The Impact of Non-Ideal Analog Factors on OFDM Signaling

By: Chiko Lee, B. Eng.

A thesis submitted to the
Faculty of Graduate Studies and Research
In partial fulfillment of the requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario

Your file Votre référence
ISBN: 0-494-00749-4
Our file Notre référence
ISBN: 0-494-00749-4

ABSTRACT

In any communication system, such as in OFDM signaling systems, interfacing between digital and analog domain is a required function within the system. This thesis focuses on such analog interfacing factors, specifically the impact it has on the OFDM baseband signal in typical hardware systems. To investigate these factors, hardware experiments are performed then emulated in an elaborate software model to verify contribution from DC offset error, gain error, I/Q amplitude imbalance, and fixed DAC /ADC resolution. To perform the hardware experiments in this study, a Texas Instruments (TI) TMS320C6711 DSK development kit, equipped with a C6711 floating point processor was studied, programmed, and configured to perform OFDM modulation needed. Further investigations use software simulations to isolate each analog factor and combine them to characterize its effect on performance. In addition, the simulations also examine, ADC saturation, and relative I/Q delay. The performance of received signal at the demodulator is measured by MSE of the waveform (MSE_{wave}), MSE of the constellation (MSE_{cst}), signal to noise ratio normalized to a bit (SIR_{bit}).

The results of the experiments and simulations reveal the sensitivity of demodulation performance and magnitude of impact each case has on OFDM signaling. In simulations, the maximum DC offset distortion permitted in the most sensitive modulation scheme probed is 0.15% for the case of 256 QAM, and ADC saturation due to DC offset has no obvious impacts for the values probed. Uncompensated gain error from 0.5% to 1% in received samples for 256QAM, results in SIR_{bit} decrease 6dB. Similar to the effect of gain error, SIR_{bit} degrades in the same manner for I/Q amplitude imbalances. In situations where resolution is less in the receiver relative to the transmitter, the SIR_{bit} performance degrades by 6dB for every bit decremented. Up to 10% I/Q delay caused by filtering results in approximately 0.5dB of variance in SIR_{bit} for all constellation sizes investigated. However, all these numerical figures are for 'isolated' cases, when factors are combined the results cannot be predicted by superposition of the 'isolation' data cases.

ACKNOWLEDGEMENTS

To begin, I would first like to thank my supervisor Professor Dr. Mohamed El-Tanany and Professor Dr. Rafik A. Goubran for their guidance, invaluable time, care, and support. Both individuals are quite exceptional, dedicating their talents to academics and research at Carleton University. As a true supervisor/teacher, they have shed just enough light to help me find the answers for myself in this thesis study. I would like to thank the National Sciences and Engineering Research Council of Canada (NSERC), Communications and Information Technology Ontario (CITO), and Carleton University for providing financial support for this research.

Also, I would like to express my gratitude to my parents, two younger brothers, all my uncles & aunts, and the numerous cousins for their love, positive influence, and encouragement.

Finally, I would like to thank all my wonderful friends, and other individuals, whom are responsible for aiding me in this fantastic journey, in any way they have contributed.

Education and knowledge is truly a glorious gift, and I am thankful for such an opportunity in life and the enlightenment it has rewarded me with, at Carleton University.

TABLE OF CONTENTS

List of Tables

List of Figures

List of Symbols

A	The peak signal level permitted by ADC converter (not peak-to-peak, which defines the input range allowable)
θ	Angular velocity
$C_{n,k}$	n^{th} QAM symbol transmitted on k^{th} carrier
d_{min}	Minimal distance between any two points in a constellation
DR	Dynamic range of converter
DVDD	Digital voltage drain drain
$\delta(t)$	Impulse in the time domain
Eb	Energy of a bit
Eb/No	Signal to noise ratio normalized to a bit of information with respect to constellation size.
E_j	Error between actual and quantized values during sampling
$\overline{E_j}$	Mean square error of Ej
f_{DAC}	Frequency of DAC clock
f_{ADC}	Frequency of ADC sampling rate
$f_{OFDM_{sym}}$	Frequency of OFDM symbols
$F(t)$	Time function
J	The number of binary bits per QAM symbol
f_k	The k^{th} orthogonal frequency OFDM carrier
$g_k(t)$	Complex time function
I	Average interference power
M	Number of points in M-ARY QAM scheme
MSE_{cst}	Mean Squared Error of constellation QAM points
MSE_{wave}	Mean Squared Error of time waveform
n	Number of resolution bits for the converter
N	FFT/IFFT size. Also equivalent to number of OFDM subcarriers
N_s	Number of OFDM subcarriers
N_{2a}	The noise from input clipping at a ADC
$\overline{N}$	Total mean square quantization noise error over a step
No	Average noise power
$OFDM_{sym}$	OFDM symbol, which consists of N OFDM samples
$OFDM_{sample}$	OFDM sample: In an OFDM modulator, each output of an N-point FFT is an OFDM sample.
q	Width of one ADC step
S	Average power of signal
$s(t)$	Time waveform
σ	Variance of the signal

σ^2	Expected mean of the signal
SIR	Signal-to-interference ratio
SIR_{bit}	Signal-to-interference ratio normalized to a bit with respect to constellation size
SNR	Signal to noise ratio
$S_{n\text{-}ideal\_RX}$	Ideal coordinates of n^{th} QAM point transmitted
$S_{n\text{-}ideal\_TX}$	Received coordinates of n^{th} QAM point that may include non-ideal analog distortion.
t	Continuous time index
T_{DAC}	Clock period DAC uses as reference
$T_{OFDM_{Sample}}$	Period of an OFDM sample
t_s	Beginning time for one OFDM symbol
T_s	QAM data symbol period
V_j	Quantized voltage level at jth step
V_{ideal}	Ideal voltage level at jth step
W_N	Twiddle factor for N point FFT/IFFT
$x(n)$	Inverse fourier discrete transform
x_{act}	Waveform received at the receiver
x_{ideal}	Ideal waveform the receiver could receive without distortion

List of Acronyms

A/D	ADC Analog to Digital Converter
AIC	Analog Interface Chip
ALU	Arithmetic Logic Units
API	Application Program Interface
ASIC	Application Specific Integrated Circuit
AWGN	All White Guassian Noise
BER	Bit Error Rate
BIOS	Basic Input/Output System
BPSK	Binary Phase Shift Key
CCS	Code Composer Studio
CPLD	Complex Programmable Logic Device
CSI	Channel State Information
CDMA	Code Division Multiple Access
D/A,	DAC Digital to Analog Converter
DC	Direct Current
DCO	DC Offset Compensation
DFT	Discrete Fourier Transform
DNL	Differential Non-linearity
DR	Data Received
DRR	Data Receive Register
DSK	Digital Signal Processing Starters Kit
DSL	Digital Subscriber Line
DSP	Digital Signal Processing
DVDD	Digital circuitry supply Voltage Drain-to-Drain
EDA	Electronic Development Automation
EDMA	Enhanced Direct Memory Access
EMIF	External Memory Interface
ENOB	Effective Number of Bits
EVM	Evaluation Module
FDM	Frequency Division Multiplex
FEC	Forward Error Correction
FIR	Finite Impluse Response
FLOPS	Million Floating-Points Operations per Second
FFT	Fast Fourier Transform
FSR	Full Scale Range
GUI	Graphic User Interface
HW	Hardware
I	Inphase
ICI	Intercarrier Interference
IF	Intermediate Frequency

IDE	Integrated Development Environment
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse Fast Fourier Transform
INL	Integral Non-linearity
ISI	Intersymbol Interference
ISR	Interrupt Service Routine
L1	Level 1
L2	Level 2
L1D	Level 1 Data
L1P	Level 1 Program
LAN	Local Area Network
LFSR	Linear Feedback Shift Register
LPF	Low Pass Filter
LSB	Least Significant Bit
LO	Local Oscillator
MB	Megabytes
MIPS	Million Instructions Per Second
MSE	Mean Square Error
MSPS	Million Samples Per Second
PAPR	Power Average-to-Peak Ratio
PaRAM	Parameter Table
PC	Personal Computer
PDA	Personal Digital Assistant
PER	Packet Error Rate
PLL	Phase Lock Loop
PN	Pseudo Random
Q	Quadrature
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Key
S/I	Signal-to-Image ratio
SDRAM	Synchronous Dynamic Random Access Memory
SICI	Signal-to-Intercarrier Interference
SW	Software
STS	Short Training Sequence
TTL	Transistor-Transistor Logic
TI	Texas Instrument
VLSI	Very Large-Scale Integration
VLIW	Very Long Instruction Word
WLAN	Wide Local Area Network

Chapter 1

Introduction

1.0 Thesis Objectives

In this thesis, the objective is to investigate various analog interfacing factors (interfacing with the digital domain) to have an impact on system performance in a typical orthogonal frequency division multiplexing (OFDM) transceiver. Analog factors of interest include I/Q amplitude imbalance, gain error at the receiver, reduced resolution in receiver relative to transmitter, ADC saturation, and relative I/Q phase delay factors caused by filtering in a bandwidth limited system. The impact on the performance of received signal at the demodulator is gauged by the figure of merit of mean squared error (MSE) of the waveform (MSE_{wave}), MSE of the constellation (MSE_{cst}), signal to interference ratio normalized to a bit (SIR_{bit}) figure of merits.

1.1 Literature Survey Overview

Before derivation of such a definitive thesis objective, a background survey is performed on published work within the area that deals with non-ideal analog factors influencing OFDM signaling. In summary of the survey, several studies share similar investigation

topics among each other and with this thesis, but differ in contribution value. Analog component interfacing issues covered in the open literature include the following factors: DC offset, effective number of bits in transceiver, I/Q imbalance of both phase and amplitude, which are caused by various process tolerances, or limitation in the analog components. To provide a quick primer on key studies, a few comments are made on selected studies that have the most relevance to this thesis.

The first study [37], has a close relationship to this thesis study by investigating the non-ideal properties of converters. Specifically, it looks into three of the four static errors which includes integral non-linearity (INL), differential non-linearity (DNL), DC offset[1] and excludes gain error by assuming the automatic gain controller (AGC) provides perfect compensation, which is not necessarily true. Also, the investigation covers the effects of the power amplifier with respect to the signal peak-to-average power ratio (PAPR). To characterize these factors, analysis of the devices in an OFDM system are performed in a simulation model based on WLAN 802.11a specifications. Complete details of the model are unclear, only listing several WLAN parameters to claim that it conforms to 802.11a standards using a fixed point DSP computation. In addition, the work includes investigation into the effective number of bit (ENOB) in the DAC/ADC converters on system performance. In its conclusion of converter static errors, it claims there is an insignificant impact of the INL and DNL errors on packet error rate (PER) performance. Also it observes no obvious impact from DC offset, but states a concern for potential reduced ENOB. As for ENOB, it concludes 10-bits in the converters provides significant PER performance gains and there is an optimal PAPR value to take advantage of the dynamic range of the available ENOB. PER performance gains beyond 10-bits of ENOB diminishes, as the improvements levels off into a plateau.

In support of the above AGC assumption to provide perfect gain, [5] discusses the general methodology on how the AGC is able to compensate for the gain error in an OFDM system including the dynamic channel attenuation. Thus the AGC has the ability

[1] INL, DNL, and offset error are defined and described in section 2.5 'DAC and ADC Converters.'

to compensate for static errors in DAC converters, but not in the ADC, because it trails the AGC in the receiver.

Among all factors in the literature survey, the analog factor receiving the greatest amount of attention relative to other analog interfacing issues is I/Q imbalances in both phase and amplitude. The imbalance effect is apparent when the amplitudes of I/Q channel are not equal and/or phase is not orthogonal (90degress) between the I and Q branches during the down conversion. Specifically, one source engendering I/Q imbalance source is found at the local oscillators (LO) driving the I and Q down conversion mixers. Ideally, they are 90 degrees out of phase and equal in drive amplitude. Also the mixers would be identical, but realistically have a process tolerance that causes mismatches. Thus non-ideal LO and mixer factors cause the I/Q imbalance. The net effect penalized for I/Q imbalance causes inter-carrier interference (ICI) causing cross talk between carriers.

Investigating I/Q imbalances in [36], the study proposes a solution to combat I/Q imbalances improving signaling performance. Its solution allows for relaxing of the demanding front-end specification in the receiver used for down conversion to reduce cost. The methodology partially compensates for the I/Q imbalances effects by the use of pilot symbols to estimate the correction factors to undo the imbalances based on an equation it has derived. Proof of performance enhancement is gauged by plotting BER versus Eb/No. The results are based on a BPSK scheme that both includes and excludes error control coding. Also, it examines the model's performance in different scenarios that include an AWGN channel and multipath channel environment. Both cases assume perfect channel state information (CSI). However, without CSI, long sequences of training symbol must be used, as simulations show I/Q imbalances have a significantly negative impact on channel estimation. To compensate for this flaw, it proposes estimation of I/Q effect on channel estimation corrections factors, before data symbols arrive. As before, it estimates the correction factors to minimize the I/Q imbalances based on an equation it has derived. Then is reapplies these correction factor to the data symbols that follow the training symbols. The correction factors on the I/Q parameters are static enough to be used on data for time compensation. In conclusion, the study

claims up to 10dB improvement for constellations as large as 64QAM using the suggested proposals.

