------
sign_reconstructed_data1 = zeros(data_length,2);
abs_reconstructed_data1 = zeros(data_length,2);
```

```
sign_reconstructed_data1 = sign(reconstructed_data1);
abs_reconstructed_data1 = abs(reconstructed_data1);
//----------
for L = 1:data_length,
  if (abs_reconstructed_data1(L,1) < 2 & abs_reconstructed_data1(L,2) < 2) then
    reconstructed_data(L,1) = 1*sign_reconstructed_data1(L,1);
    reconstructed_data(L,2) = 1*sign_reconstructed_data1(L,2);

    elseif (abs_reconstructed_data1(L,1) >= 2 & abs_reconstructed_data1(L,2) < 2) then
    reconstructed_data(L,1) = 3*sign_reconstructed_data1(L,1);
    reconstructed_data(L,2) = 1*sign_reconstructed_data1(L,2);

    elseif (abs_reconstructed_data1(L,1) < 2 & abs_reconstructed_data1(L,2) >= 2) then
    reconstructed_data(L,1) = 1*sign_reconstructed_data1(L,1);
    reconstructed_data(L,2) = 3*sign_reconstructed_data1(L,2);

    elseif (abs_reconstructed_data1(L,1) >= 2 & abs_reconstructed_data1(L,2) >= 2) then
    reconstructed_data(L,1) = 3*sign_reconstructed_data1(L,1);
    reconstructed_data(L,2) = 3*sign_reconstructed_data1(L,2);
  end
    //----------
    if ( (reconstructed_data(L,1) ~= data_before_norm(L,1)) | (reconstructed_data(L,2) ~= data_before_norm(L,2)) ) then
      counter1(1,signal_to_noise_ratio+1) = counter1(1,signal_to_noise_ratio+1) + 1;
    end //if data1
  end // for L
//----------
end // for no_of_times_to_avg_on
end // for signal_to_noise_ratio
bit_error_rate1 = counter1/(4*data_length*number_of_times_to_avg_on);
signal_to_noise_ratio = 0:8
// the theorytical results
Q = Q_FUNCTION(sqrt(8/10*(10.^(signal_to_noise_ratio/10))));
//sigma^2 = 5/(4*SNR_b)
QAM16 = (3*Q - 9/4*Q^2)/4;
// Plot using Tikz:
fd = mopen('\Users\Tassniem\Documents/TikZ/figure_1/figure_1.tex','wt');
mfprintf(fd,'%s \n','\documentclass{article}');
```

```
mfprintf(fd,'%s \n','\usepackage{tikz}');
mfprintf(fd,'%s \n','\usepackage{pgfplots}');
mfprintf(fd,'%s \n','\begin{document}');
mfprintf(fd,'%s \n','\begin{center}');
//-----------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{semilogyaxis}[');
mfprintf(fd,'%s \n','xlabel=SNR(dB),');
mfprintf(fd,'%s \n','ylabel={BEP}]');
mfprintf(fd,'%s \n','\addplot[color=black,mark=triangle]');
mfprintf(fd,'%s \n','coordinates{');
for signal_to_noise_ratio = 0:8,
  mfprintf(fd,'%s %d %s %.6f %s \n','(', signal_to_noise_ratio,',', QAM16
      (1,signal_to_noise_ratio+1),')');
end
mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\addplot[color=black,mark=o]');
mfprintf(fd,'%s \n','coordinates{');
for signal_to_noise_ratio = 0:8,
  mfprintf(fd,'%s %d %s %.6f %s \n','(', signal_to_noise_ratio,',',
      bit_error_rate1(1,signal_to_noise_ratio+1),')');
end
mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\legend{Numerical\\Simulated\\}');
mfprintf(fd,'%s \n','\end{semilogyaxis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//-----------------
mfprintf(fd,'%s \n','\end{center}');
mfprintf(fd,'%s \n','\end{document}');
mclose(fd);
```

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

```
clc
clear
//--------------------------------------------------------------
//CIRCULAR_SHIFT_BY_TWO
function [shifted_sequence] = CIRCULAR_SHIFT_BY_TWO(to_be_shifted_sequence),

  to_be_shifted_sequence_length = length(to_be_shifted_sequence);

  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)];

endfunction
//--------------------------------------------------------------
// MATRIX_EVALUATION
function [matrix_evaluation] = MATRIX_EVALUATION(transformation_length, filters),
  [r, c] = size(filters);
  matrix_evaluation = zeros(transformation_length, transformation_length);

  matrix_evaluation(1,1:c) = filters(1,1:c);
  matrix_evaluation(1+transformation_length/2,1:c) = filters(2,1:c);

  for row = 2:transformation_length/2,
      matrix_evaluation(row,1:c) = CIRCULAR_SHIFT_BY_TWO(matrix_evaluation(row-1,1:c));
      matrix_evaluation(row+transformation_length/2,1:c) = CIRCULAR_SHIFT_BY_TWO( matrix_evaluation(row-1+transformation_length/2,1:c));
    end
endfunction
//--------------------------------------------------------------
// DWT
function [dwt_matrix] = DWT(data_length,filter_length,no_stages,filters),

  dwt_matrix = eye(data_length,data_length);

  if no_stages > 1 then

```

```
    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)]);

    if no_stages > 2 then
      for stage = no_stages-1:-1:2,
        matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters(3:4,:), zeros(2,data_length/2^stage-filter_length)]);
        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);
        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);
      end
    end
  end
  matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,data_length-filter_length)]);
  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);
  dwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(data_length/2+1:data_length,1:data_length);
endfunction
//----------------------------------------------------------------
// IDWT
function [idwt_matrix] = IDWT(data_length,filter_length,no_stages,filters),

  idwt_matrix = eye(data_length,data_length);

    if no_stages > 1 then

      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)]);

      if no_stages > 2 then
        for stage = no_stages-1:-1:2,
          matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters(3:4,:),zeros(2,data_length/2^stage-filter_length)]);
```

```
            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);
            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);
          end
        end
      end

      matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,data_length-filter_length)]);
      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);
      idwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(data_length/2+1:data_length,1:data_length);

  idwt_matrix = idwt_matrix.';
endfunction
//---------------------------------------------------------------------
// Q_FUNCTION
function [q_function] = Q_FUNCTION(some_value),

  q_function = 0.5*erfc(sqrt(0.5)*some_value);

endfunction
//---------------------------------------------------------------------
// AVERAGE
function [average] = AVERAGE(some_vector),

  average = sum(some_vector)/length(some_vector);

endfunction
//---------------------------------------------------------------------
number_of_times_to_avg_on = 3*10^4;
data_length = 64;
number_channels = 64;
sqrt_no_channels = sqrt(number_channels);
data_before_norm = zeros(data_length,2);
SNR = 0:8;
SNR_range = length(SNR);
```

```
//-------------------------------
// Transmitter
//---------------
// Data
  data_before_norm = grand(data_length,2,'uin',0,1);
  random = rand(data_before_norm,'normal');
  for L = 1:data_length,
    if random(L,1) > 0 then
      data_before_norm(L,1) = 2*data_before_norm(L,1)+1;
    else data_before_norm(L,1) = -1*(2*data_before_norm(L,1)+1);
    end
     if random(L,2) > 0 then
      data_before_norm(L,2) = 2*data_before_norm(L,2)+1;
    else data_before_norm(L,2) = -1*(2*data_before_norm(L,2)+1);
    end
  end
  normalization = sqrt(10);
  //-------------------------------
  // DWT-OFDM, Haar filters
  //-------------------------------
  wavelet = dbwavf('db1');// it is Haar
  [low_analysis_filter, high_analysis_filter, low_synthesis_filter,
      high_synthesis_filter] = orthfilt(wavelet);
  analysis_filter = [low_analysis_filter; high_analysis_filter];
  First_analysis_filter = analysis_filter;
  synthesis_filter = [low_synthesis_filter; high_synthesis_filter];
  Final_synthesis_filter = synthesis_filter;
  //----------------------------------------------------------------
  no_stages = 5;
  filter_length = length(analysis_filter(1,:));
  //-------------------------------
  data = (data_before_norm(1:data_length,1) + %i*data_before_norm(1:
      data_length,2))/normalization; //normalized
  //-------------------------------
  // Filters
  low_filter = 1;
  high_filter = 2;

  ANALYSIS_FILTERS = [First_analysis_filter(low_filter:high_filter,:);
      analysis_filter(low_filter:high_filter,:)];

  SYNTHESIS_FILTERS = [Final_synthesis_filter(low_filter:high_filter,:);
      synthesis_filter(low_filter:high_filter,:)];
```

```
the_idwt_matrix_Haar = IDWT(data_length,filter_length,no_stages,flipdim(SYNTHESIS_FILTERS,2));
//--------------------------------
// Transformation
//--------------------------------
output_IDWT_Haar = the_idwt_matrix_Haar*data;
//--------------------
// Channel
//--------------------
counter1 = zeros(1,9);
for signal_to_noise_ratio = 0:8,
  for no_of_times_to_avg_on = 1:number_of_times_to_avg_on,

    copy_output_IDWT_Haar = output_IDWT_Haar;
    some_constants = 1/normalization*sqrt(5/4*10^(-0.1*signal_to_noise_ratio));
    // both real and imag noise have same power as regular noise "one dimentinal noise" because nbar^2=nbar_c^2=nbar_s^2

    real_noise = some_constants*rand(data_length,1,'normal');
    imag_noise = %i*some_constants*rand(data_length,1,'normal');

    copy_output_IDWT_Haar = copy_output_IDWT_Haar + real_noise + imag_noise;

    the_dwt_matrix_Haar = DWT(data_length,filter_length,no_stages,ANALYSIS_FILTERS);

    output_DWT_Haar = the_dwt_matrix_Haar*copy_output_IDWT_Haar*normalization;
    //--------------
    output_DWT_Haar1(1:data_length,1) = real(output_DWT_Haar);
    output_DWT_Haar1(1:data_length,2) = imag(output_DWT_Haar);
    //--------------
    sign_output_DWT_Haar = sign(output_DWT_Haar1);
    abs_output_DWT_Haar = abs(output_DWT_Haar1);
    //--------------
    for L = 1:data_length,

      if (abs_output_DWT_Haar(L,1) < 2 & abs_output_DWT_Haar(L,2) < 2) then
        output_DWT_Haar(L,1) = 1*sign_output_DWT_Haar(L,1);
        output_DWT_Haar(L,2) = 1*sign_output_DWT_Haar(L,2);
```