In study [4], the paper investigates DC offset, I/Q amplitude imbalance, I/Q phase imbalances, and phase noise from the LO in a zero IF receiver. The motivation for the author to focus on IF receivers over superheterodyne systems is based on the potential for better feasibility in integration of components, potentially lowering cost. Acknowledging the non-ideal analog factors, the study suggests a digital compensation solution to relax the specification in the analog front-end of the receiver to operate with acceptable SNR degradation by meeting/exceeding the IEEE 802.11a SNR requirement, which requires better than 10^{-5} BER. In the first phase of the solution, it addresses zero IF structures high amplification requirements at the baseband that causes DC offset saturation problems. To correct, AGC and DC offset compensation (DOC) are performed based on received preamble during the acquisition, estimation, and compensation phase. The method suggested for DOC and AGC functionality is based on a digital front-end controlled AGC/DCO controller, which makes the proper adjustments during acquisition phase. Similar to other gain compensation solutions, it is not continuous in time. Exploring tradeoffs between preamble compensation and performance, a suggestion is made to reducing the samples used for estimation, saving the rest of the short training sequence (STS) for other tasks with only a small penalty in performance. Addressing the phase noise issue from the mixers, the study suggests improvement using an already existing technique known as 'Common Phase Noise correction per OFDM symbol', which operates on a symbol-by-symbol basis. To prove the suggested improvements made to the system, a simulation model based on WLAN standards is utilized. From the results, BER versus Eb/No data plots show the different compensation techniques combined to improve BER performance.

A very fundamental study done by [10], specifically analyzes the impact of I/Q imbalance on QPSK-OFDM-QAM scheme. In this paper, the author wishes to provide data to be used as a guideline for down converter designs. The concern of the I/Q imbalance in this study is addressed at the imperfect input levels and not having an ideal

90 degree phase difference between two LOs used to drive the mixer for down conversion on I and Q components. To investigate the effect, the simulation model used in the study features a noise free environment to purely focus on the effects of phase imbalance and amplitude imbalance both individually and in a combined scenario. Sensitivity of the system to the imbalance is characterized by BER performance for coherent QPSK system with respect to signal to intercarrier interference (SICI), which is another method to express the magnitude of I/Q imbalances.

Study [40] also addresses the same I/Q imbalance issues and DC offset from down conversion circuitry and attempts to compensate for the effect in a digital receiver. The DC offset error sources the paper is concerned with includes not only the receiver ADC, but also leakage from the LO in the down conversion mixing process. The study presents derived signal-to-image ratio (S/I) equations individually for both phase and amplitude mismatch, where image frequency noise is added due to the mismatch. To show a few numerical values it computes tolerable S/I limits with respect to the magnitude of mismatch, and specifically the limit value for WLAN IEEE802.11a system. This minimal limit requires an 802.11a receiver to have an S/I > 35dB. Wanting to minimize the impact of the two analog factors, the study proposes a solution to modify the standard digital IF receiver. Modifications include increasing sampling the IF signal 4 times higher followed by rectangular windowing of length 4 to cancel out the DC offset. However, due to poor filtering of the rectangular windows, phase mismatch is created and contributes to the already present amount in the signal. To compensate for this, training sequences are used to correct the phase, instead of using digital filtering for delay matching. This eliminates the potential for the digital filtering to alter channel response. In addition, frequency compensation is added to the receiver to help correct the phase and amplitude mismatch.

Unlike the studies already performed in this area, this thesis study provides more fundamental data on the impact on the baseband signal. Other works observe the impact on a very specific model or constellation scheme such as WLAN 802.11a. Instead, this thesis's goal is to strip away most of the system, such as coding, cyclic prefix, equalization, and acquisition, and provide data to be used in estimation of baseband

distortion and signal quality with respect to only these analog factors for 16, 64, and 256QAM constellation sizes. Specific issues addressed will include I/Q amplitude imbalances, system gain error, DC offset, ADC saturation, relative I/Q delay caused by filtering, and receiver resolution reduction. Relative to the studies found and discussed in this section, this thesis confirms DC offset factor in an OFDM system is not a critical issue, and it verifies the theoretical I/Q amplitude imbalance impact in terms of signal to inter-carrier interference degradation presented by [10] and [40]. In addition, the study addresses the concern made by [40] and [37] about reduced ENOB due to DC offset, but they do not perform any investigations on this subject.

1.2 Contributions

The contributions from the thesis studies on non-ideal analog factors are as follows:

- DC offset, I/Q amplitude imbalance, gain error, system resolution factors have been characterized in a hardware setup in terms of MSE_{wave}, MSE_{cst}, and SIR_{bit}. A software simulation model verifies and validates the presence of these non-ideal analog errors experienced in the hardware experimental system, with each error source magnitude quantified. In addition to the cross verification, the results provide insight to the MSE_{wave}, MSE_{cst}, and SIR_{bit} performance expected for the given magnitudes from each error source in this combined scenario.

- Several simulation have been configured to examine each of the following analog factors separately:

    - Gain error
    - I/Q amplitude imbalances
    - Reduced receiver resolution relative the transmitter
    - Relative channel delay caused by filtering

The results are plotted in term of MSE_{wave}, MSE_{cst}, and SIR_{bit} to gauge the OFDM receiver/demodulation performance. Each plot reveals the performance degradation trends and magnitude of impact of each isolated factor.

Also in the simulations, the effects of I/Q amplitude imbalances have been verified and are in agreement with a theoretical derived equation, found in [10], which predicts the impact of this non-ideal factor in terms of SIR.

- A simulation verifies that DC offset error without ADC saturation has no significant impact on demodulation performance, assuming the first carrier is not used for data

- By simulations, the impact of ADC saturation caused by DC offset error (assuming the first carrier is not used for data) or gain error does not have any significant impact on the SIR_{bit} for the magnitudes explored.

- By simulations, the results in a combination case that include:

    - DC offset
    - Gain Error
    - I/Q amplitude imbalances
    - Reduced receiver resolution relative the transmitter
    - Relative I/Q channel delay caused by filtering

prove non-ideal factors do not contribute to performance degradation in a linear manner from the data extracted in the individual simulations cases. Therefore when combined, the data from individual simulations should be used as an estimation guideline of what is the 'worst possible case' scenario.

1.3 Thesis Organization

The presentation of this thesis is divided into five main chapters.

In this first chapter, the thesis objective, related background literature material, and the contribution from the thesis work are presented.

The second chapter leads into background information on OFDM modulation, then OFDM systems, DAC/ADC converter errors, and finishes with information pertaining to the main hardware devices used in the experiments.

Chapter three discusses the details and objectives of different models used in simulations and hardware experiments. There are two main models described for the hardware experiments and software simulations. Unlike the hardware model, the software model is reconfigured in different configurations to isolate and/or combine different analog factors. This is done to fully characterize each non-ideal analog error contribution, and then demonstrate the non-linear error summation effect in a combinational case.

Chapter four presents the results from simulation and/or experiments done in each model. Empirical data results for the hardware experiments are compared to its software equivalent simulation for similarities in performance. Next, each analog factor is examined in software simulation and is cross-compared with results from different constellation sizes. A further complex simulation case, extending beyond the hardware coverage investigates delay factors and a comprehensive combination of different analog factors.

In chapter five, conclusions are stated with a summary of the key results. Also a section on suggestions for possible future works is included.

Chapter 2

Background Information

2.0 OFDM Introduction

To begin, orthogonal frequency division multiplexing (OFDM), is a modulation technique that has become a popular choice in many applications requiring large bandwidth for high data rate transmissions. Some examples of applications include xDSL modems, fixed wireless data transmission and 802.11 WLAN, which is becoming ubiquitous in mobile devices such as laptops and PDAs. Presently, wireless communication development is pushing OFDM application growth and research in this area. Generally, OFDM is used in high data rate applications to effectively take advantage of its multicarrier configuration. Advantages of OFDM include robustness against frequency selective fading (assuming error correction coding used), efficient spectral usage, and multi-path fading compensation by equalization (immune to narrowband interference). However some drawbacks of OFDM include sensitivity to frequency offset (loss of orthogonality) and phase noise from down conversion. In addition, its relative large peak-to-average power ratio reduces the power efficiency of the RF amplifier, an important factor in portable devices.

2.1 OFDM Modulation Theory

This section briefly reviews the fundamentals of OFDM modulation theory, which derived from sources [7] and [12]. The primary principle in OFDM modulation is to create parallel data streams a lower rate, where each stream is modulated on its own carrier. Each carrier, also termed 'sub-carrier', is mathematically orthogonal to each other in the frequency domain, allowing bandwidth spectral overlap between carriers without interference. As show in figure 2-1, spectral overlap permits more efficient use of bandwidth relative to conventional frequency division multiplexing (FDM) multicarrier techniques. Both techniques have potential to deliver the same data rate, but OFDM occupies less bandwidth.

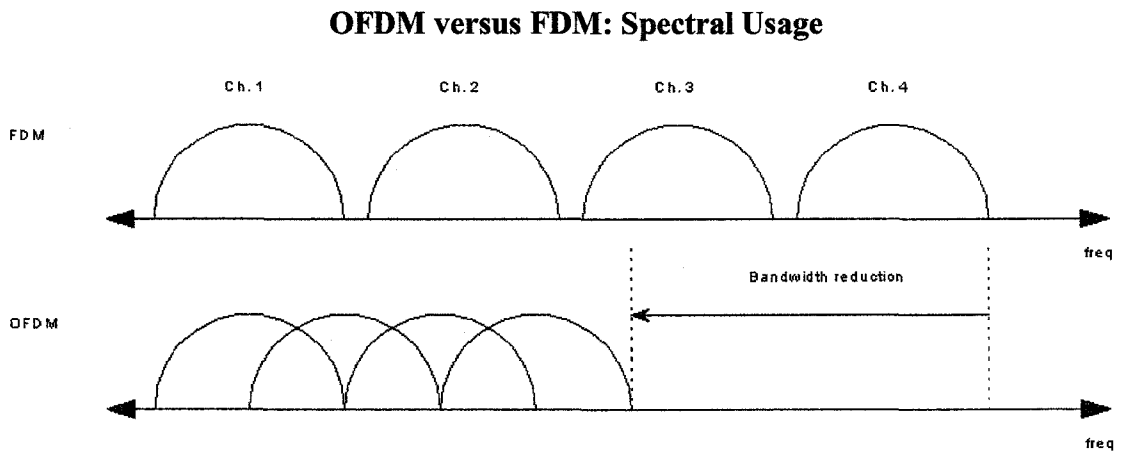


Figure 2-1. OFDM versus FDM: Spectral Usage

An infinite sequence of incoming data modulated to an OFDM signal can be represented by the equation:

$$s(t) = \sum_{n=-\infty}^{\infty} \left(\sum_{k=0}^{N-1} C_{n,k} g_k (t - nT_s) \right) \tag{2-1}$$

Where,

- $g_k(t) = \begin{cases} e^{j2\pi f_k t} & t \in [0, T_s) \\ 0 & otherwise \end{cases}$, Represents a complex phase shift with respect to frequency f_k over the symbol interval T_s

- $f_k = f_o + \frac{k}{T_s}, k = 0,1,2....N-1$ Represents orthogonal frequency components indexed by k

- $C_{n,k}$ is the n^{th} symbol, mapped from the constellation, and assigned to the k^{th} carrier

- N is the total number of carriers, and equivalent to N-Point FFT/IFFT taken to perform the modulation

The complex functions $\{g_k\}$ are orthogonal in the sense that:

$$\int g_k(t) * g_k^*(t) dt = T_s \cdot \delta(k-1) \tag{2-2}$$

To extract the n^{th} desired symbol with respect to the carrier, multiplication of the signal $s(t)$ with the complex conjugate value of $g_k(t)$, integrating, and normalizing as follows is performed:

$$c_{n,k} = \frac{1}{T_s} \cdot \int_{nT_s}^{(n+1)Ts} s(t) g_k^*(t) dt \tag{2-3}$$

Figures 2-2 and 2-3 are pictorials of a basic OFDM modulator and demodulator, as described by equations 2-1 and 2-3.

OFDM Modulator: Conceptual Schematic

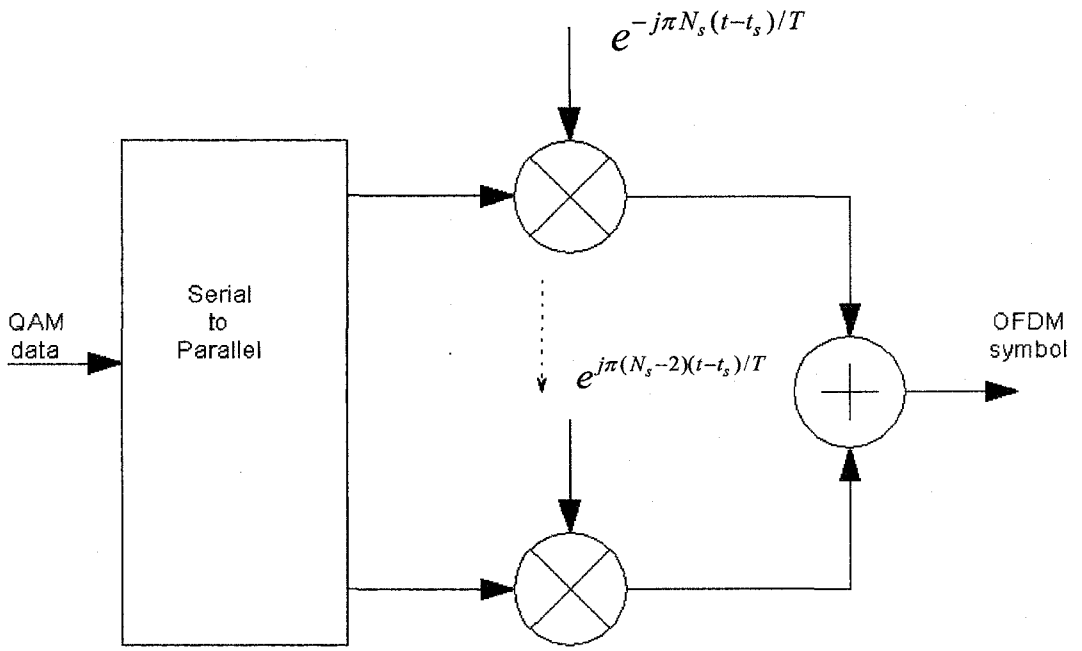


Figure 2-2. Conceptual OFDM Modulator

OFDM Demodulator: Conceptual Schematic

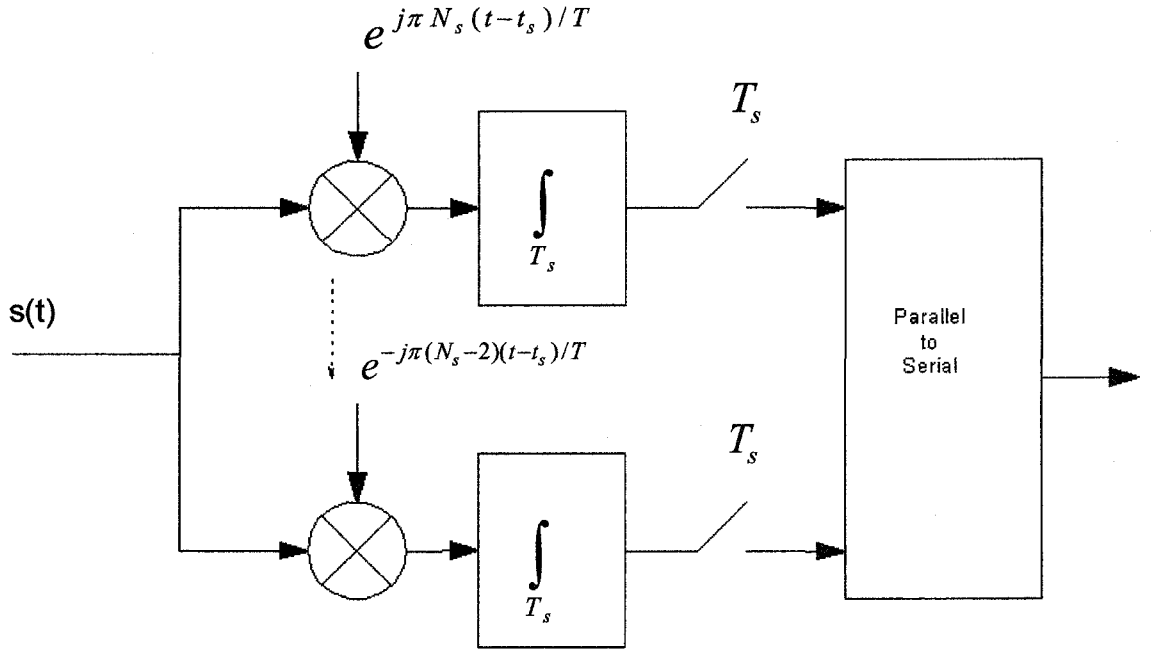


Figure 2-3. Conceptual OFDM Demodulator

Continuing to proceed into the discrete domain, the symbol and phase shift components from equation 2-1 are rewritten as a single OFDM symbol and defined as the n^{th} symbol transmitted in the n^{th} interval during a period T_s:

$$F_n(t) = \sum_{k=0}^{N-1} C_{n,k} g_k(t - nTs) \quad . \tag{2-4}$$

To convert (2-0) to a discrete format, sampling the low pass equivalent at N-times greater than $1/T_s$ gives:

$$F_n(m) = \sum_{k=0}^{N-1} C_{n,k} g_k(t - nT_s)\Big|_{t=(n+\frac{m}{N})T_s}, m = 0,1,2....N-1 \tag{2-5}$$

Substituting of $g_k(t)$ into (2-5) yields:

$$F_n(m) = e^{-j\pi(m+nN)} \cdot \left(\sum_{k=0}^{N-1} C_{n,k} e^{j2\pi k\frac{m}{N}} \right) \tag{2-6}$$

Further simplification of the expression assuming '$n*N$' is a positive integer value, simplifies results to:

$$F_n(m) = \pm(-1)^m \cdot \left(\sum_{k=0}^{N-1} C_{n,k} e^{j2\pi k\frac{m}{N}} \right) . \tag{2-7}$$

The inverse discrete Fourier Transform (IDFT) is defined as:

$$x(n) = \frac{1}{n} \sum_{k=0}^{N-1} X(K) W_N^{-nk}, 0 \le n \le N-1 . \tag{2-8}$$

Where the twiddle factor W_N is defined as:

$$W_N = e^{-j2\pi/N} \tag{2-9}$$

Comparison of equation 2-7 and 2-8 suggests that $F_n(m)$ is simply the IDFT of $\{C_{n,1}, C_{n,2}, C_{n,3}, C_{n,4}.....\}$. This simplifies the hardware required to perform the modulation since typical DSP processors can efficiently perform this function. To recover the data at the receiver, the Discrete Fourier Transform (DFT) can be applied to reverse the IFDT manipulation. The IFDT/DFT functions are key mathematical functions to simplification of the modulator/demodulator found in figures 2-2 and 2-3. Following the simplification, they are implemented in hardware using IFDT/DFT, as shown in figures 2-4 and 2-5, which is more computational efficient. Further details on the relationship between IFDT/DFT and IFFT/FFT are discussed in section 3.2.3 'DSP IFFT/FFT'.

OFDM Modulator: Hardware Implementation

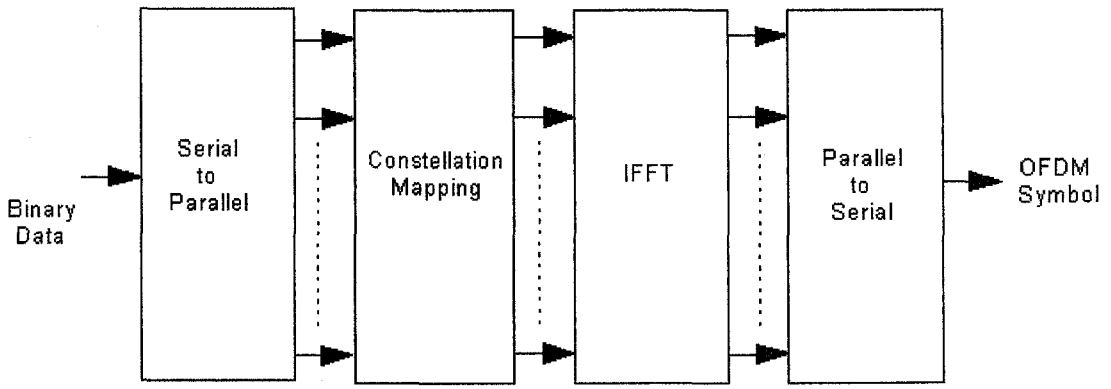


Figure 2-4. Implementation of OFDM Modulator in Hardware

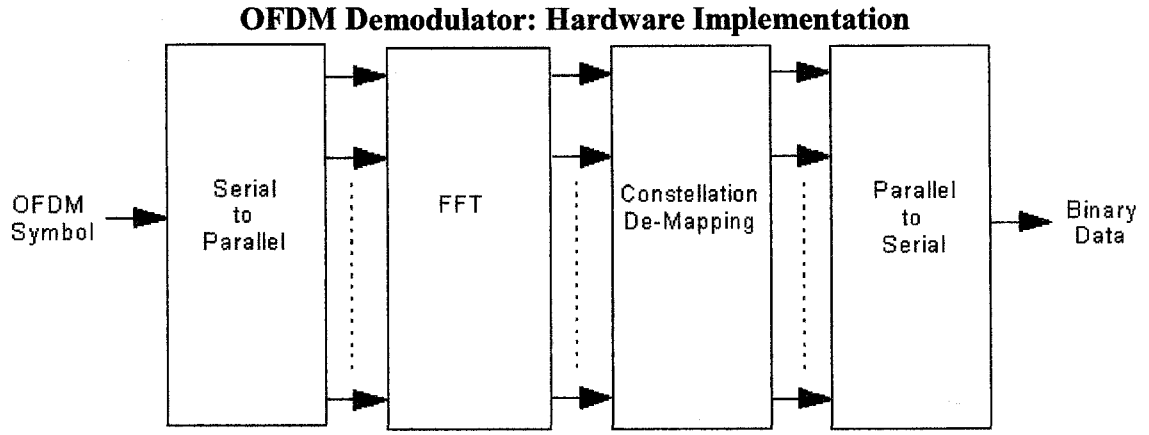


Figure 2-5. Implementation of OFDM Demodulator in Hardware

2.2 Typical OFDM systems

A general block diagram of an OFDM transceiver, shown in figure 2-6 presents some basic modules required for OFDM modulation/demodulation. To provide a working concept of the system, an introduction for the system blocks in the flow diagram is presented in this section.

2.2.0 Coding

Multipath fading is a common factor in radio channels that causes subcarriers to arrive at the receiver with below acceptable amplitudes. With a few faded subcarriers at the receiver, BER performance would substantially degrade, but forward-error correction (FEC) coding can provide immunity to losing data. If coding were applied across all subcarriers, then information loss due to weak carriers could be corrected, within certain limitations. Using this technique, the performance at the receiver would be determined by the average power received and not the power levels of the weakest carriers.