```

            elseif (abs_output_DWT_Haar(L,1) >= 2 & abs_output_DWT_Haar(L,2) < 2) then
            output_DWT_Haar(L,1) = 3*sign_output_DWT_Haar(L,1);
            output_DWT_Haar(L,2) = 1*sign_output_DWT_Haar(L,2);

            elseif (abs_output_DWT_Haar(L,1) < 2 & abs_output_DWT_Haar(L,2) >= 2) then
            output_DWT_Haar(L,1) = 1*sign_output_DWT_Haar(L,1);
            output_DWT_Haar(L,2) = 3*sign_output_DWT_Haar(L,2);

            elseif (abs_output_DWT_Haar(L,1) >= 2 & abs_output_DWT_Haar(L,2) >= 2) then
            output_DWT_Haar(L,1) = 3*sign_output_DWT_Haar(L,1);
            output_DWT_Haar(L,2) = 3*sign_output_DWT_Haar(L,2);
         end
            //----------
            if ( (output_DWT_Haar(L,1) ~= data_before_norm(L,1)) | ( output_DWT_Haar(L,2) ~= data_before_norm(L,2)) ) then
              counter1(1,signal_to_noise_ratio+1) = counter1(1,signal_to_noise_ratio+1) + 1;
            end //if data1
         end // for L
       //----------
end // for no_of_times_to_avg_on
end // for signal_to_noise_ratio

bit_error_rate1 = counter1/(4*data_length*number_of_times_to_avg_on);
signal_to_noise_ratio = 0:8
// the theorytical results
Q = Q_FUNCTION(sqrt(8/10*(10.^(signal_to_noise_ratio/10))));
//sigma^2 = 5/(4*SNR_b)
QAM16 = (3*Q - 9/4*Q^2)/4;
// Plot using Tikz:
fd = mopen('\Users\Tassniem\Documents/TikZ/figure_2/figure2.tex','wt');
mfprintf(fd,'%s \n','\documentclass{article}');
mfprintf(fd,'%s \n','\usepackage{tikz}');
mfprintf(fd,'%s \n','\usepackage{pgfplots}');
mfprintf(fd,'%s \n','\begin{document}');
//----------
mfprintf(fd,'%s \n','\begin{center}');
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{semilogyaxis}[');
```

```
mfprintf(fd,'%s \n','xlabel=SNR(dB),');
mfprintf(fd,'%s \n','ylabel={BEP}]');
mfprintf(fd,'%s \n','\addplot[color=black,mark=triangle]');
mfprintf(fd,'%s \n','coordinates{');
for signal_to_noise_ratio = 0:8,
  mfprintf(fd,'%s %d %s %.6f %s \n','(', signal_to_noise_ratio,',', QAM16(1,signal_to_noise_ratio+1),')');
end
mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\addplot[color=black,mark=o]');
mfprintf(fd,'%s \n','coordinates{');
for signal_to_noise_ratio = 0:8,
  mfprintf(fd,'%s %d %s %.6f %s \n','(', signal_to_noise_ratio,',', bit_error_rate1(1,signal_to_noise_ratio+1),')');
end
mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\legend{Numerical\\Simulated\\}');
mfprintf(fd,'%s \n','\end{semilogyaxis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
mfprintf(fd,'%s \n','\end{center}');
mfprintf(fd,'%s \n','\end{document}');
mclose(fd);
```

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

```
clc
clear
//-----------------------------------------------------------------
//CIRCULAR_SHIFT_BY_TWO
function [shifted_sequence] = CIRCULAR_SHIFT_BY_TWO(to_be_shifted_sequence),

  to_be_shifted_sequence_length = length(to_be_shifted_sequence);

  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)];

endfunction
//-----------------------------------------------------------------
// MATRIX_EVALUATION
function [matrix_evaluation] = MATRIX_EVALUATION(transformation_length, filters),
  [r, c] = size(filters);
  matrix_evaluation = zeros(transformation_length, transformation_length);

  matrix_evaluation(1,1:c) = filters(1,1:c);
  matrix_evaluation(1+transformation_length/2,1:c) = filters(2,1:c);

  for row = 2:transformation_length/2,
      matrix_evaluation(row,1:c) = CIRCULAR_SHIFT_BY_TWO(matrix_evaluation(row-1,1:c));
      matrix_evaluation(row+transformation_length/2,1:c) = CIRCULAR_SHIFT_BY_TWO( matrix_evaluation(row-1+transformation_length/2,1:c));
    end
endfunction
//-----------------------------------------------------------------
// DWT
function [dwt_matrix] = DWT(data_length,filter_length,no_stages,filters),

  dwt_matrix = eye(data_length,data_length);

  if no_stages > 1 then

```

```
      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)]);

      if no_stages > 2 then
        for stage = no_stages-1:-1:2,
          matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters(3:4,:), zeros(2,data_length/2^stage-filter_length)]);
          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);
          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);
        end
      end
    end
    matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,data_length-filter_length)]);
    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);
    dwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(data_length/2+1:data_length,1:data_length);
endfunction
//------------------------------------------------------------
// IDWT
function [idwt_matrix] = IDWT(data_length,filter_length,no_stages,filters),

    idwt_matrix = eye(data_length,data_length);

      if no_stages > 1 then
        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)]);

        if no_stages > 2 then
          for stage = no_stages-1:-1:2,

            matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters(3:4,:),zeros(2,data_length/2^stage-filter_length)]);
```

```
                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);
                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);
            end
        end
    end
    matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,data_length-filter_length)]);
    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);
    idwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(data_length/2+1:data_length,1:data_length);

  idwt_matrix = idwt_matrix.';
endfunction
//------------------------------------------------------------------
// Q_FUNCTION
function [q_function] = Q_FUNCTION(some_value),

  q_function = 0.5*erfc(sqrt(0.5)*some_value);

endfunction
//------------------------------------------------------------------
// AVERAGE
function [average] = AVERAGE(some_vector),

  average = sum(some_vector)/length(some_vector);

endfunction
//------------------------------------------------------------------
number_of_times_to_avg_on = 3*10^4;
data_length = 64;
number_channels = 64;
sqrt_no_channels = sqrt(number_channels);
data_before_norm = zeros(data_length,2);
SNR = 0:8;
SNR_range = length(SNR);
//------------------------------
```

```
// Transmitter
//—————————
  // Data
  data_before_norm = grand(data_length,2,'uin',0,1);
  random = rand(data_before_norm,'normal');
  for L = 1:data_length,
    if random(L,1) > 0 then
      data_before_norm(L,1) = 2*data_before_norm(L,1)+1;
    else data_before_norm(L,1) = -1*(2*data_before_norm(L,1)+1);
    end
    if random(L,2) > 0 then
      data_before_norm(L,2) = 2*data_before_norm(L,2)+1;
    else data_before_norm(L,2) = -1*(2*data_before_norm(L,2)+1);
    end
  end
  normalization = sqrt(10);
  //——————————————————
  // DWT-OFDM, D-6 filters
  //——————————————————
  wavelet = dbwavf('db3');// it is D-6
  [low_analysis_filter, high_analysis_filter, low_synthesis_filter,
      high_synthesis_filter] = orthfilt(wavelet);
  analysis_filter = [low_analysis_filter; high_analysis_filter];
  First_analysis_filter = analysis_filter;
  synthesis_filter = [low_synthesis_filter; high_synthesis_filter];
  Final_synthesis_filter = synthesis_filter;
  //——————————————————————————————————————————
  no_stages = 3;
  filter_length = length(analysis_filter(1,:));
  //——————————————————
  data = (data_before_norm(1:data_length,1) + %i*data_before_norm(1:
      data_length,2))/normalization; //normalized
  //——————————————————
  // Filters
  low_filter = 1;
  high_filter = 2;

  ANALYSIS_FILTERS = [First_analysis_filter(low_filter:high_filter,:);
      analysis_filter(low_filter:high_filter,:)];

  SYNTHESIS_FILTERS = [Final_synthesis_filter(low_filter:high_filter,:);
      synthesis_filter(low_filter:high_filter,:)];
```

```
the_idwt_matrix_D6 = IDWT(data_length,filter_length,no_stages,flipdim(SYNTHESIS_FILTERS,2));
//----------------------
// Transformation
//----------------------
output_IDWT_D6 = the_idwt_matrix_D6*data;
//----------------
// Channel
//----------------
counter1 = zeros(1,9);

for signal_to_noise_ratio = 0:8,
  for no_of_times_to_avg_on = 1:number_of_times_to_avg_on,

    copy_output_IDWT_D6 = output_IDWT_D6;
    some_constants = 1/normalization*sqrt(5/4*10^(-0.1*signal_to_noise_ratio));
    // both real and imag noise have same power as regular noise "one dimentinal noise" because nbar^2=nbar_c^2=nbar_s^2
    real_noise = some_constants*rand(data_length,1,'normal');
    imag_noise = %i*some_constants*rand(data_length,1,'normal');

    copy_output_IDWT_D6 = copy_output_IDWT_D6 + real_noise + imag_noise;

    the_dwt_matrix_D6 = DWT(data_length,filter_length,no_stages,ANALYSIS_FILTERS);

    output_DWT_D6 = the_dwt_matrix_D6*copy_output_IDWT_D6*normalization;
    //----------
    output_DWT_D61(1:data_length,1) = real(output_DWT_D6);
    output_DWT_D61(1:data_length,2) = imag(output_DWT_D6);
    //----------
    sign_output_DWT_D6 = sign(output_DWT_D61);
    abs_output_DWT_D6 = abs(output_DWT_D61);
    //----------
    for L = 1:data_length,

      if (abs_output_DWT_D6(L,1) < 2 & abs_output_DWT_D6(L,2) < 2) then
        output_DWT_D6(L,1) = 1*sign_output_DWT_D6(L,1);
        output_DWT_D6(L,2) = 1*sign_output_DWT_D6(L,2);

```