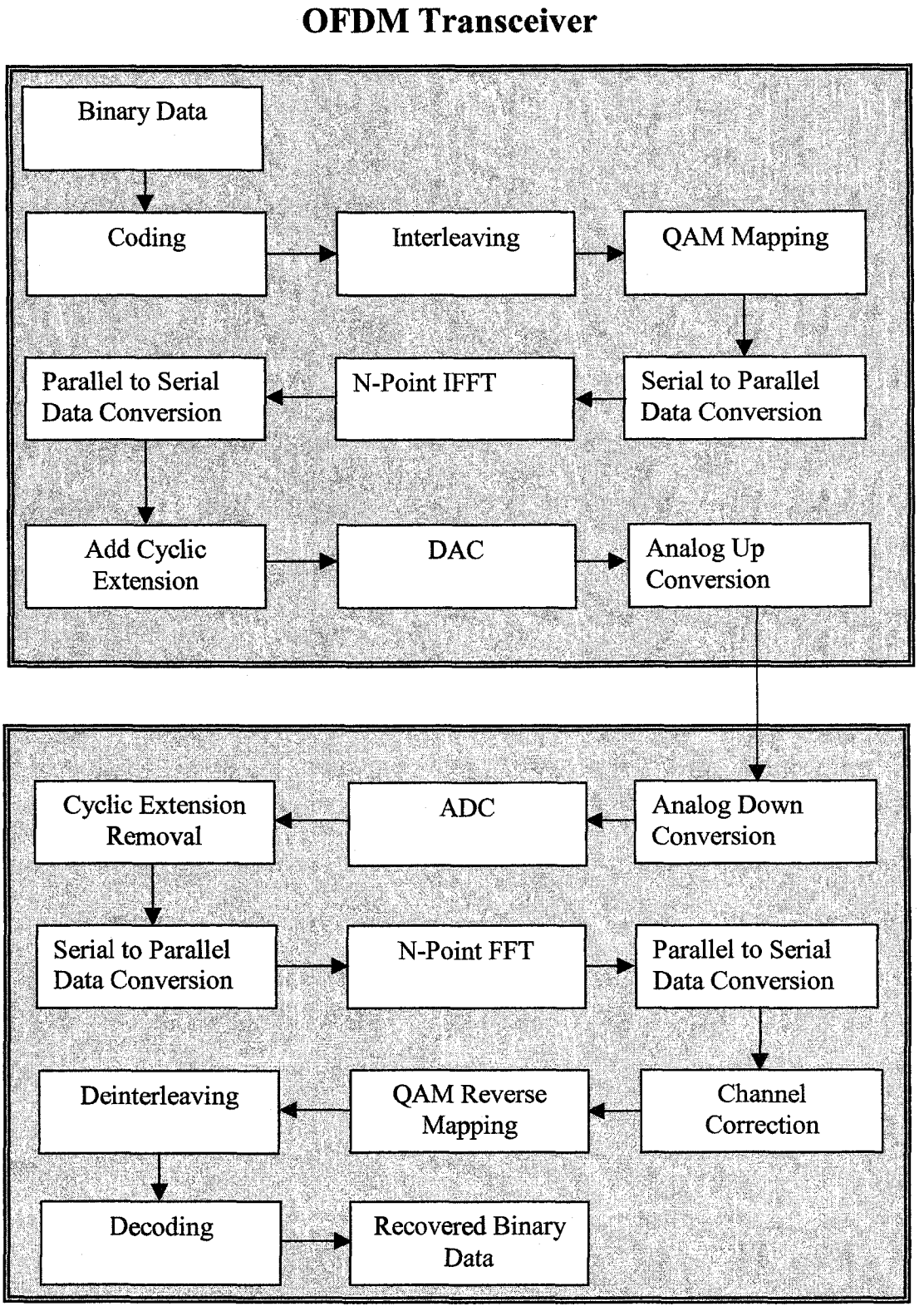


Figure 2-6. Typical OFDM Transceiver

Block codes and convolution codes are two categories of forward error correction methods. In block coding, a group of k input symbols are encoded into n output symbols, where $n > k$. The output symbols are larger because there is redundant information added to provide the receiver higher probability to properly correct any errors. Reed-Solomon codes is a popular block type code choice because its properties makes it relatively more effective for channels causing bursty error. This would be applicable to combating multipath fading error on a few selective subcarries as discussed previously.

Convolutional codes differ to block codes by their continuous input streaming and coding by convolution of data bits. For each input bit, a relationship is created with the previous input bits by convolution. Errors in the received data can be corrected to a certain degree using the relationship embedded with other neighboring bits. An example is using a Virterbi decoder that looks at the embedded relationship over a fixed time, calculating the hamming distance before deciding what the output should be. This method uses information encoded between data bits by the convolution process to help improve estimation of correct data sequence at the receiver. However, the receiver must delay the output upon reception of the incoming signal to view a segment of the data before deciding on the output.

2.2.1 Interleaving

Selective fading is common in wireless systems, and deep fades in the frequency spectrum may cause groups of subcarriers to be relatively less reliable than others. Thus, errors occur in bursts rather than being randomly scattered among the carriers. Interleaving is a method of spreading/rearranging the data across the carriers to make errors appear random in a selective fading environment. Block interleavers applied to binary bits or symbols is a common method to interlace the data. Interleaving symbols is useful with Reed-Solomon codes because the codes operate on symbols. Another option is to use a convolution interleaver, which relates data together in another manner.

In addition to frequency selective fading, radio channels also fade with respect to time. Radio channels tend to fade both in time and frequency. If the time-varying amplitudes and phases of all subcarriers were known at the receiver, then it would be possible to correct the fading with respect to time. This can be accomplished by inserting known values called pilots in a time varying manner across a few subcarriers.

2.2.2 QAM Mapping

Mapping converts fixed length binary data to a value on a constellation point. The length of the binary data depends on the available number of constellation points. For example, 64 points on a 64QAM constellation can represent all permutations of a 6-bit binary word (2^6=64 possible mappings). Larger constellations relative to smaller ones increase the data rate transmitted, but become generally more susceptible to noise if the average power transmitted does not increase. Also various constellations configurations for the same size do exist, each with a possible difference in BER performance.

2.2.3 Cyclic Prefix

To prevent intersymbol interference (ISI) in OFDM, a guard time is introduced for each symbol. The guard length should be selected to be longer than the largest delay spread value expected between the transmitter and receiver. With the guard-band, multipath is unlikely to cause any interference between any two symbols at the receiver because the symbols would ideally not overlap in time. The guard-band must also meet another criteria to avoid intercarrier interference (ICI). The guard length must be cyclically extended to ensure delayed replicas of the OFDM symbol always have an integer number of cycles within the FFT interval to keep its orthogonal properties between other carriers.

2.2.4 Serial-to-Parallel and Parallel-to-Serial Conversion

The serial-to-parallel conversion blocks are used to convert serial data into a parallel format to feed the IFFT and FFT functions, then to parallel-to-serial conversion is used to revert the data back to its serial stream.

2.2.5 IFFT and FFT

As previously discussed in the 'OFDM modulation' section, the IFFT routine is used to modulate the signal in the transmitter by mapping the QAM data to its corresponding carrier. At the receiver, the QAM data is recovered by performing reciprocal FFT function on the received samples.

2.2.6 DAC/ADC

The digital-to-analog and analog-to-digital converters (DAC and ADC) interface between digital and analog domains. Conversion to the analog domain is necessary because channel bandwidth is usually limited, thus conversion of the digital binary words into a corresponding output level is performed to create a time waveform. The conversion is not a prefect process due to non-ideal factors of converters, which are discussed later in this chapter in section 2.5 'DAC and ADC Converters.'

2.2.7 Analog Up/Down Conversion

Proceeding the DAC conversion to analog waveform, the OFDM signal undergoes up conversion to shift the spectrum of the signal centered around a higher frequency carrier, which maybe necessary to operate within a assigned channel bandwidth. At the receiver, down conversion is required to return the signal to baseband before filtering and re-sampling the analog waveform with ADCs, and then performing the demodulation. The analog hardware components used in the up/down conversion are plagued with inaccuracies. As an example, at the mixers in the down conversion there are non-ideal factors such as LO drive levels and phase orthogonality issues. Ideally the LOs driving the mixers for the I/Q branches have balanced drive levels, 90 degree phase difference and have no phase noise. However, this is impossible to realize and these factors must be considered in a system.

2.2.8 Demodulation

After down conversion, to recover the original data from the sampled signal at the receiver, the data must go through the reciprocal blocks found in the transmitter to demodulate the data. This begins with removal of the cyclic extension, parallel-and-serial data conversion, IFFT, serial-to-parallel conversion, de-interleaving, reverse mapping on the constellation, and error correction/decoding to recover the original binary data.

2.3 DSP Processors

Microprocessors specialized for digital signals such as voice and video, are commonly referred to as 'Digital Signal Processors' (DSP). They are optimized to perform common digital signal processing algorithms as quickly and efficiently as possible, often for real-time applications. DSP applications are ubiquitous in modern technology and can be found in wireless communication handsets, xDSL high speed modems, consumer audio, multimedia imaging, and automotive sensory, to list a few examples.

Specific applications, requiring higher sampling rates or geared for mass production may opt for application specific integrated circuits (ASIC), which are relatively more expensive and requires more time to design/develop than the widely available commercial DSPs. However, high volume ASIC production has the potential to reduce unit cost below the alternative commercial DSPs available.

In comparison to general microprocessors, DSPs are able to perform several memory access within a single instruction cycle, optimized to handle looping algorithms, allow for specialized circular addressing, and feature I/O interfacing optimized for specific external peripheral devices. Such external devices include timers, various memory types, external clocks, and duplex serial data port communication links.

As technology advances with time, DSP processors are getting faster, cheaper, smaller, and its power consumption is getting lower. However, the computational result of the DSP does not change for both floating-point and fixed-point processors.

2.4 DSP: Floating-Point versus Fixed-Point

In a system design that requires a DSP processor(s), a choice between floating-point and fixed-point processor(s) must be decided upon. There are tradeoffs between the two that must be examined carefully with respect to design criteria before a selection can be made. Table 2-1 summarizes the significant tradeoff factors between floating and fixed-point processors, which can be used to aid processor selection based on application requirements.

Table 2-1 Comparison Between Fixed and Floating-Point DSP Processors

Criteria	Fixed-Point Processors	Floating-Point Processors	Comments
Speed	√	X	Floating-point arithmetic generally reduces overall computational speed of the processor relative to fix-point arithmetic.
Precision	X	√	Computing with floating point values reduces loss of information by reducing risk of overflow and truncation errors due to higher precision and dynamic range.
Power Consumption	√	X	Additional circuitry required to support integer as well as floating-point arithmetic in floating point processors, thereby increasing power consumption.
Price	√	X	Tend to be cheaper because easier to design because of less circuitry/complexity.

*'√' – *Express advantage or better performance in contrast to 'x', which expresses the opposite.*

2.5 DAC and ADC Converters

Converters are sometimes referred to as interfacing circuits, because they act as the portal between digital words and analog waveforms. Commercially available converters are priced low enough to make consumer electronics affordable. Available resolution for commercially available devices typically range from 8-bits to 24-bits. Higher resolution is more desired because less information is lost when interfacing analog to digital domain and vice-versa, but usually has a greater cost.

The dynamic range of a converter is defined as:

$$DR = 20LOG_{10}\left(2^n\right)dB \tag{2-10}$$

Where n is the number of bits.

For typical floating-point processors, the output words most likely are a minimal of 32 bits, while fixed point would be at least 16-bits. If such a processor were to send data to a DAC with less resolution, data must be truncated and information is lost. In another scenario, a DAC transmitting data and received by an ADC ideally has matching resolution to minimize information loss. However, if the ADC has less resolution, then information is lost due to lack of precision needed to represent the DAC output waveform.

2.5.0 Sources of Errors

Errors causing inaccuracies in the converters are termed 'Static errors', and characterized using DC signals. Categories of static errors include offset, gain, integral non-linearity (INL) and differential non-linearity (DNL). Typical units for the errors are expressed in terms of least significant bit (LSB) units or a percentage of the full scale range (FSR).

The following subsections on converters discuss the details about each type of static error and help explain the specification details on the selected DAC used in this thesis.

2.5.1 Offset Error

For a given input range, converter offset error occurs when the actual output points shift (shown by 'Actual Diagram') by an offset from the ideal curve (shown by 'Ideal Diagram'), as demonstrated in figure 2-7. The actual offset value is measured at 0V (ADC) or all zero code (DAC). Trimming is a method of tuning certain components specifically specialized for adjustments in integrated circuits during manufacturing. This method enables devices to be tweaked to meet certain specifications. For DC offset error, trimming can be applied can to compensate for this error, but if it cannot, then the imminent error is referred to as the 'zero-scale' error. Although trimming is a initial solution, other factors such as degradation over time must be considered.

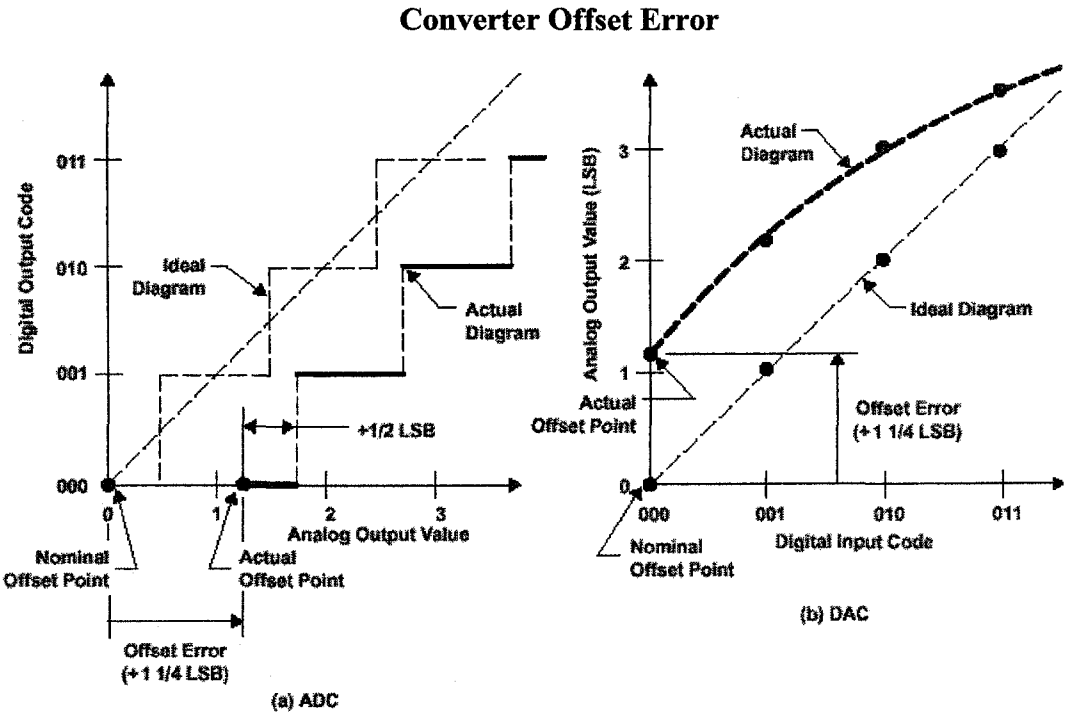


Figure 2-7 from [20]. Converter Offset Error.