```
            elseif (abs_output_DWT_D6(L,1) >= 2 & abs_output_DWT_D6(L,2) < 2) then
            output_DWT_D6(L,1) = 3*sign_output_DWT_D6(L,1);
            output_DWT_D6(L,2) = 1*sign_output_DWT_D6(L,2);

            elseif (abs_output_DWT_D6(L,1) < 2 & abs_output_DWT_D6(L,2) >= 2) then
            output_DWT_D6(L,1) = 1*sign_output_DWT_D6(L,1);
            output_DWT_D6(L,2) = 3*sign_output_DWT_D6(L,2);

            elseif (abs_output_DWT_D6(L,1) >= 2 & abs_output_DWT_D6(L,2) >= 2) then
            output_DWT_D6(L,1) = 3*sign_output_DWT_D6(L,1);
            output_DWT_D6(L,2) = 3*sign_output_DWT_D6(L,2);
          end
            //----------
            if ( (output_DWT_D6(L,1) ~= data_before_norm(L,1)) | ( output_DWT_D6(L,2) ~= data_before_norm(L,2)) ) then
              counter1(1,signal_to_noise_ratio+1) = counter1(1, signal_to_noise_ratio+1) + 1;
            end //if data1
          end // for L
        //----------
end // for no_of_times_to_avg_on
end // for signal_to_noise_ratio

bit_error_rate1 = counter1/(4*data_length*number_of_times_to_avg_on);
signal_to_noise_ratio = 0:8
// the theorytical results
Q = Q_FUNCTION(sqrt(8/10*(10.^(signal_to_noise_ratio/10))));
//sigma^2 = 5/(4*SNR_b)
QAM16 = (3*Q - 9/4*Q^2)/4;
// Plot using Tikz:
fd = mopen('\Users\Tassniem\Documents/TikZ/figure_3/figure3.tex','wt');
mfprintf(fd,'%s \n','\documentclass{article}');
mfprintf(fd,'%s \n','\usepackage{tikz}');
mfprintf(fd,'%s \n','\usepackage{pgfplots}');
mfprintf(fd,'%s \n','\begin{document}');
//----------
mfprintf(fd,'%s \n','\begin{center}');
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{semilogyaxis}[');
mfprintf(fd,'%s \n','xlabel=SNR(dB),');
```

```
mfprintf(fd,'%s \n','ylabel={BEP}]');
mfprintf(fd,'%s \n','\addplot[color=black,mark=triangle]');
mfprintf(fd,'%s \n','coordinates{');
for signal_to_noise_ratio = 0:8,
  mfprintf(fd,'%s %d %s %.6f %s \n','(', signal_to_noise_ratio,',', QAM16
      (1,signal_to_noise_ratio+1),')');
end
mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\addplot[color=black,mark=o]');
mfprintf(fd,'%s \n','coordinates{');
for signal_to_noise_ratio = 0:8,
  mfprintf(fd,'%s %d %s %.6f %s \n','(', signal_to_noise_ratio,',',
      bit_error_rate1(1,signal_to_noise_ratio+1),')');
end
mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\legend{Numerical\\Simulated\\}');
mfprintf(fd,'%s \n','\end{semilogyaxis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
mfprintf(fd,'%s \n','\end{center}');
mfprintf(fd,'%s \n','\end{document}');
mclose(fd);
```

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

```
clc
clear
//-----------------------------------------------------------------
//CIRCULAR_SHIFT_BY_TWO
function [shifted_sequence] = CIRCULAR_SHIFT_BY_TWO(to_be_shifted_sequence),

  to_be_shifted_sequence_length = length(to_be_shifted_sequence);

  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)];

endfunction
//-----------------------------------------------------------------
// MATRIX_EVALUATION
function [matrix_evaluation] = MATRIX_EVALUATION(transformation_length, filters),
  [r, c] = size(filters);
  matrix_evaluation = zeros(transformation_length, transformation_length);

  matrix_evaluation(1,1:c) = filters(1,1:c);
  matrix_evaluation(1+transformation_length/2,1:c) = filters(2,1:c);

  for row = 2:transformation_length/2,
      matrix_evaluation(row,1:c) = CIRCULAR_SHIFT_BY_TWO(matrix_evaluation(row-1,1:c));
      matrix_evaluation(row+transformation_length/2,1:c) = CIRCULAR_SHIFT_BY_TWO( matrix_evaluation(row-1+transformation_length/2,1:c));
    end
endfunction
//-----------------------------------------------------------------
// DWT
function [dwt_matrix] = DWT(data_length,filter_length,no_stages,filters),

  dwt_matrix = eye(data_length,data_length);

  if no_stages > 1 then
```

```
    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)]);

    if no_stages > 2 then
      for stage = no_stages-1:-1:2,
        matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters(3:4,:), zeros(2,data_length/2^stage-filter_length)]);
        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);
        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);
      end
    end
  end
  matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,data_length-filter_length)]);
  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);
  dwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(data_length/2+1:data_length,1:data_length);

endfunction
//------------------------------------------------------------
// IDWT
function [idwt_matrix] = IDWT(data_length,filter_length,no_stages,filters),

  idwt_matrix = eye(data_length,data_length);

    if no_stages > 1 then

      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)]);

      if no_stages > 2 then
        for stage = no_stages-1:-1:2,
          matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters(3:4,:),zeros(2,data_length/2^stage-filter_length)]);
```

```
                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);
                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);
            end
        end
    end
    matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,data_length-filter_length)]);
    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);
    idwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(data_length/2+1:data_length,1:data_length);

  idwt_matrix = idwt_matrix.';
endfunction
//----------------------------------------------------------------
// Q_FUNCTION
function [q_function] = Q_FUNCTION(some_value),

  q_function = 0.5*erfc(sqrt(0.5)*some_value);

endfunction
//----------------------------------------------------------------
// AVERAGE
function [average] = AVERAGE(some_vector),

  average = sum(some_vector)/length(some_vector);

endfunction
//----------------------------------------------------------------
number_of_times_to_avg_on = 3*10^4;
data_length = 64;
number_channels = 64;
sqrt_no_channels = sqrt(number_channels);
data_before_norm = zeros(data_length,2);
SNR = 0:8;
SNR_range = length(SNR);
//--------------------------
```

```
// Transmitter
//----------------
  // Data
  data_before_norm = grand(data_length,2,'uin',0,1);
  random = rand(data_before_norm,'normal');
  for L = 1:data_length,
    if random(L,1) > 0 then
      data_before_norm(L,1) = 2*data_before_norm(L,1)+1;
    else data_before_norm(L,1) = -1*(2*data_before_norm(L,1)+1);
    end
     if random(L,2) > 0 then
      data_before_norm(L,2) = 2*data_before_norm(L,2)+1;
    else data_before_norm(L,2) = -1*(2*data_before_norm(L,2)+1);
    end
  end
  normalization = sqrt(10);
  //------------------------------
  // DT-CWT-OFDM, q-shift filters
  //--------------------------------------------------------------------
  [First_analysis_filter, Final_synthesis_filter] = FSfarras('f');
  [analysis_filter, synthesis_filter] = dualfilt1('f');
  //--------------------------------------------------------------------
  no_stages = 2 ;
  filter_length = length(analysis_filter(1,:));
  //------------------------------
  data = (data_before_norm(1:data_length,1) + %i*data_before_norm(1:data_length,2))/normalization; //normalized
  //------------------------------
  // Filters
  //-------------------------
  for tree_no = 1:2,
      if tree_no == 1 then
        low_filter = 1;
        high_filter = 2;
      elseif tree_no == 2 then
        low_filter = 3;
        high_filter = 4;
      end // for elseif
      //----------------------

  SYNTHESIS_FILTERS = [Final_synthesis_filter(low_filter:high_filter,:); synthesis_filter(low_filter:high_filter,:)];

```

```
    the_idwt_matrix = IDWT(data_length , filter_length , no_stages , flipdim(
        SYNTHESIS_FILTERS,2));

    //--------------------
    // Transformation
    //--------------------
    output_IDWT = the_idwt_matrix*data;
    //--------------------------
    // to store output of both trees
    if tree_no == 1 then
      output_IDWT_upper = output_IDWT;
    elseif tree_no == 2 then
      output_IDWT_lower = output_IDWT;
    end // if tree_no
end // for tree_no
    //-------------
    // Channel
    //-------------
    counter = zeros(1,9);

    for signal_to_noise_ratio = 0:8,
      for no_of_times_to_avg_on = 1:number_of_times_to_avg_on ,

        for tree_no = 1:2,
          if tree_no == 1 then
            low_filter = 1;
            high_filter = 2;
            copy_output_IDWT = output_IDWT_upper;
          elseif tree_no == 2 then
            low_filter = 3;
            high_filter = 4;
            copy_output_IDWT = output_IDWT_lower;
          end // for elseif

          ANALYSIS_FILTERS = [First_analysis_filter(low_filter:high_filter
              ,:);analysis_filter(low_filter:high_filter ,:)];


          some_constants = sqrt(1/4*10^(-0.1*signal_to_noise_ratio));
          // both real and imag noise have same power as regular noise "one
              dimentinal noise" because nbar^2=nbar_c^2=nbar_s^2
          real_noise = some_constants*rand(data_length ,1,'normal');
          imag_noise = %i*some_constants*rand(data_length ,1,'normal');
```