2.5.2 Gain Error

Gain error is determined after correction on any offset error and is defined to be the difference between ideal output points and actual output points. To elaborate, analog input values to an ADC converter do not necessarily convert to the ideal output values. In an ADC converter, the location of the points used to determine gain error is when the output is at full scale and the input is at full scale for a DAC. Gain error is measured between the actual gain point and the nominal point at code: all ones (ADC output and DAC input). Please refer to figure 2-8 for illustrations. Similar to offset error, this phenomenon can be compensated by trimming.

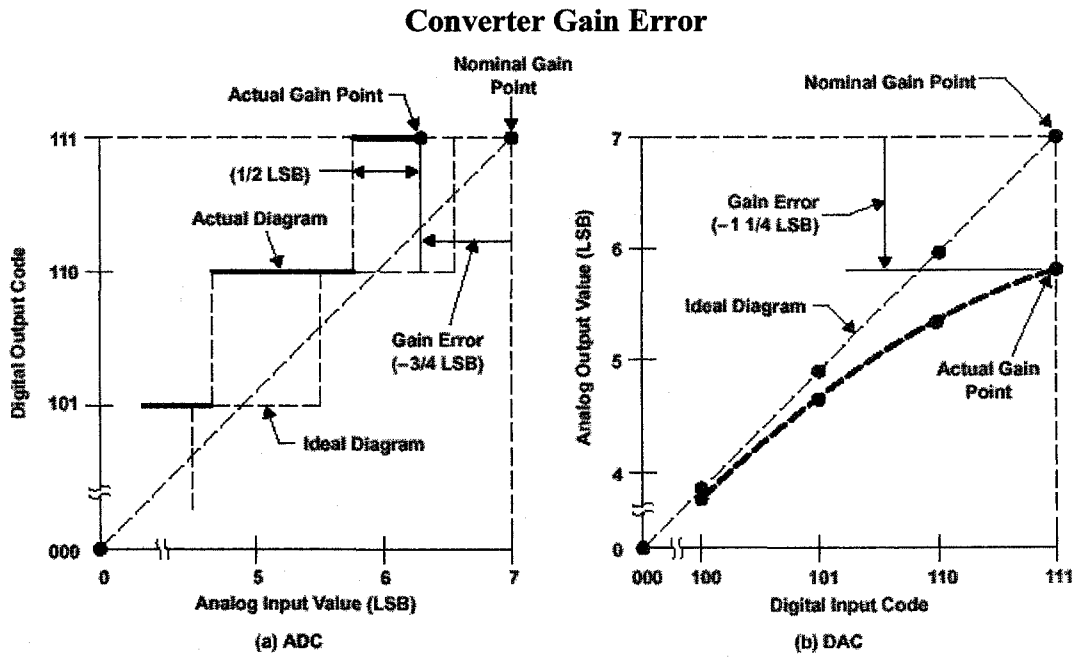


Figure 2-8 from [20]. Converter Gain Error.

2.5.3 Differential Non-linearity (DNL) Error

DNL exists when the step widths for the analog input for ADC (figure 2-9) or analog output code height for DAC (figure 2-10) is not a constant size of 1 LSB, but rather varies with respect to analog levels. In figure 2-9, the input range for a given output value varies in width and is not 1 LSB uniform across all input intervals.

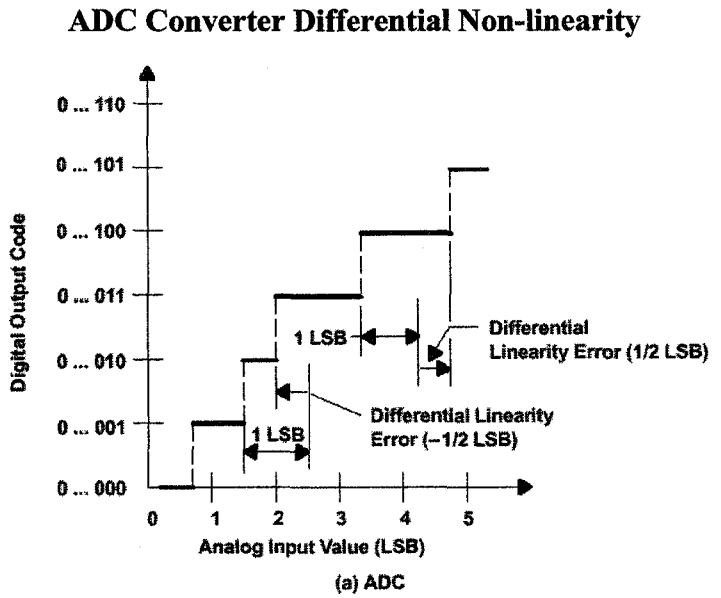


Figure 2-9 from [20]. ADC Converter Differential Non-linearity.

Figure 2-10, demonstrates adjacent DAC input codes are not uniformly spaced by 1 LSB at the output, rather it contracts across the input codes from left to right in the example given by the figure. The worst case in both the DAC and ADC is defined as the DNL error.

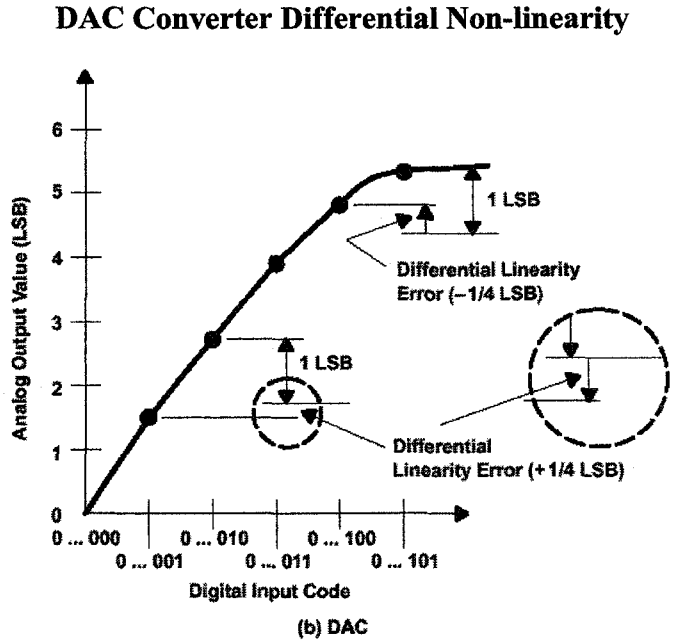


Figure 2-10 from [20]. DAC Converter Differential Non-linearity

2.5.4 Integral Non-linearity (INL) Error

With gain and offset error corrected, the ideal transfer line would pass end points on the curves for ADC and DAC as shown in figure 2-11. When integral non-linearity error occurs, causes step sizes vary between the two end points. For ADCs, figure 2-11 (a) shows the input intervals that translate to a specific output code do not have ideal uniform intervals between the possible input range. In figure 2-11 (b), the adjacent input codes, have varying output spacing, which expands and contracts from one end to the other, and unlike DNL the output end points are located at the ideal locations. INL is defined as the worst-case deviation between the two end points.

Converter Integral Non-linearity

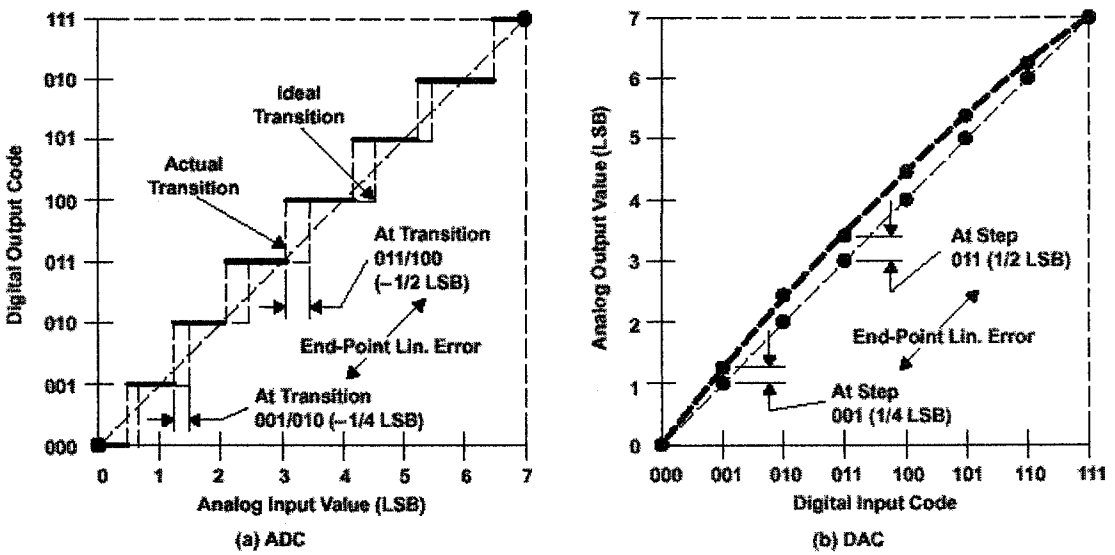


Figure 2-11 from [20]. Converter Integral Non-linearity

2.5.6 Absolute Accuracy Error

Absolute accuracy error is the total error combined from offset, gain, and INL errors. Also, it includes the quantization noise factor for ADCs. This error generally gives an overall estimate on how well the converter performs. Figure 2-12 illustrates how absolute accuracy error is quantified. The ADC the error value is observed at each mid-step, and is the maximum deviation between the ideal value and actual value, with respect to the

input levels. The absolute error for a DAC is observed at every transfer code point, looking at the deviation between the ideal transfer points versus the actual curve points, with respect to the output level. Absolute accuracy error in both the ADC and DAC is specified for the worst case.

Absolute Accuracy Error

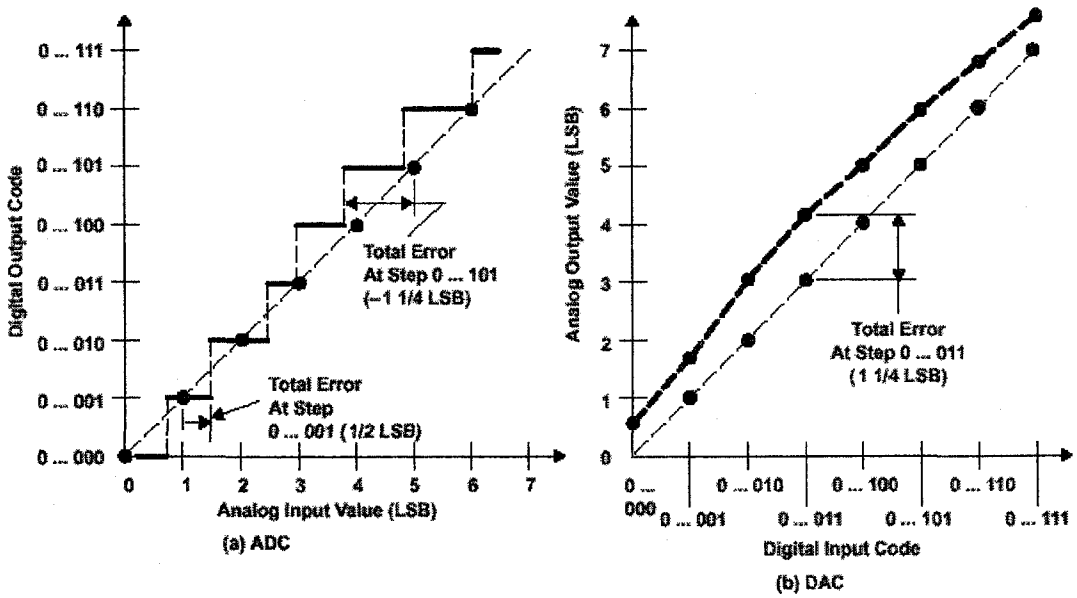


Figure 2-12 from [20]. Absolute Accuracy Error

2.5.5 Quantization Noise

Converting analog signals to the discrete digital domain always result in information loss, since a limited number of discrete levels are available to represent an infinite domain. The loss in information is referred to as quantization noise or errors, and cannot be reversed to recover the exact original signal.

To explain the effects of quantization noise of an ADC, mathematical equations are used to find average noise power and relate it to the signal-to-noise ratio (SNR). Figure 2-13 is used as reference to the variables used in the equation below. Assumptions in the derivations include:

- Midrise or Midtread Uniform quantizer used

- Symmetric quantization curve with respect to input and output voltage with the middle of the curve centered at zero volts.
- Input analog signal into quantizer has a zero mean
- Quantization error at output is uniformly distributed over a range of: $-q/2$ *to* $q/2$
- The input signal does not saturate the input of the ADC

Quantization Noise

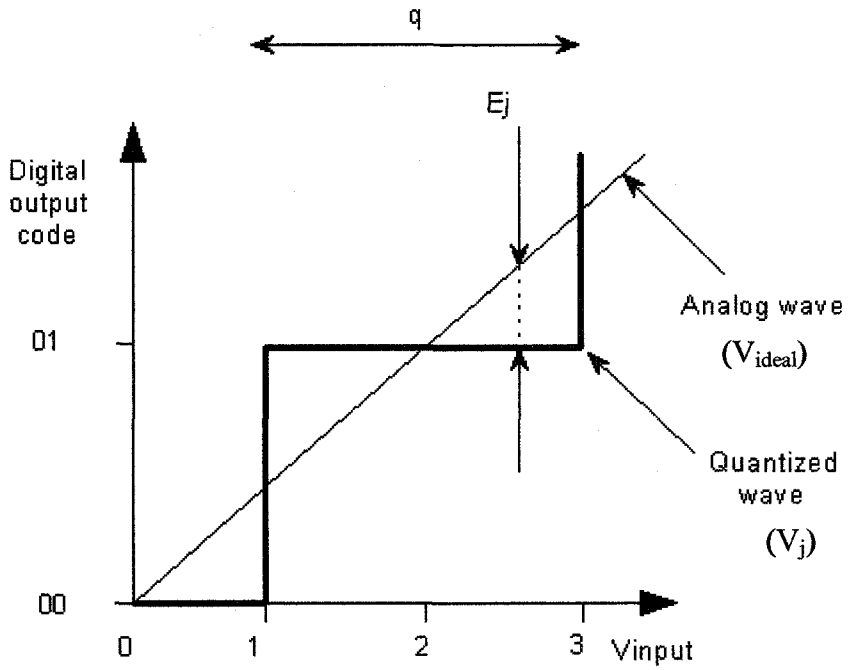


Figure 2-13. Illustration of Quantization Noise

To begin, referring to figure 2-13, the voltage error at the j^{th} step is:

$$Ej = Vj - V_{ideal} \ . \tag{2-11}$$

Where:

- j is the index of sample, j = 1,2,3…

The mean square error over width of the step is then:

$$\overline{Ej} = \frac{1}{q}\int_{-q/2}^{+q/2} Ej^2 dE \,. \tag{2-12}$$

$$\overline{Ej} = \frac{q^2}{12}$$

$$\overline{E}_j = \frac{(2A/2^n)^2}{12}$$

$$\overline{E}_j = \frac{A^2/2^{2n}}{3} \tag{2-13}$$

$$\overline{E}_j = \overline{N}^2 \tag{2-14}$$

Where:

- q represents the quantized step width. Refer to previous figure 2-13.
- A represents the peak in value of signal permitted by ADC converter and $2*A$ is the full input range permitted
- n is the number of resolution bits for the converter
- $\overline{N}$ is the average noise power

Using the average quantization noise power and average input signal power, an expression for the SNR can be found. As an example, assume the input to be a sinusoidal waveform in the form:

$$F(t) = A\sin\theta t \,. \tag{2-15}$$

Where:

- θ is the angular frequency in radians

The average power of the sinusoidal can be found by integrating over one period as follows:

$$\overline{F}^2(t) = \frac{1}{2\pi}\int_0^{2\pi} A^2 \sin^2(\theta t)dt \tag{2-16}$$

$$\overline{F}^2(t) = \frac{A^2}{2} \tag{2-17}$$

Then the signal to noise ratio in decibels is given by:

$$SNR = 10LOG(\frac{\overline{F}^2}{\overline{N}^2}) \tag{2-18}$$

$$SNR = 10LOG\left(\frac{A^2/2}{A^2/3\times 2^{2n}}\right)$$

$$SNR = 6.02n + 1.76dB \tag{2-19}$$

Equation 2-19 reveals the relationship between resolution and SNR of the received signal, showing a 6dB improvement/degradation for every 1-bit increment/decrement. The above derivation is based on [6] and [20].

If the input signal were Gaussian distributed, unlike the above derivation for a sinusoidal waveform, then the SNR approximation for $A>>Q/2$ (derivation taken fron [8]) is given by:

$$SNR \approx \frac{\sigma^2}{\left(q^2/12\right) + N_{2a}} \tag{2-20}$$

Where:

- σ^2 is the expected mean of the input signal to the ADC
- q the step size of quantizer given by: $q = \frac{2A}{2^N}$, for N bits of resolution
- A is the input peak amplitude limit allowed by the ADC; max peak-to-peak input to ADC is *2*A*
- N_{2a} is the noise from input saturation and clipping at the input of the ADC and it is defined by equation 2-21.

The noise from input saturation and clipping at the input of the ADC is given by:

$$N_{2a} = 2\left(\sigma^2 + A^2\right) \cdot \int_{L/\sigma}^{\infty} e^{-0.5v^2} dv - 2\sigma A e^{-0.5(A/\sigma)^2} \qquad (2\text{-}21)$$

Where:

- σ is the variance of the input Guassian distributed signal to the ADC
- $v = x/\sigma$
- x is the input Guassian signal
- A is the input peak amplitude limit allowed by the ADC; max peak-to-peak input to ADC is *2*A*

In equation 2-20, N_{2a} is the significant noise contributor in the denominator, which is caused by clipping/saturation of the input signal at the quantizer.

2.6 Hardware Overview

In this section, a quick background description with important specification on the main hardware components in the system is given.

2.6.0 TMS320C6711 DSP Processor

For the hardware experiments performed in this thesis, a floating-point processor is selected over a fixed-point processor to minimize computational quantization error. With various floating-point processors commercially available, a Texas Instruments (TI) TMS320C6711 processor, referred to C6711 for short, is selected and utilized with the available optional starter kit. This kit, referred to a DSKC6711, provides a support package that makes the C6711 ready to use for prototyping, evaluations, projects, etc. Included in this package, the DSP board is equipped with a parallel port interface for PC communications, external I/O ports from chips have interfacing connectors and components, 16MB of external SDRAM memory, power supply, onboard voice coder/decoder, audio jacks, and crystal clock oscillators are among other support components.

The C67X processor itself, features floating-point computation and has the ability to run at 100MHz and/or 150MHz. Being part of the C6x processor family from TI, the C67x is based on a very long instruction word (VLIW) architecture. Taking advantage of this architecture, by *parallelism* techniques, arithmetic logic units (ALUs) are able to support and process combined smaller data units contained within a single word, thereby increasing the number of calculations performed per instruction cycle.

The C67x series has the capacity to perform up to six floating-point instructions during a single clock cyle at 100MHz (10ns/cycle) or 150MHz (6.67ns/cycle). This equates to a possible 600 million floating-point operations per second (MFLOPS) performed at 100MHz and 900MFLOPS at 150MHz. Also, it can perform 2 multiplies and accumulates per instruction; 300Million multiplies and accumulates per second

The C6711 processor has circuitry to support various external peripherals including external timers, interface to external memory of various types, on chip PLLs, and two serial ports referred to as McBSP0 and McBSP1. A block diagram of the internal structure of the DSP is shown in figure 2-14.

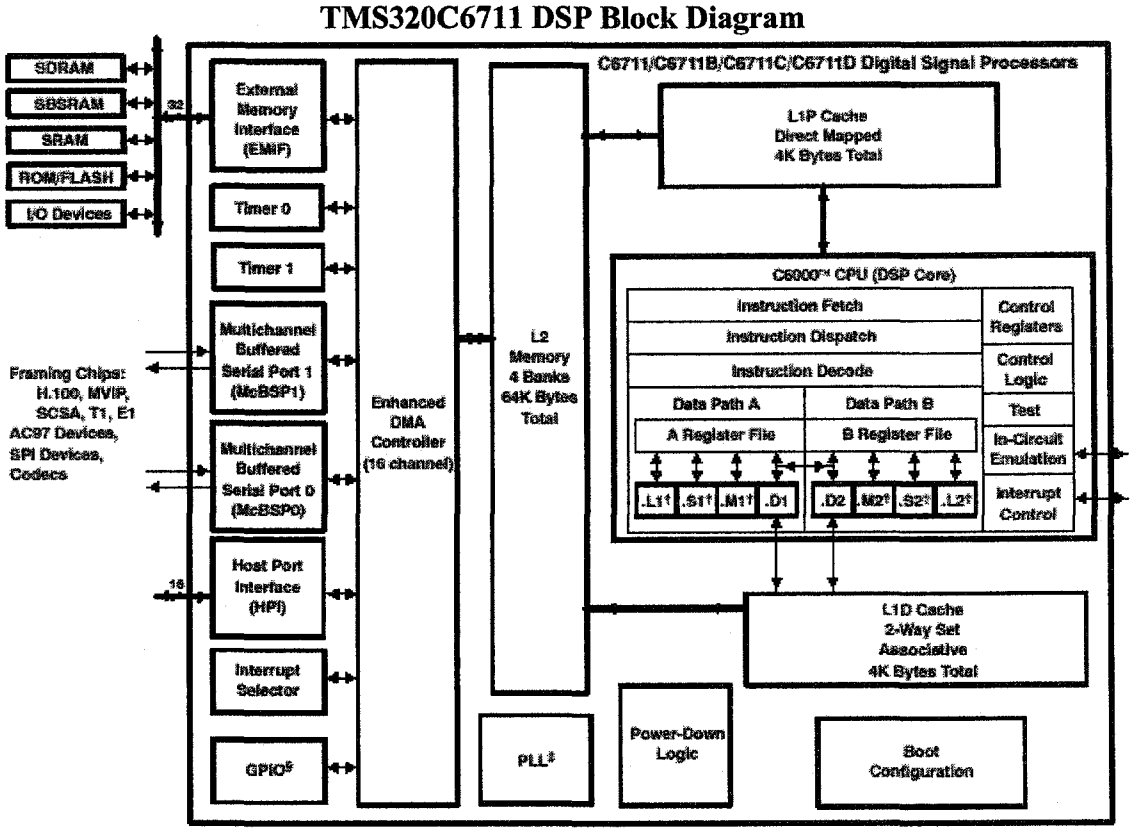


Figure 2-14 from [21]. TMS320C6711 DSP Block Diagram

The serial ports are commonly used to interface with analog interface chips (AICs), ADC, and DAC, as done in the thesis experiments. The two serial ports available on the C6711 can independently receive/transmit external digital data at rates up to 35.71Mbps, assuming specific configuration settings and proper external signal conditions are met, according to [35]. The ports have a double buffering interface, making it possible for data to be streamed in/out continuously. Timing used to orchestrate port operations may be derived internally from the CPU clock or referenced from an external source.

Enhanced direct memory access (EDMA) channels C6711 reduces processor load by acting as a mechanism to transport data between L2 cache and the peripheral devices with minimal consumption of the processing resources. On the C6711, 16 EDMA channels are each dedicated to a specific device. An EDMA transfer only interrupts the CPU

when ready for data transport or when reception of data has been completed. This allows the CPU to perform other tasks while the EDMA operates in the background versus the CPU actively polling the ports and/or performing the data transfer itself. Assigning priorities to EDMA channels enables the CPU to service requests of higher priority when simultaneous interrupts occur. Before compiling the program, configuration in the DSP/BIOS is necessary before using the EMDA channels. Options in the configuration include: size of memory elements, number of elements, number of frames, and array type transfers if applicable, which all must be specified. Overall, the channels are quite flexible and cater to different data transfer format needs.

Memory within the processor is limited to 8K in the L1 cache and 64K L2 cache. L1 cache is broken down to two parts: One 4KB of level 1 program cache (L1P) and another 4KB of level 1 data cache (L1D). L2 cache is used as either RAM and/or cache for data/program allocation. Memory in the L1 cache is used for frequent and quick access relative to L2. Less frequent accessed memory space, if required, must be provided by an external source through the external memory interface (EMIF). The EMIF require support from EDMA channels to transport the data between the L2 cache and external memory.

Hardware Interrupts (HWI) are high priority events the CPU must handle and manage to service requests from internal chip devices and external devices. They have a critical deadline and must be met with in a few instruction cycles to avoid other HWIs because only one HWI is serviced at a time. Thus data processing is not suitable in the interrupt service routines (ISR), where HWIs are handled. The interrupts are handled by triggering the CPU to go into an ISR written in C or assembly language. If it is written in C, a dispatcher setup is used because it allows the ISR to use the DSP/BIOS application program interface (API) that affects other DSP/BIOS objects and controls HWI enter and exit. HWIs can only be preempted by HWIs that have been enabled when the function HWI enter is called. A bitmask is passed to the function when called. This bitmask can be set up in the configuration tool when the dispatcher is enabled for the HWI.