```

      copy_output_IDWT = copy_output_IDWT + real_noise + imag_noise;

      the_dwt_matrix = DWT(data_length,filter_length,no_stages, ANALYSIS_FILTERS);

      output_DWT = the_dwt_matrix*copy_output_IDWT;

      if tree_no == 1 then
        output_DWT_upper = output_DWT;
      elseif tree_no == 2 then
        output_DWT_lower = output_DWT;
      end // if tree_no
    end // for tree_no
    reconstructed_data_both = zeros(data_length,1);
    reconstructed_data_both = .5*(output_DWT_upper + output_DWT_lower)* normalization;
    //----------
    output_DWT_both = zeros(data_length,2);
    output_DWT_both(1:data_length,1) = real(reconstructed_data_both);
    output_DWT_both(1:data_length,2) = imag(reconstructed_data_both);
    //----------
    sign_output_DWT = zeros(data_length,2);
    abs_output_DWT = zeros(data_length,2);

    sign_output_DWT = sign(output_DWT_both);
    abs_output_DWT = abs(output_DWT_both);
    //----------
    for L = 1:data_length,

      if (abs_output_DWT(L,1) < 2 & abs_output_DWT(L,2) < 2) then
        output_DWT_both(L,1) = 1*sign_output_DWT(L,1);
        output_DWT_both(L,2) = 1*sign_output_DWT(L,2);

        elseif (abs_output_DWT(L,1) >= 2 & abs_output_DWT(L,2) < 2) then
        output_DWT_both(L,1) = 3*sign_output_DWT(L,1);
        output_DWT_both(L,2) = 1*sign_output_DWT(L,2);

        elseif (abs_output_DWT(L,1) < 2 & abs_output_DWT(L,2) >= 2) then
        output_DWT_both(L,1) = 1*sign_output_DWT(L,1);
        output_DWT_both(L,2) = 3*sign_output_DWT(L,2);
```

```
            elseif (abs_output_DWT(L,1) >= 2 & abs_output_DWT(L,2) >= 2) then
              output_DWT_both(L,1) = 3*sign_output_DWT(L,1);
              output_DWT_both(L,2) = 3*sign_output_DWT(L,2);
          end
          //---------
            if ( (output_DWT_both(L,1) ~= data_before_norm(L,1)) | ( output_DWT_both(L,2) ~= data_before_norm(L,2)) ) then
              counter(1,signal_to_noise_ratio+1) = counter(1, signal_to_noise_ratio+1) + 1;
            end //if
          end // for L
        //---------
end // for no_of_times_to_avg_on
end // for signal_to_noise_ratio

bit_error_rate = counter/(4*data_length*number_of_times_to_avg_on);

signal_to_noise_ratio = 0:8

// the theorytical results
Q = Q_FUNCTION(sqrt(8/10*(10.^(signal_to_noise_ratio/10))));
//sigma^2 = 5/(4*SNR_b)
QAM16 = (3*Q - 9/4*Q^2)/4;
// Plot using Tikz:
fd = mopen('\Users\Tassniem\Documents/TikZ/figure_4/figure4.tex','wt');
mfprintf(fd,'%s \n','\documentclass{article}');
mfprintf(fd,'%s \n','\usepackage{tikz}');
mfprintf(fd,'%s \n','\usepackage{pgfplots}');
mfprintf(fd,'%s \n','\begin{document}');
//---------
mfprintf(fd,'%s \n','\begin{center}');
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{semilogyaxis}[');
mfprintf(fd,'%s \n','xlabel=SNR(dB),');
mfprintf(fd,'%s \n','ylabel={BEP}]');
mfprintf(fd,'%s \n','\addplot[color=black,mark=triangle]');
mfprintf(fd,'%s \n','coordinates{');
for signal_to_noise_ratio = 0:8,
  mfprintf(fd,'%s %d %s %.6f %s \n','(', signal_to_noise_ratio ,',', QAM16(1,signal_to_noise_ratio+1),')');
end
```

```
mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\addplot[color=black,mark=o]');
mfprintf(fd,'%s \n','coordinates{');
for signal_to_noise_ratio = 0:8,
  mfprintf(fd,'%s %d %s %.6f %s \n','(', signal_to_noise_ratio,',', bit_error_rate(1,signal_to_noise_ratio+1),')');
end
mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\legend{Numerical\\Simulated\\}');
mfprintf(fd,'%s \n','\end{semilogyaxis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
mfprintf(fd,'%s \n','\end{center}');
mfprintf(fd,'%s \n','\end{document}');
mclose(fd);
```

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

A.2.1 FFT-based OFDM.

```
clc
clear

//------------------------------------------------------------------------
// Q_FUNCTION
function [q_function] = Q_FUNCTION(some_value),

  q_function = 0.5*erfc(sqrt(0.5)*some_value);

endfunction

//------------------------------------------------------------------------

//------------------------------------------------------------------------
//-------------------------
// the transmitter side
//-------------------------
//------------------------------------------------------------------------
// data_length := data is in symbols
data_length = 64;
number_channels = 64;
det_fft = number_channels;

data_in_freq = ones(data_length,1);

//------------------------------
// Data
//------------------------------

  //------------------------------
  // DFT-OFDM
  //------------------------------

  output_DFT = ifft(data_in_freq);//in time

```

```
//--------------------------------------
// FFTs
//--------------------------------------
data_in_time = zeros(64,1);
data_in_time(1,1) = 1;
fft_output = fft(data_in_time);
//fft_output = fft(output_DFT)/det_fft; //in freq


//--------------------------------------
// Frequency Response
//--------------------------------------

//------------------------------------------------------------------------------------------------------------
// Plot using Tikz:
fd = mopen('\Users\Tassniem\Documents/TikZ/IR/FFT_IR/FFT_IR.tex','wt');
mfprintf(fd,'%s \n','\documentclass{article}');
mfprintf(fd,'%s \n','\usepackage{tikz}');
mfprintf(fd,'%s \n','\usepackage{pgfplots}');
mfprintf(fd,'%s \n','\begin{document}');



mfprintf(fd,'%s \n','\begin{center}');

normalization_fft_output = max(abs(fft_output))

normalization_time_impulse = max(abs(output_DFT))

//---------------------------------
// Frequency Response
//---------------------------------

mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[ymax=1.1, ymin=-0.1,');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Real}\{\Psi(\omega)$\}]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
```

```
    mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length, ',', real(fft_output(n,1))/normalization_fft_output, ')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//----------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[ymax=1.1, ymin=-0.1,');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Imag}\{\Psi(\omega)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
    mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length, ',', imag(fft_output(n,1))/normalization_fft_output, ')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//----------------
//----------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[ymax=1.1, ymin=-0.1,');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $|\Psi(\omega)|^2$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
    mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length, ',', fft_output(n,1)*conj(fft_output(n,1))/normalization_fft_output^2, ')');
end

mfprintf(fd,'%s \n','};');

```

```
mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//-----------------
// Impulse Response
//-----------------
mfprintf(fd,'%s \n','\newpage');
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[ymax=1.1, ymin=-0.1,');
mfprintf(fd,'%s \n','xlabel=$t/64$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Real}\{\psi(t)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length, ',', real(output_DFT(n,1))/normalization_time_impulse,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//-----------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[ymax=1.1, ymin=-0.1,');
mfprintf(fd,'%s \n','xlabel=$t/64$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Imag}\{\psi(t)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length, ',', imag(output_DFT(n,1))/normalization_time_impulse,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//-----------------
//-----------------
```

```
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[ymax=1.1, ymin=-0.1,');
mfprintf(fd,'%s \n','xlabel=$t/64$,');
mfprintf(fd,'%s \n','ylabel= $|\psi(t)|$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', abs(
      output_DFT(n,1))/normalization_time_impulse,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//------------------




mfprintf(fd,'%s \n','\end{center}');
mfprintf(fd,'%s \n','\end{document}');
mclose(fd);
```

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

```
clc
clear

//-----------------------------------------------------------------
//CIRCULAR_SHIFT_BY_TWO
function [shifted_sequence] = CIRCULAR_SHIFT_BY_TWO(
    to_be_shifted_sequence),

  to_be_shifted_sequence_length = length(to_be_shifted_sequence);

  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)];

endfunction

//-----------------------------------------------------------------
// DISCRETE_WAVELET_TRANSFORM_MATRIX
function [discrete_wavelet_transform_matrix] =
    DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter,High_filter,zero_padding
    ),

  Length = length(Low_filter) + zero_padding;

  discrete_wavelet_transform_matrix = zeros(Length, Length);

  discrete_wavelet_transform_matrix(1,1:Length) = [flipdim(Low_filter,2),
      zeros(1,zero_padding)];
  discrete_wavelet_transform_matrix(1+Length/2,1:Length) = [flipdim(
      High_filter,2),zeros(1,zero_padding)];

  for row = 2:Length/2,
        discrete_wavelet_transform_matrix(row,1:Length) =
            CIRCULAR_SHIFT_BY_TWO( discrete_wavelet_transform_matrix(row
            -1,1:Length));
        discrete_wavelet_transform_matrix(row+Length/2,1:Length) =
            CIRCULAR_SHIFT_BY_TWO( discrete_wavelet_transform_matrix(row-1+
            Length/2,1:Length));
    end,

endfunction
```

```
//------------------------------------------------------------------
// INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX
function [inverse_discrete_wavelet_transform_matrix] = INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter ,High_filter , zero_padding ) ,

  inverse_discrete_wavelet_transform_matrix = DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter ,High_filter , zero_padding ) ';

endfunction

//------------------------------------------------------------------
// Q_FUNCTION
function [q_function] = Q_FUNCTION(some_value) ,

  q_function = 0.5*erfc(sqrt(0.5)*some_value);

endfunction

//------------------------------------------------------------------
wavelet = dbwavf('db1');// it is db6
[low_analysis_filter , high_analysis_filter , low_synthesis_filter , high_synthesis_filter] = orthfilt(wavelet);
analysis_filter = [low_analysis_filter; high_analysis_filter];
First_analysis_filter = analysis_filter;
synthesis_filter = [low_synthesis_filter; high_synthesis_filter];
Final_synthesis_filter = synthesis_filter;
//-----------------------
//[First_analysis_filter , Final_synthesis_filter] = FSfarras('f');
//[analysis_filter , synthesis_filter] = dualfilt1('f');
//-----------------------
//square_root_half = sqrt(.5);
//analysis_filter = [square_root_half, square_root_half ; -square_root_half, square_root_half];
//First_analysis_filter = analysis_filter;
//synthesis_filter = [square_root_half, square_root_half; square_root_half, -square_root_half];
//Final_synthesis_filter = synthesis_filter;
//------------------------------------------------------------------
no_stages = 4 ;
filter_length = length(analysis_filter(1,:));
```

```
//------------------------------

data_length = 64//filter_length*2^no_stages;

//--------------------------
// Data
//--------------------------
data = zeros(data_length,1);
data(20, 1) = 1;
normalization = sqrt(10);
data_before_norm = data;

//------------------------------
//------------------------------
zero_padding = abs(data_length - filter_length);


//------------------------------------------------------------------
// Number of data per branch
//------------------------------
branch_rate = zeros(1,no_stages+1);// to store # of ele. per branch
branch_rate(1,1) = data_length/2^no_stages;
for k = no_stages:-1:1,
  branch_rate(1,no_stages+2-k) = data_length/2^k;
end
cumsum_branch_rate = cumsum(branch_rate);// matrix to store the
    cumulative sum up to kth elem.
//----------------------
     data_copy = zeros(data_length,1);
     constructed_data = zeros(data_length,1);


//------------------------------------------------------------------
//------------------------
// the transmitter side
//------------------------
//------------------------------------------------------------------


//------------------------
for tree_no = 1:1,

     if tree_no == 1 then
```

```
        low_filter = 1;
        high_filter = 2;
      elseif tree_no == 2 then
        low_filter = 3;
        high_filter = 4;
      end // for elseif
      //-----------------------



       data_copy = data; // data has to be of even length

    //-----------------------------
    // Stages from 1:no_stages-1
    //-----------------------------
    constructed_data(1:cumsum_branch_rate(1,2),1) = data_copy(1:cumsum_branch_rate(1,2),1);


    for stage = 1:no_stages-1,

      // data_length = cumsum_branch_rate(1,stage+1)
      zero_padding = abs(cumsum_branch_rate(1,stage+1) - filter_length);


      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);
    // filters are flipped, due to using same function of the DWT, and they have to be arranged in an inverse order.

      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)];

    end

    //-----------------------------------------
    // Stage number no_stages, "last stage"
    //-----------------------------------------

```

```
  zero_padding = abs(cumsum_branch_rate(1,no_stages+1) - filter_length);


  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);
  // filters are flipped, due to using same function of the DWT, and they
       have to be arranged in an inverse order.







  //---------------------------
  // to store output of both trees
  if tree_no == 1 then
    output_tree1 = constructed_data

  elseif tree_no == 2 then
    output_tree2 = constructed_data

  end // if tree_no


end // for tree_no


//-----------------------
// FFTs
//-----------------------

fft_output_tree1 = fft(output_tree1);

//------------------------------------------------------------------


// Plot using Tikz:
fd = mopen('\Users\Tassniem\Documents/TikZ/IR/DWT_Haar_IR/DWT_Haar_IR.tex
    ','wt');
```

```
mfprintf(fd,'%s \n','\documentclass{article}');
mfprintf(fd,'%s \n','\usepackage{tikz}');
mfprintf(fd,'%s \n','\usepackage{pgfplots}');
mfprintf(fd,'%s \n','\begin{document}');


mfprintf(fd,'%s \n','\begin{center}');


normalization_time1 = max(abs(output_tree1))

normalization_freq1 = max(abs(fft_output_tree1))

//--------------------
// impulse_response in Frequency
//--------------------

mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Real}\{\Psi_h(\omega)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', real(fft_output_tree1(n,1))/normalization_freq1,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//--------------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Imag}\{\Psi_h(\omega)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

```

```
for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', imag(fft_output_tree1(n,1))/normalization_freq1,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');

//------------------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $|\Psi_h(\omega)|^2$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', (abs(fft_output_tree1(n,1)))^2/normalization_freq1^2,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//------------------------

// impulse_response in Time
//------------------------
// Upper
//------------------------
mfprintf(fd,'%s \n','\newpage');
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel= $t/64$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Real}\{\psi_h(t)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');
```

```
for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', real(output_tree1(n,1))/normalization_time1,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//------------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[ymax=1.1, ymin=-0.1,');
mfprintf(fd,'%s \n','xlabel= $t/64$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Imag}\{\psi_h(t)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', imag(output_tree1(n,1))/normalization_time1,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//------------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$t/64$,');
mfprintf(fd,'%s \n','ylabel= $|\psi_h(t)|$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', abs(output_tree1(n,1))/normalization_time1,')');
end

mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\end{axis}');
```

```
mfprintf(fd,'%s \n','\end{tikzpicture}');
//-----------------------------


mfprintf(fd,'%s \n','\end{center}');
mfprintf(fd,'%s \n','\end{document}');
mclose(fd);
```

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

```
clc
clear

//--------------------------------------------------------------------------
//CIRCULAR_SHIFT_BY_TWO
function [shifted_sequence] = CIRCULAR_SHIFT_BY_TWO(to_be_shifted_sequence),

  to_be_shifted_sequence_length = length(to_be_shifted_sequence);

  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)];

endfunction

//--------------------------------------------------------------------------
// DISCRETE_WAVELET_TRANSFORM_MATRIX
function [discrete_wavelet_transform_matrix] = DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter,High_filter,zero_padding),

  Length = length(Low_filter) + zero_padding;

  discrete_wavelet_transform_matrix = zeros(Length, Length);

  discrete_wavelet_transform_matrix(1,1:Length) = [flipdim(Low_filter,2), zeros(1,zero_padding)];
  discrete_wavelet_transform_matrix(1+Length/2,1:Length) = [flipdim(High_filter,2),zeros(1,zero_padding)];

  for row = 2:Length/2,
      discrete_wavelet_transform_matrix(row,1:Length) = CIRCULAR_SHIFT_BY_TWO( discrete_wavelet_transform_matrix(row-1,1:Length));
      discrete_wavelet_transform_matrix(row+Length/2,1:Length) = CIRCULAR_SHIFT_BY_TWO( discrete_wavelet_transform_matrix(row-1+Length/2,1:Length));
    end,

endfunction
```

```
//----------------------------------------------------------------------
// INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX
function [inverse_discrete_wavelet_transform_matrix] = INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter ,High_filter , zero_padding) ,

  inverse_discrete_wavelet_transform_matrix = DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter ,High_filter , zero_padding) ';

endfunction

//----------------------------------------------------------------------
// Q_FUNCTION
function [q_function] = Q_FUNCTION(some_value) ,

  q_function = 0.5*erfc(sqrt(0.5)*some_value);

endfunction

//----------------------------------------------------------------------
wavelet = dbwavf('db3');// it is db6
[low_analysis_filter , high_analysis_filter , low_synthesis_filter , high_synthesis_filter] = orthfilt(wavelet);
analysis_filter = [low_analysis_filter; high_analysis_filter];
First_analysis_filter = analysis_filter;
synthesis_filter = [low_synthesis_filter; high_synthesis_filter];
Final_synthesis_filter = synthesis_filter;
//-------------------------
//[First_analysis_filter , Final_synthesis_filter] = FSfarras('f');
//[analysis_filter , synthesis_filter] = dualfilt1('f');
//-------------------------
//square_root_half = sqrt(.5);
//analysis_filter = [square_root_half , square_root_half ; -square_root_half , square_root_half];
//First_analysis_filter = analysis_filter;
//synthesis_filter = [square_root_half , square_root_half; square_root_half , -square_root_half];
//Final_synthesis_filter = synthesis_filter;
//----------------------------------------------------------------------
no_stages = 4 ;
filter_length = length(analysis_filter(1,:));
```

```
//--------------------------------------------

data_length = 64//filter_length*2^no_stages;

//-------------------------------------
// Data
//-------------------------------------
data = zeros(data_length,1);
data(20, 1) = 1;
normalization = sqrt(10);
data_before_norm = data;

//--------------------------------------------
//--------------------------------------------
zero_padding = abs(data_length - filter_length);


//--------------------------------------------------------------------------------------------
// Number of data per branch
//--------------------------------------------
branch_rate = zeros(1,no_stages+1);// to store # of ele. per branch
branch_rate(1,1) = data_length/2^no_stages;
for k = no_stages:-1:1,
  branch_rate(1,no_stages+2-k) = data_length/2^k;
end
cumsum_branch_rate = cumsum(branch_rate);// matrix to store the cumulative sum up to kth elem.
//---------------------------------
    data_copy = zeros(data_length,1);
    constructed_data = zeros(data_length,1);


//--------------------------------------------------------------------------------------------
//-----------------------------------
// the transmitter side
//-----------------------------------
//--------------------------------------------------------------------------------------------


//-----------------------------------
for tree_no = 1:1,

    if tree_no == 1 then
```

```
      low_filter = 1;
      high_filter = 2;
    elseif tree_no == 2 then
      low_filter = 3;
      high_filter = 4;
    end // for elseif
    //--------------------------



     data_copy = data; // data has to be of even length

  //----------------------------------
  // Stages from 1:no_stages-1
  //----------------------------------
  constructed_data(1:cumsum_branch_rate(1,2),1) = data_copy(1:cumsum_branch_rate(1,2),1);


  for stage = 1:no_stages-1,

    // data_length = cumsum_branch_rate(1,stage+1)
    zero_padding = abs(cumsum_branch_rate(1,stage+1) - filter_length);


    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);
  // filters are flipped, due to using same function of the DWT, and they have to be arranged in an inverse order.

    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)];

  end

  //------------------------------------------------
  // Stage number no_stages, "last stage"
  //------------------------------------------------

```

```
    zero_padding = abs(cumsum_branch_rate(1,no_stages+1) - filter_length);


    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);
    // filters are flipped, due to using same function of the DWT, and they have to be arranged in an inverse order.