Developing a program 'load' for the DSP/DSK is done in an integrated development environment (IDE) packaged together by TI. This software development studio called 'Code Composer Studio (CCS)' can be used for various processors offered by TI. At the heart of the studio, a compiler and optimizer for C & assembly are provided to translate the written 'C' language to machine instructions for the DSP and link all relevant libraries to the programs. The user interface of the studio is beyond text, and provides a complete graphic user interface (GUI) to using the software workspace and tools it offers. Some valuable tools CCS has to offer for program control, analysis, and debugging while the DSP is running include break points, watch windows, memory viewer, clock/instruction counter for code profiling, signal FFT analysis, and graphic variable slide bars.

2.6.1 THS56X1EVM Board and THS5661 DAC Chips

The DACs used in the experiments, model THS5661, is a product designed by TI. For quick implementation with the hardware system in the thesis, the units are ready for quick use because it is already packaged with THS56X1EVM evaluation boards. Using the evaluation module (EVM), versus constructing a support PCB for the DAC chips, simplifies hardware development and allows more focus on evaluation of the devices, research, prototyping, etc. Another benefit of the EVM unit is the ease of configuration/reconfiguration of the DAC by providing onboard jumper circuitry to switch between various modes of operation. In table 2-2, a summary of the important specifications is tabulated for the DAC and figure 2-15 is a system block diagram of the EVM that includes the DAC.

The DAC chip itself, has on-chip 1.2V temperature-compensated bandgap reference, thus eliminating the need for a stable external reference voltage source, unless another voltage value is desired. This stable 1.2Volts provides the DAC chip with the V_{ref} used to scale the output levels of the DAC.

Table 2-2. Table of Key Specifications for THS5661 DAC

Specifications	Min	Typical	Max	Units
Resolution	12-bit	-	-	**Bits**
INL	-4	+/- 0.75	4	**LSB**
DNL	-2	+/-0.75	2	**LSB**
Offset Error	-	0.02	-	**%FSR**
Gain Error	-	1.3		**%FSR**
Full scale current	2	-	20	**mA**
Reference Voltage	0.1	-	1.25	**V**
Maximum Sampling rate	100/67	-	-	**MSPS**
Output propagation delay	-	1	-	**ns**

The external clock feeding the DAC activates latching of the input word upon the rising edge of the clock edge. With a well-conditioned clock supplying the DAC, it can operate at 100MSPS with 5DVDD[2] or 67MSPS with 3.3DVDD power-supply to the digital circuitry. A second 5V supply is necessary to supply the analog circuitry. With the separate supplies, noise is isolated between the noisy digital and noise-sensitive analog circuits. When passing a signal through the DAC, there is significant delay between the time when the input word is entered, till the time it reaches the output. This delay is over one DAC clock cycle, but the delay difference between two different DAC chip is less than or equal to 1ns. Therefore, between two DACs or more, the probable worst case in relative delay is not of significance to the experiments in this thesis.

[2] Digital supply voltage drain to drain (DVDD)

THS56X1 EVM Block Diagram

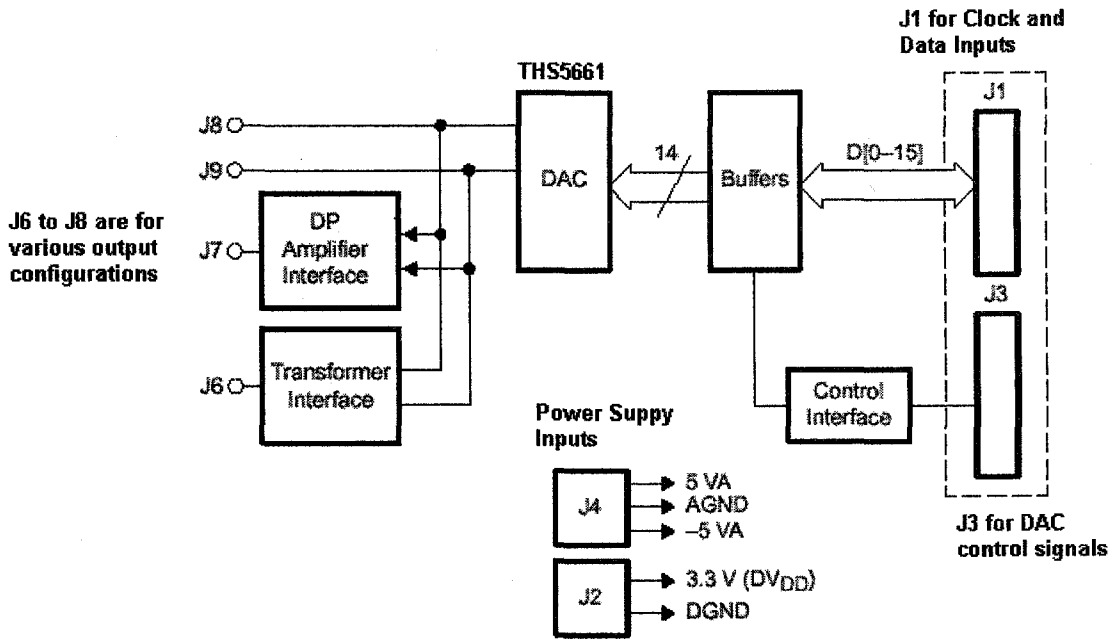


Figure 2-15 from [22]. THS56X1 EVM Block Diagram

Input words to the device can be configured for either binary or two's complement format and the bits must be fed in parallel versus serial.

This particular DAC model has 0.02% FSR offset error and the gain error is 1.3% of FSR, which maybe of concern to causing OFDM waveform distortion that translates into demodulation performance degradation. These two factors are further discussed in section 3.2.8 'Transmitting OFDM Symbols Out Through I/Q DACs.'

Chapter 3

System Models

3.0 Overview

In this thesis, there is a hardware model to represent hardware configuration used for experiments performed and a software model to represent the software setup for the simulations executed. The software model can be configured to be equivalent to the hardware model, as done in this thesis to cross-verify the hardware results. In this chapter, the model blocks for each model are presented and the details of each block are discussed.

3.1 Hardware Model Introduction

By performing hardware experiments, empirical data is collected and carefully analyzed to observe the combined effect of DC offset, gain error, I/Q amplitude imbalance, and DAC/ADC resolution. The MSE_{wave}, MSE_{cst} and SIR_{bit} [3] are used to quantify the combined impact of these factors on the performance of the experimental system. Different hardware devices are used in the setup as suggested by the model of figure 3-1.

[3] MSE_{wave}, MSE_{cst} and SIR_{bit} are defined in section 3.7 'Figure of Merits'.

Model 1: Hardware Flow Diagram

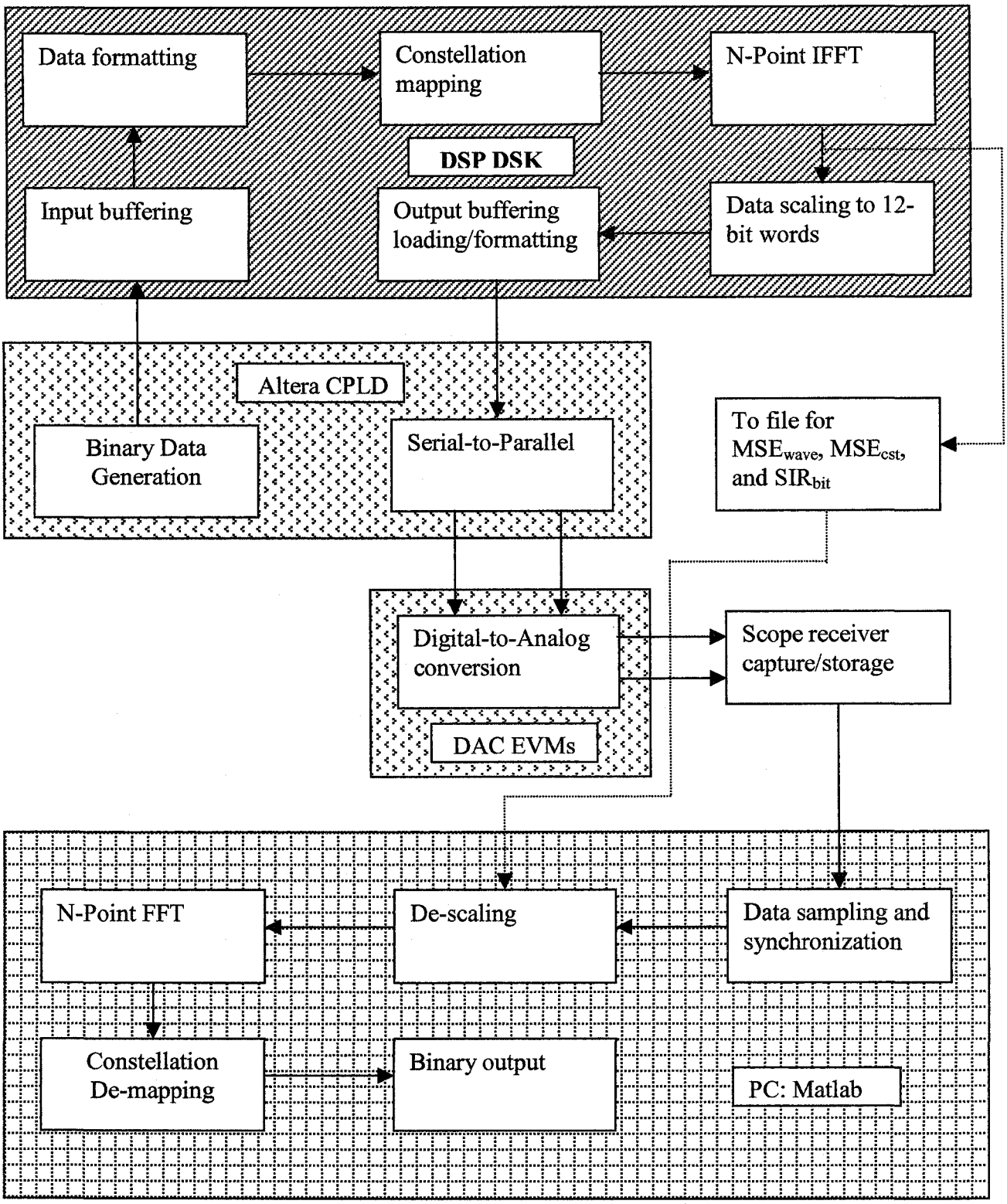


Figure 3-1. Hardware Model Diagram

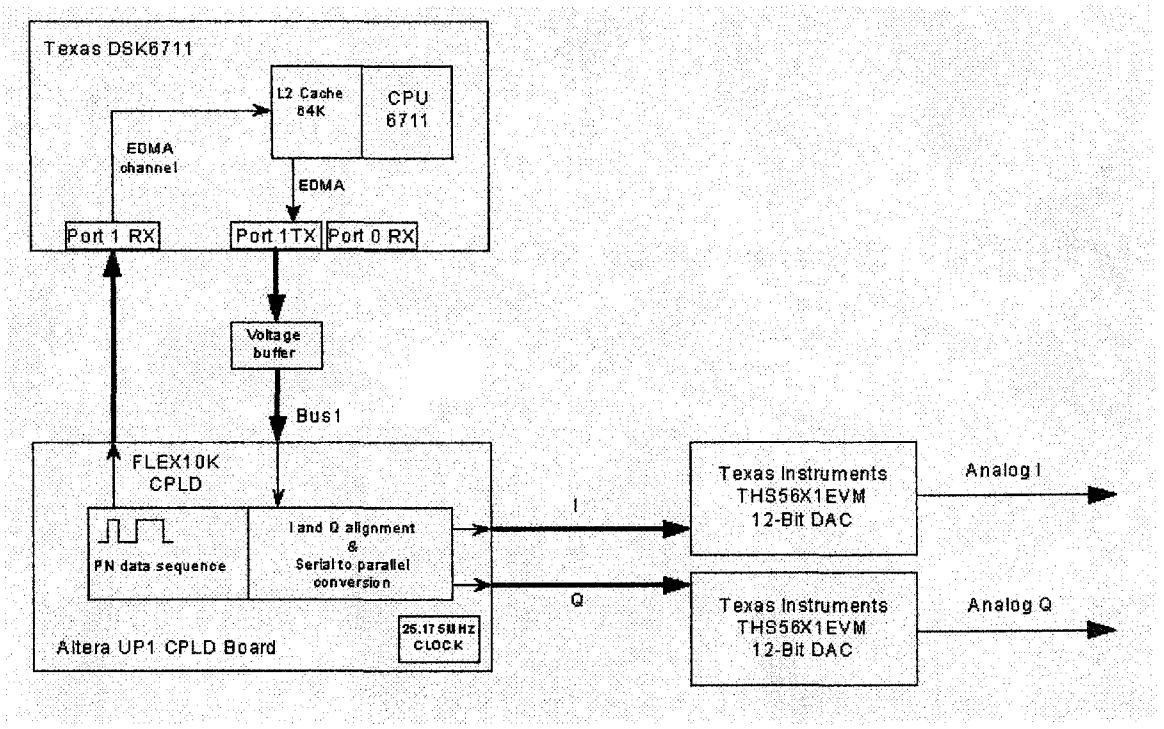
This hardware model diagram provides flow map between the devices used to perform the modulation and demodulation, while figure 3-2 provides a structural illustration of the modulator part of the system. The model shown in figure 3-2 is composed of three main components:

1. A CPLD is used for PN data sequence generation, and to provide essential serial-to-parallel data interfacing between the DSP and DACs.
2. A floating-point DSP used to carry out OFDM modulation operation, which includes performing the data constellation mapping and FFT.
3. Two high-speed Texas Instruments 12-bit digital-to-analog converters (DAC) are used to convert the output of the modulated data from the DSP to its equivalent analog OFDM waveform .