    //----------------------
    // to store output of both trees
    if tree_no == 1 then
      output_tree1 = constructed_data

    elseif tree_no == 2 then
      output_tree2 = constructed_data

    end // if tree_no






end // for tree_no



//--------------------
// FFTs
//--------------------

fft_output_tree1 = fft(constructed_data);

//--------------------
```

```
// ESD
//------------------

energy_spectral_density_tree1 = fft_output_tree1.*conj(fft_output_tree1)/data_length;

//------------------
// Impulse Response in frequency
//------------------

freq_impulse_response_tree1 = sqrt(fft_output_tree1.*conj(fft_output_tree1));

//--------------------------------------------------

//------------------
// FFTs
//------------------

fft_output_tree1 = fft(output_tree1);

//--------------------------------------------------


// Plot using Tikz:
fd = mopen('\Users\Tassniem\Documents/TikZ/IR/DWT_D6_IR/DWT_D6_IR.tex','wt');
mfprintf(fd,'%s \n','\documentclass{article}');
mfprintf(fd,'%s \n','\usepackage{tikz}');
mfprintf(fd,'%s \n','\usepackage{pgfplots}');
mfprintf(fd,'%s \n','\begin{document}');



mfprintf(fd,'%s \n','\begin{center}');


normalization_time1 = max(abs(output_tree1))

normalization_freq1 = max(abs(fft_output_tree1))

//----------------
// impulse_response in Frequency
```

```
//--------------------------------------------

mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Real}\{\Psi_h(\omega)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', real(
      fft_output_tree1(n,1))/normalization_freq1,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//--------------------------------------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Imag}\{\Psi_h(\omega)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', imag(
      fft_output_tree1(n,1))/normalization_freq1,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');

//--------------------------------------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $|\Psi_h(\omega)|^2$]');
```

```
mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', (abs(fft_output_tree1(n,1)))^2/normalization_freq1^2,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//----------------

// impulse_response in Time
//----------------
// Upper
//----------------
mfprintf(fd,'%s \n','\newpage');
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel= $t/64$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Real}\{\psi_h(t)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', real(output_tree1(n,1))/normalization_time1,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//----------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[ymax=1.1, ymin=-0.1,');
mfprintf(fd,'%s \n','xlabel= $t/64$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Imag}\{\psi_h(t)\}$]');
```

```
mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', imag(output_tree1(n,1))/normalization_time1,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//-----------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$t/64$,');
mfprintf(fd,'%s \n','ylabel= $|\psi_h(t)|$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', abs(output_tree1(n,1))/normalization_time1,')');
end

mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//-----------------


mfprintf(fd,'%s \n','\end{center}');
mfprintf(fd,'%s \n','\end{document}');
mclose(fd);
```

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

```
clc
clear

//------------------------------------------------------------------
//CIRCULAR_SHIFT_BY_TWO
function [shifted_sequence] = CIRCULAR_SHIFT_BY_TWO(to_be_shifted_sequence),

  to_be_shifted_sequence_length = length(to_be_shifted_sequence);

  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)];

endfunction

//------------------------------------------------------------------
// DISCRETE_WAVELET_TRANSFORM_MATRIX
function [discrete_wavelet_transform_matrix] = DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter ,High_filter ,zero_padding ),

  Length = length(Low_filter) + zero_padding;

  discrete_wavelet_transform_matrix = zeros(Length, Length);

  discrete_wavelet_transform_matrix(1,1:Length) = [(Low_filter),zeros(1,zero_padding)];
  discrete_wavelet_transform_matrix(1+Length/2,1:Length) = [(High_filter),zeros(1,zero_padding)];

  for row = 2:Length/2,
      discrete_wavelet_transform_matrix(row,1:Length) = CIRCULAR_SHIFT_BY_TWO( discrete_wavelet_transform_matrix(row-1,1:Length));
      discrete_wavelet_transform_matrix(row+Length/2,1:Length) = CIRCULAR_SHIFT_BY_TWO( discrete_wavelet_transform_matrix(row-1+Length/2,1:Length));
   end,

endfunction
```

```

//------------------------------------------------------------------------
// INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX
function [inverse_discrete_wavelet_transform_matrix] = INVERSE_DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter,High_filter, zero_padding),

  inverse_discrete_wavelet_transform_matrix = (DISCRETE_WAVELET_TRANSFORM_MATRIX( Low_filter,High_filter, zero_padding)).';

endfunction

//------------------------------------------------------------------------
// Q_FUNCTION
function [q_function] = Q_FUNCTION(some_value),

  q_function = 0.5*erfc(sqrt(0.5)*some_value);

endfunction

//------------------------------------------------------------------------
// AVERAGE
function [average] = AVERAGE(some_vector),

  average = sum(some_vector)/length(some_vector);

endfunction

//------------------------------------------------------------------------
//wavelet = dbwavf('db1');// it is db6
//[low_analysis_filter, high_analysis_filter, low_synthesis_filter, high_synthesis_filter] = orthfilt(wavelet);
//analysis_filter = [low_analysis_filter; high_analysis_filter];
//First_analysis_filter = analysis_filter;
//synthesis_filter = [low_synthesis_filter; high_synthesis_filter];
//Final_synthesis_filter = synthesis_filter;
//-------------------------
[First_analysis_filter, Final_synthesis_filter] = FSfarras('f');
[analysis_filter, synthesis_filter] = dualfilt1('f');
//-------------------------
//square_root_half = sqrt(.5);
```

```
//analysis_filter = [square_root_half, square_root_half ; -square_root_half, square_root_half];
//First_analysis_filter = analysis_filter;
//synthesis_filter = [square_root_half, square_root_half; square_root_half, -square_root_half];
//Final_synthesis_filter = synthesis_filter;
//----------------------------------------------------------------------
no_stages = 2;
filter_length = length(analysis_filter(1,:));
//------------------------------
// data_length := data is in symbols
data_length = 64//filter_length*2^no_stages;//64

//multi_output_tree1 = zeros(data_length,1);
//multi_output_tree2 = zeros(data_length,1);

number_of_times_to_avg_on = 1;


//-------------------------
// Data
//-------------------------
data = zeros(data_length,1);
data(20, 1) = 1;
normalization = sqrt(10);
data_before_norm = data;
//data_before_norm = grand(data_length,2,'uin',0,1);
//random = rand(data_before_norm,'normal');
//for L = 1:data_length,
//   if random(L,1) > 0 then
//       data_before_norm(L,1) = 2*data_before_norm(L,1)+1;
//   else data_before_norm(L,1) = -1*(2*data_before_norm(L,1)+1);
//   end
//     if random(L,2) > 0 then
//       data_before_norm(L,2) = 2*data_before_norm(L,2)+1;
//   else data_before_norm(L,2) = -1*(2*data_before_norm(L,2)+1);
//   end

//   end

//normalization = sqrt(10);
//data = data_before_norm; //normalized

```

```
//------------------------------
zero_padding = abs(data_length - filter_length);


//------------------------------------------------------------
// Number of data per branch
//------------------------------
branch_rate = zeros(1,no_stages+1);// to store # of ele. per branch
branch_rate(1,1) = data_length/2^no_stages;
for k = no_stages:-1:1,
  branch_rate(1,no_stages+2-k) = data_length/2^k;
end
cumsum_branch_rate = cumsum(branch_rate);// matrix to store the cumulative sum up to kth elem.
//------------------------
    data_copy = zeros(data_length,1);
    constructed_data = zeros(data_length,1);


//------------------------------------------------------------
//-----------------------
// the transmitter side
//-----------------------
//------------------------------------------------------------


//-----------------------
for tree_no = 1:2,

    if tree_no == 1 then
      low_filter = 1;
      high_filter = 2;
    elseif tree_no == 2 then
      low_filter = 3;
      high_filter = 4;
    end // for elseif
    //-----------------------



     data_copy = data; // data has to be of even length

  //----------------------------
```

```
// Stages from 1:no_stages-1
//----------------------------
constructed_data(1:cumsum_branch_rate(1,2),1) = data_copy(1:cumsum_branch_rate(1,2),1);


for stage = 1:no_stages-1,

  // data_length = cumsum_branch_rate(1,stage+1)
  zero_padding = abs(cumsum_branch_rate(1,stage+1) - filter_length);


  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);
// filters are flipped, due to using same function of the DWT, and they have to be arranged in an inverse order.

  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)];

end

//----------------------------------------
// Stage number no_stages, "last stage"
//----------------------------------------

zero_padding = abs(cumsum_branch_rate(1,no_stages+1) - filter_length);


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);
// filters are flipped, due to using same function of the DWT, and they have to be arranged in an inverse order.
```

```
  //------------------------------------
  // to store output of both trees
  if tree_no == 1 then
    output_tree1 = constructed_data

  elseif tree_no == 2 then
    output_tree2 = constructed_data

  end // if tree_no






end // for tree_no

both_trees_output = output_tree1 + %i*output_tree2;





//------------------------------------------------------------------------------------------------------------



//--------------------------------
// FFTs
//--------------------------------



fft_output_tree1 = fft(output_tree1);
fft_output_tree2 = fft(output_tree2);
fft_output_both_trees = fft(both_trees_output);

//----------------------------------------------------------------------------------------------
```

```
// Plot using Tikz:
fd = mopen('\Users\Tassniem\Documents/TikZ/IR/DTCWT_IR/DTCWT_IR.tex','wt');
mfprintf(fd,'%s \n','\documentclass{article}');
mfprintf(fd,'%s \n','\usepackage{tikz}');
mfprintf(fd,'%s \n','\usepackage{pgfplots}');
mfprintf(fd,'%s \n','\begin{document}');



mfprintf(fd,'%s \n','\begin{center}');


normalization_time1 = max(abs(output_tree1))
normalization_time2 = max(abs(output_tree2))
normalization_time_DT = max(abs(both_trees_output))


normalization_freq1 = max(abs(fft_output_tree1))
normalization_freq2 = max(abs(fft_output_tree2))
normalization_freq_DT = max(abs(fft_output_both_trees))

//------------------
// impulse_response in Frequency
//------------------

mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Real}\{\Psi_h(\omega)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', real(fft_output_tree1(n,1))/normalization_freq1,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
```

```
mfprintf(fd,'%s \n','\end{tikzpicture}');
//----------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Imag}\{\Psi_h(\omega)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', imag(fft_output_tree1(n,1))/normalization_freq1,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');

//----------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $|\Psi_h(\omega)|^2$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', (abs(fft_output_tree1(n,1)))^2/normalization_freq1^2,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//----------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Real}\{\Psi_g(\omega)\}$]');
```

```
mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', real(fft_output_tree2(n,1))/normalization_freq2,')');
end

mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//-----------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Imag}\{\Psi_g(\omega)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', imag(fft_output_tree2(n,1))/normalization_freq2,')');
end

mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');

//-----------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $|\Psi_g(\omega)|^2$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', (abs(fft_output_tree2(n,1)))^2/normalization_freq2^2,')');
end
```

```
mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//--------------------

mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Real}\{\Psi_h(\omega) + j\Psi_g(\
    omega)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', real(
      fft_output_both_trees(n,1))/normalization_freq_DT,')');
end

mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//--------------------

mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$\omega/2\pi$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Imag}\{\Psi_h(\omega) + j\Psi_g(\
    omega)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', imag(
      fft_output_both_trees(n,1))/normalization_freq_DT,')');
end

mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
```

```
//-------------------
mfprintf(fd, '%s \n', '\begin{tikzpicture}');
mfprintf(fd, '%s \n', '\begin{axis}[');
mfprintf(fd, '%s \n', 'xlabel=$\omega/2\pi$,');
mfprintf(fd, '%s \n', 'ylabel= $|\Psi_h(\omega) + j\Psi_g(\omega)|^2$]');

mfprintf(fd, '%s \n', '\addplot[color=black]');
mfprintf(fd, '%s \n', 'coordinates{');

for n = 1:data_length,
  mfprintf(fd, '%s %.6f %s %.6f %s \n', '(', n/data_length, ',', (abs(
      fft_output_both_trees(n,1)))^2/normalization_freq_DT^2, ')');
end

mfprintf(fd, '%s \n', '};');
mfprintf(fd, '%s \n', '\end{axis}');
mfprintf(fd, '%s \n', '\end{tikzpicture}');
//-------------------
// impulse_response in Time
//-------------------
// Upper
//-------------------
mfprintf(fd, '%s \n', '\newpage');
mfprintf(fd, '%s \n', '\begin{tikzpicture}');
mfprintf(fd, '%s \n', '\begin{axis}[');
mfprintf(fd, '%s \n', 'xlabel= $t/64$,');
mfprintf(fd, '%s \n', 'ylabel= $\mathrm{Real}\{\psi_h(t)\}$]');

mfprintf(fd, '%s \n', '\addplot[color=black]');
mfprintf(fd, '%s \n', 'coordinates{');

for n = 1:data_length,
  mfprintf(fd, '%s %.6f %s %.6f %s \n', '(', n/data_length, ',', real(
      output_tree1(n,1))/normalization_time1, ')');
end

mfprintf(fd, '%s \n', '};');

mfprintf(fd, '%s \n', '\end{axis}');
mfprintf(fd, '%s \n', '\end{tikzpicture}');
//-------------------
mfprintf(fd, '%s \n', '\begin{tikzpicture}');
mfprintf(fd, '%s \n', '\begin{axis}[ymax=1.1, ymin=-0.1,');
```

```
mfprintf(fd,'%s \n','xlabel= $t/64$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Imag}\{\psi_h(t)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', imag(
      output_tree1(n,1))/normalization_time1,')');
end

mfprintf(fd,'%s \n','};');

mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//---------------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$t/64$,');
mfprintf(fd,'%s \n','ylabel= $|\psi_h(t)|$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', abs(
      output_tree1(n,1))/normalization_time1,')');
end

mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//---------------------
//---------------------
// Lower
//---------------------
mfprintf(fd,'%s \n','\newpage');
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel= $t/64$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Real}\{\psi_g(t)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
```

```
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', real(output_tree2(n,1))/normalization_time2,')');
end

mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//----------------------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[ymax=1.1, ymin=-0.1,');
mfprintf(fd,'%s \n','xlabel= $t/64$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Imag}\{\psi_g(t)\}$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', imag(output_tree2(n,1))/normalization_time2,')');
end

mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//----------------------------

//----------------------------

mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$t/64$,');
mfprintf(fd,'%s \n','ylabel= $|\psi_g(t)|$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', abs(output_tree2(n,1))/normalization_time2,')');
end
```

```
mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//--------------------
// Both
//--------------------
mfprintf(fd,'%s \n','\newpage');
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$t/64$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Real}\{\psi_h(t) + j\psi_g(t)\}$]')
    ;

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', real(
      both_trees_output(n,1))/normalization_time1,')');
end

mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\end{axis}');
//--------------------
mfprintf(fd,'%s \n','\end{tikzpicture}');
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$t/64$,');
mfprintf(fd,'%s \n','ylabel= $\mathrm{Imag}\{\psi_h(t) + j\psi_g(t)\}$]')
    ;

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', imag(
      both_trees_output(n,1))/normalization_time2,')');
end

mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
```

```
//--------------------

//--------------------

mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{axis}[');
mfprintf(fd,'%s \n','xlabel=$t/64$,');
mfprintf(fd,'%s \n','ylabel= $|\psi_h(t) + j\psi_g(t)|$]');

mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');

for n = 1:data_length,
  mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n/data_length,',', abs(both_trees_output(n,1))/normalization_time_DT,')');
end

mfprintf(fd,'%s \n','};');
mfprintf(fd,'%s \n','\end{axis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//--------------------


mfprintf(fd,'%s \n','\end{center}');
mfprintf(fd,'%s \n','\end{document}');
mclose(fd);
```

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

```
clc
clear
//----------------------------------------------------------------
//CIRCULAR_SHIFT_BY_TWO
function [shifted_sequence] = CIRCULAR_SHIFT_BY_TWO( to_be_shifted_sequence),

  to_be_shifted_sequence_length = length(to_be_shifted_sequence);

  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)];

endfunction
//----------------------------------------------------------------
// MATRIX_EVALUATION
function [matrix_evaluation] = MATRIX_EVALUATION(transformation_length, filters),
  [r, c] = size(filters);
  matrix_evaluation = zeros(transformation_length, transformation_length);

  matrix_evaluation(1,1:c) = filters(1,1:c);
  matrix_evaluation(1+transformation_length/2,1:c) = filters(2,1:c);

  for row = 2:transformation_length/2,
      matrix_evaluation(row,1:c) = CIRCULAR_SHIFT_BY_TWO( matrix_evaluation(row-1,1:c));
      matrix_evaluation(row+transformation_length/2,1:c) = CIRCULAR_SHIFT_BY_TWO( matrix_evaluation(row-1+transformation_length/2,1:c));
    end
endfunction
//----------------------------------------------------------------
// DWT
function [dwt_matrix] =DWT(data_length,filter_length,no_stages,filters),

  dwt_matrix = eye(data_length,data_length);

```

```
  if no_stages > 1 then

    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)]);

    if no_stages > 2 then
      for stage = no_stages-1:-1:2,
        matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters(3:4,:), zeros(2,data_length/2^stage-filter_length)]);
        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);
        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);
      end
    end
  end
  matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,data_length-filter_length)]);
  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);
  dwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(data_length/2+1:data_length,1:data_length);

endfunction
//------------------------------------------------------------------
// IDWT
function [idwt_matrix] = IDWT(data_length,filter_length,no_stages,filters),

  idwt_matrix = eye(data_length,data_length);

    if no_stages > 1 then
      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)]);

      if no_stages > 2 then
        for stage = no_stages-1:-1:2,
          matrix_i = MATRIX_EVALUATION(data_length/2^stage,[filters(3:4,:),zeros(2,data_length/2^stage-filter_length)]);
```

```
            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);
            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);
          end
        end
      end
      matrix_i = MATRIX_EVALUATION(data_length,[filters(1:2,:),zeros(2,
          data_length-filter_length)]);
      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);
      idwt_matrix(data_length/2+1:data_length,1:data_length) = matrix_i(
          data_length/2+1:data_length,1:data_length);

  idwt_matrix = idwt_matrix.';
endfunction
//------------------------------------------------------------------
// Q_FUNCTION
function [q_function] = Q_FUNCTION(some_value),

  q_function = 0.5*erfc(sqrt(0.5)*some_value);

endfunction
//------------------------------------------------------------------
// AVERAGE
function [average] = AVERAGE(some_vector),

  average = sum(some_vector)/length(some_vector);

endfunction
//------------------------------------------------------------------
stacksize(800000)
number_of_OFDM_symbols = 3*10^5;
data_length = 64;
number_channels = 64;
sqrt_no_channels = sqrt(number_channels);
data_before_norm = zeros(data_length,2);
PAPR_range = 0:.1:10;
length_PAPR_range = length(PAPR_range);
```

```
//---------------------------------------------------------------
// counter:
counter = zeros(length_PAPR_range,6);
// transform
output = zeros(data_length,1);
// power:
p_power_output = zeros(data_length,1);
// Average power:
average_power_output = zeros(1,1);
//---------------------------------------------------------------
for no_of_OFDM_symbols = 1:number_of_OFDM_symbols
  //------------------------
  // Data
  //------------------------
  data_before_norm = grand(data_length,2,'uin',0,1);
  random = rand(data_before_norm,'normal');
  for L = 1:data_length,
    if random(L,1) > 0 then
      data_before_norm(L,1) = 2*data_before_norm(L,1)+1;
    else data_before_norm(L,1) = -1*(2*data_before_norm(L,1)+1);
    end
     if random(L,2) > 0 then
      data_before_norm(L,2) = 2*data_before_norm(L,2)+1;
    else data_before_norm(L,2) = -1*(2*data_before_norm(L,2)+1);
    end
  end
    normalization = sqrt(10);
    data = (data_before_norm(1:data_length,1) + %i*data_before_norm(1:
        data_length,2))/normalization; //normalized
    //------------------------
  for transformation = 1:6,
    // 1:DFT
    // 2:DWT-Haar
    // 3:DWT-D6
    // 4:Upper-DT-CWT
    // 5:Lower-DT-CWT
    // 6:DT-CWT
    //------------------------
    if transformation == 1 then // 1:DFT
      output = sqrt_no_channels*ifft(data);
    end
    //------------------------
    if transformation == 2 then // 2:DWT-Haar
```

```
      no_stages = 4 ;
      low_filter = 1;
      high_filter = 2;
      wavelet = dbwavf('db1');// it is Haar
      [low_analysis_filter, high_analysis_filter, low_synthesis_filter,
          high_synthesis_filter] = orthfilt(wavelet);
      analysis_filter = [low_analysis_filter; high_analysis_filter];
      First_analysis_filter = analysis_filter;
      synthesis_filter = [low_synthesis_filter; high_synthesis_filter];
      Final_synthesis_filter = synthesis_filter;

    elseif transformation == 3 then // 3:DWT-D6
      no_stages = 3 ;
      low_filter = 1;
      high_filter = 2;
      wavelet = dbwavf('db3');// it is D-6
      [low_analysis_filter, high_analysis_filter, low_synthesis_filter,
          high_synthesis_filter] = orthfilt(wavelet);
      analysis_filter = [low_analysis_filter; high_analysis_filter];
      First_analysis_filter = analysis_filter;
      synthesis_filter = [low_synthesis_filter; high_synthesis_filter];
      Final_synthesis_filter = synthesis_filter;

    elseif transformation == 4 then // 4:Upper-DT-CWT
      no_stages = 2;
      low_filter = 1;
      high_filter = 2;
      [First_analysis_filter, Final_synthesis_filter] = FSfarras('f');
      [analysis_filter, synthesis_filter] = dualfilt1('f');

    elseif transformation == 5 then // 5:Lower-DT-CWT
      no_stages = 2;
      low_filter = 3;
      high_filter = 4;
      [First_analysis_filter, Final_synthesis_filter] = FSfarras('f');
      [analysis_filter, synthesis_filter] = dualfilt1('f');

    end
    //------------------------------
    if transformation ~= 1 & transformation ~= 6 then

      ANALYSIS_FILTERS = [First_analysis_filter(low_filter:high_filter,:)
          ;analysis_filter(low_filter:high_filter,:)];
```

```
        SYNTHESIS_FILTERS = [Final_synthesis_filter(low_filter:high_filter
            ,:);synthesis_filter(low_filter:high_filter,:)];

        [row, filter_length] = size(analysis_filter(1,:));

        output = IDWT(data_length, filter_length, no_stages, flipdim(
            SYNTHESIS_FILTERS,2))*data;
      end


      if transformation ~= 6 then
        power_output = abs(output.*conj(output))/data_length;

        average_power_output = AVERAGE(power_output);
        p_power_output = power_output/average_power_output;
      end


      if transformation == 4 then p_output_Upper = p_power_output;
        elseif transformation == 5 then p_output_Lower = p_power_output;
      end

      if transformation == 6 then
        power_output = p_output_Upper + p_output_Lower;
        average_power_output = AVERAGE(power_output);
        p_power_output = power_output/average_power_output;
      end

      for DL = 1:data_length
        location = 1;
        for PAPR_0 = PAPR_range
          if p_power_output(data_length,1) > PAPR_0 then counter(location,
              transformation) = counter(location,transformation) + 1;
            location = location + 1;
          end
        end
      end
    end
end
counter = counter/number_of_OFDM_symbols/data_length;
PAPR_range_sub = 0:.5:10;
// Plot using Tikz:
```

```
fd = mopen('\Users\Tassniem\Documents/TikZ/PAPR/PAPR_figure.tex','wt');
mfprintf(fd,'%s \n','\documentclass{article}');
mfprintf(fd,'%s \n','\usepackage{tikz}');
mfprintf(fd,'%s \n','\usepackage{pgfplots}');
mfprintf(fd,'%s \n','\begin{document}');
mfprintf(fd,'%s \n','\begin{center}');
//-------------------
mfprintf(fd,'%s \n','\begin{tikzpicture}');
mfprintf(fd,'%s \n','\begin{semilogyaxis}[xmin=-0.5, xmax=10, width=12 cm,');
mfprintf(fd,'%s \n','xlabel=$\mathrm{PAPR}_0$,');
mfprintf(fd,'%s \n','ylabel=$\mathbf{P}_\mathrm{PAPR}$]');
mfprintf(fd,'%s \n','\pgfplotsset{every axis legend/.append style={at={(0.72,0.66)},anchor=south}}, ');
//"Marks only"
//-------------------
// DFT
//-------------------
mfprintf(fd,'%s \n','\addplot[color=black,mark=*,only marks]');
mfprintf(fd,'%s \n','coordinates{');
m = 1;
for n = PAPR_range_sub,
   mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n,',', counter(m,1),')');
   m = m + 5;
end
mfprintf(fd,'%s \n','};');
//-------------------
mfprintf(fd,'%s \n','\addlegendentry{DFT-Based} ');
//-------------------
// DWT-Haar
//-------------------
mfprintf(fd,'%s \n','\addplot[color=black,mark=+,only marks]');
mfprintf(fd,'%s \n','coordinates{');
m = 1;
for n = PAPR_range_sub,
   mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n,',', counter(m,2),')');
   m = m + 5;
end
mfprintf(fd,'%s \n','};');
//-------------------
mfprintf(fd,'%s \n','\addlegendentry{DWT-Based, Haar filters} ');
//-------------------
// DWT-D6
```

```
//--------------------------------
mfprintf(fd, '%s \n', '\addplot[color=black,mark=x,only marks]');
mfprintf(fd, '%s \n', 'coordinates{');
m = 1;
for n = PAPR_range_sub,
  mfprintf(fd, '%s %.6f %s %.6f %s \n', '(', n, ',', counter(m,3), ')');
  m = m + 5;
end
mfprintf(fd, '%s \n', '};');
//--------------------------------
mfprintf(fd, '%s \n', '\addlegendentry{DWT-Based, D-6 filters} ');
//--------------------------------
// DT-CWT
//--------------------------------
mfprintf(fd, '%s \n', '\addplot[color=black,mark=triangle,only marks]');
mfprintf(fd, '%s \n', 'coordinates{');
m = 1;
for n = PAPR_range_sub,
  mfprintf(fd, '%s %.6f %s %.6f %s \n', '(', n, ',', counter(m,4), ')');
  m = m + 5;
end
mfprintf(fd, '%s \n', '};');
//--------------------------------
mfprintf(fd, '%s \n', '\addlegendentry{DT-CWT-Based, Upper-tree} ');
//--------------------------------
//--------------------------------
mfprintf(fd, '%s \n', '\addplot[color=black,mark=o,only marks]');
mfprintf(fd, '%s \n', 'coordinates{');
m = 1;
for n = PAPR_range_sub,
  mfprintf(fd, '%s %.6f %s %.6f %s \n', '(', n, ',', counter(m,5), ')');
  m = m + 5;
end
mfprintf(fd, '%s \n', '};');
//--------------------------------
mfprintf(fd, '%s \n', '\addlegendentry{DT-CWT-Based, Lower-tree} ');
//--------------------------------
//--------------------------------
mfprintf(fd, '%s \n', '\addplot[color=black,mark=square,only marks]');
mfprintf(fd, '%s \n', 'coordinates{');
m = 1;
for n = PAPR_range_sub,
  mfprintf(fd, '%s %.6f %s %.6f %s \n', '(', n, ',', counter(m,6), ')');
```

```
    m = m + 5;
end
mfprintf(fd, '%s \n', '};');
//-------------------
mfprintf(fd, '%s \n', '\addlegendentry{DT-CWT-Based, Dual-tree} ');
//"Without marks"
//-------------------
// DFT
//-------------------
mfprintf(fd, '%s \n', '\addplot[color=black]');
mfprintf(fd, '%s \n', 'coordinates{');
m = 0;
for n = PAPR_range,
    m = m + 1;
    mfprintf(fd, '%s %.6f %s %.6f %s \n', '(', n, ',', counter(m,1), ')');
end
mfprintf(fd, '%s \n', '};');
//-------------------
// DWT-Haar
//-------------------
mfprintf(fd, '%s \n', '\addplot[color=black]');
mfprintf(fd, '%s \n', 'coordinates{');
m = 0;
for n = PAPR_range,
    m = m + 1;
    mfprintf(fd, '%s %.6f %s %.6f %s \n', '(', n, ',', counter(m,2), ')');
end
mfprintf(fd, '%s \n', '};');
//-------------------
// DWT-D6
//-------------------
mfprintf(fd, '%s \n', '\addplot[color=black]');
mfprintf(fd, '%s \n', 'coordinates{');
m = 0;
for n = PAPR_range,
    m = m + 1;
    mfprintf(fd, '%s %.6f %s %.6f %s \n', '(', n, ',', counter(m,3), ')');
end
mfprintf(fd, '%s \n', '};');
//-------------------
// DT-CWT
//-------------------
mfprintf(fd, '%s \n', '\addplot[color=black]');
```

```
mfprintf(fd,'%s \n','coordinates{');
m = 0;
for n = PAPR_range,
  m = m + 1;
   mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n,',', counter(m,4),')');
end
mfprintf(fd,'%s \n','};');
//----------------------
//----------------------
mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');
m = 0;
for n = PAPR_range,
  m = m + 1;
   mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n,',', counter(m,5),')');
end
mfprintf(fd,'%s \n','};');
//----------------------
//----------------------
mfprintf(fd,'%s \n','\addplot[color=black]');
mfprintf(fd,'%s \n','coordinates{');
m = 0;
for n = PAPR_range,
  m = m + 1;
   mfprintf(fd,'%s %.6f %s %.6f %s \n','(', n,',', counter(m,6),')');
end
mfprintf(fd,'%s \n','};');
//----------------------
mfprintf(fd,'%s \n','\end{semilogyaxis}');
mfprintf(fd,'%s \n','\end{tikzpicture}');
//----------------------
mfprintf(fd,'%s \n','\end{center}');
mfprintf(fd,'%s \n','\end{document}');
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.