In the receive end, data is captured by an ADC embedded inside a HP54542A storage scope[4]. This data is then processed and demodulated for analysis in Matlab software, running off a PC platform. The hardware model specification values are summarized and tabulated in table 3-1. They include incoming and outgoing data rates of the modulator and the amount of binary data it processes. Other important values include the DAC output swing and resolution of the ADC/DAC, which are the focus of the experiments. Further details of each block in the model figure are described the following sections from 3.2 to 3.4.

[4] HP54542A storage scope details are found in section 3.2.9 'ADC Sampling at Receiver Input'

OFDM Modulation Hardware Structural Diagram



**Analog I/Q are feed to scope, which is not shown.*

Figure 3-2. Structural Hardware Diagram of Modulation Section Only

3.2 Hardware Model Block Descriptions

3.2.0 Pseudo Random Data Generation

In the hardware system, high-level language 'Verilog' procedural code is written to describe to the synthesizer a linear feedback shift register (LFSR) configuration, which is responsible for generating a pseudo random (PN) binary data sequence. This data sequence simulates data in a communications system in need of transport over a data link. An example 4th order LFSR is shown in figure 3-3. The LFSR generation polynomial for this example is given by: $x^4 + x + 1$. In this example, every clock cycle data is shifted from left to right through the flip-flops and looped back at 'x^0' to 'x^4'.

Table 3-1. Hardware Simulation Model Summary

Item	Value	Comments
Net PN sequence length	7680	Half of sequence generated from 12^{th} order LFSR, other half from 13^{th} order LFSR. Hardware systems requires a full reset to migrate between sequences.
CPLD to DSP receive rate	25.175 Mbps	CPLD supplies master clock
N-IFFT/FFT	64	Equates to 64 orthogonal carriers
Constellation	64QAM	▪ 6-bits per symbol ▪ Rectangular constellation configuration
Total number of OFDM symbols transmitted and captured per data set.	20	Number of OFDM symbols created from the available PN sequence. Two trials[5] are required to reach 20 symbols because of PN generator reconfiguration necessary.
OFDM Symbol rate	8.19Ksym/sec	▪ Maximum rate possible using a single serial port with 25.175MHz clock ▪ Assumes all 64 OFDM samples carry data
Bits per OFDM symbol	384 bits/sym	
DAC resolution	12-bits	DAC in binary mode
DAC output swing	4Vpp	Calibrated on scope
ADC Resolution	8-bits over eight 1V divisions	Fixed
ADC sampling rate	25 MSPS	These values allow for capture all 10 OFDM symbols plus one all zero symbol to be used as time maker in demodulation.
Over-sampling ratio (ADC sampling rate over data rate from DAC)	23.83	
Receiver/Demodulation DAC voltage anticipation	4Vpp	If this does not match the actual DAC output swing a mismatch causes gain errors.

[5] A 'trial' is a set of data collected per specific LFSR configuration on the CPLD; each LFSR configuration on the CPLD is used to generate 10 OFDM symbols. A new 'trial' is made when the hardware system is restarted.

Depending on the equation for the PN sequence, exclusive-OR (XOR) gates must be inserted in the feedback loop for every 'x' with an exponential integer value from {n-1,n-2,n-3.....2}; this excludes the highest order variable x^n and lowest x^0. For example, equation $x^4 + x + 1$ describes a 4th order LFSR, where a XOR gate is required at the conjunction of the 'x' output and the feedback path. Outputs for the binary PN sequence can be taken at any flip-flop location from x^3 to x^0 as illustrated in figure 3-3.

Example of a 4th Order Linear Feedback Shift Register

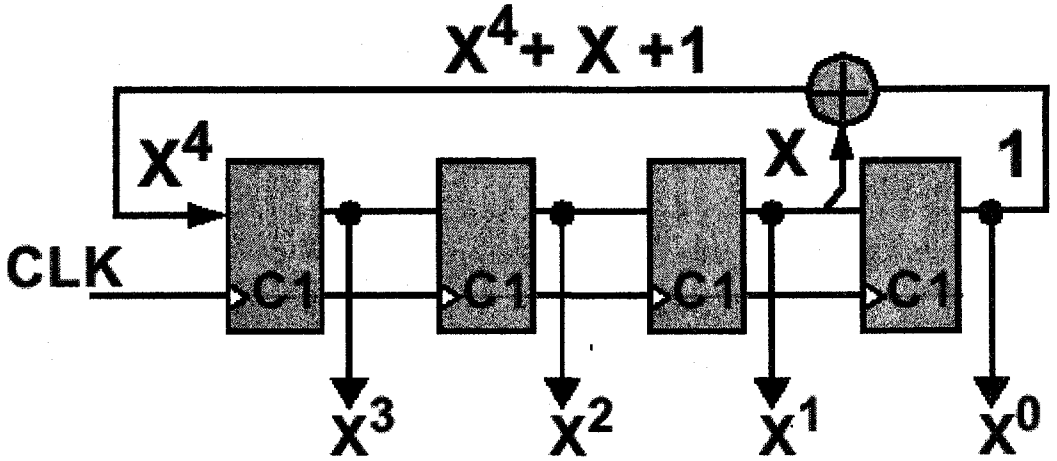


Figure 3-3 from [9]. Illustration of a LFSR

Table 3-2. **Summary Table of PN Generator used for Hardware Experiments**

LFSR n^{th} order	Polynomial equation[6]	Total PN Bits available	Total Used
12^{th}	$x^{12} + x^7 + x^4 + x^3 + 1$	4095	3840
13^{th}	$x^{13} + x^4 + x^3 + x + 1$	8191	3840

Since the DSP can receive 128 words per frame (A 'frame' is a group of words) received, each word comprises of 32-bits, then a 12th order LFSR is the minimal order to deliver

[6] Equations are taken from [9]

enough data for 10 OFDM symbols given the modulation specifications. To provide more experimental data, the DSP is restarted, and the CPLD is re-configured with a 13^{th} order LFSR to generate another data sequence for another distinct set of 10 OFDM symbols for the experiments. Although the 13^{th} order polynomial is selected, other polynomials of 12^{th} order or greater can be used. A summary of the two LFSRs configured in the CPLD is presented in table 3-2. Later on, in the software simulation model section, the equivalent software simulation model uses the same LFSR configuration to match the PN sequence.

3.2.1 Binary Data Transfer from the CPLD to the DSP

Transferring the data from the CPLD into the DSP's serial port requires proper timing and synchronization between the two devices. In this data transfer, the PN binary data sequence from the CPLD continuously streams into the DSP, but the DSP only accepts the data given proper synchronization from an external or internal source. In this setup, the CPLD provides the DSP serial port with a synchronization pulse for the transfer of every PN cycle and the master clock signal for the reference timing. The beginning of each PN cycle is identified by comparator logic block in the CPLD when a specific LFSR binary sequence is selected after reset state. In this thesis, the binary sequences used for synchronization pulse generation are:

- 12^{th} Order LFSR: 000 000 000 010 (x^{11} to x^0)
- 13^{th} Order LFSR: 0 000 000 000 010 (x^{12} to x^0)

When the LFSR is filled with the above sequence, a pulse is generated on the next clock cycle and sent out the CPLD for synchronization with the binary data output. At the DSP, the binary data sequence is not captured by the serial port until arrival of the synchronize pulse, which triggers the port to begin reception. Any binary sequence in the LFSR is valid except for all zeros, since this sequence never occurs. Figure 3-4 is a simulation of the 13^{th} order LFSR in the CPLD, showing the synchronization pulse generated ('synch') and the data generated ('regan1') and sent out the CPLD. The 'synch' pulse is regenerated after the PN sequence for 'regan1' repeats, which is not shown.

Timing Diagram for 13th Order LFSR Simulation in Altera CPLD

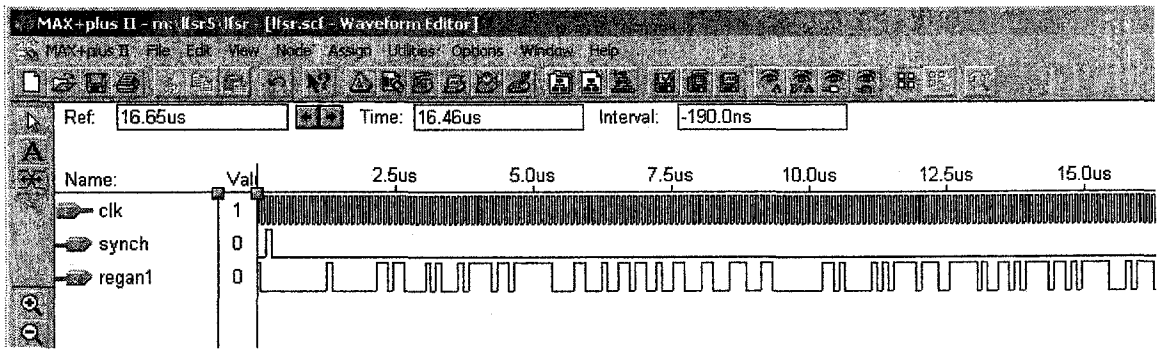
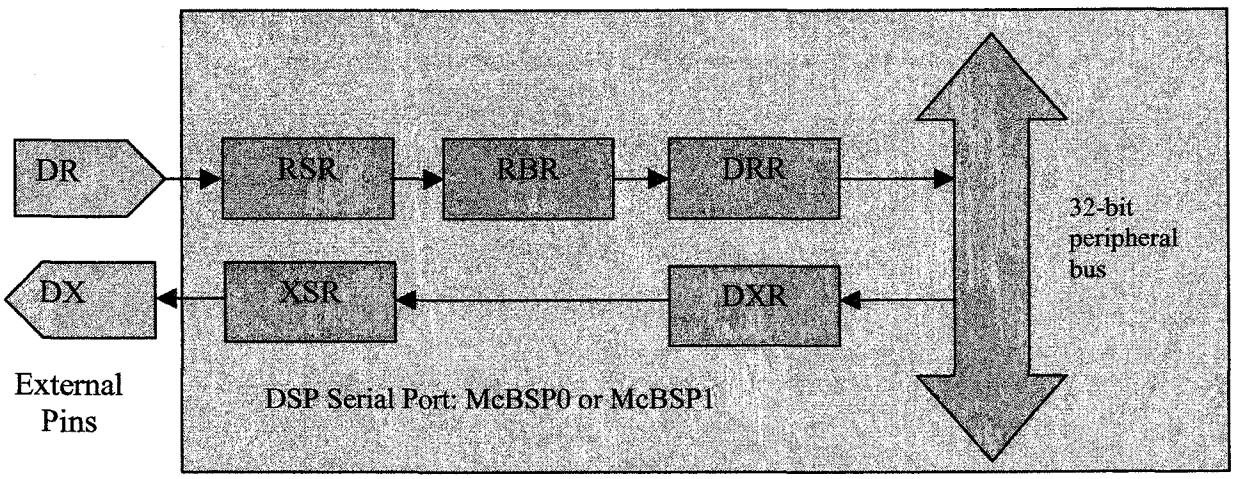


Figure 3-4. CPLD Timing diagram for PN Data Generation and Synchronization Pulse

synch – the synchronization pulse generated every data cycle in 'regan1'

regan1 – serial data output taken from a 13th order LFSR used in thesis at location x^1

DSP - Texas Instruments C6711: Serial Port Data Interface



Acronyms

DR – Data received pin
RSR – Received shift register
RBR – Receive buffer register
DRR – Data receive register
DX – Data transmit pin
XSR – Transmit serial register
DXR- Data Transmit register

Figure 3-5. DSP - Texas Instruments C6711: Serial Port Data Interface

Refer to figure 3-5 for the following discussion on data flow into the serial port. Binary data streaming into the serial port receive register pin (DR) passes through two buffers (RSR & RBR), then grouped into 32-bit words in the data receive register (DRR). Once the DRR is filled up, the serial port indicates a 'full' signal to the EDMA, since the transfer is configured to use the EMDA channels. In response, the data is then emptied by the EDMA channel and transported to an incoming buffer array located in the L2 cache as illustrated by the 'primary buffer' in figure 3-6.

Data Arriving to L2 Cache Input Buffer from EDMA

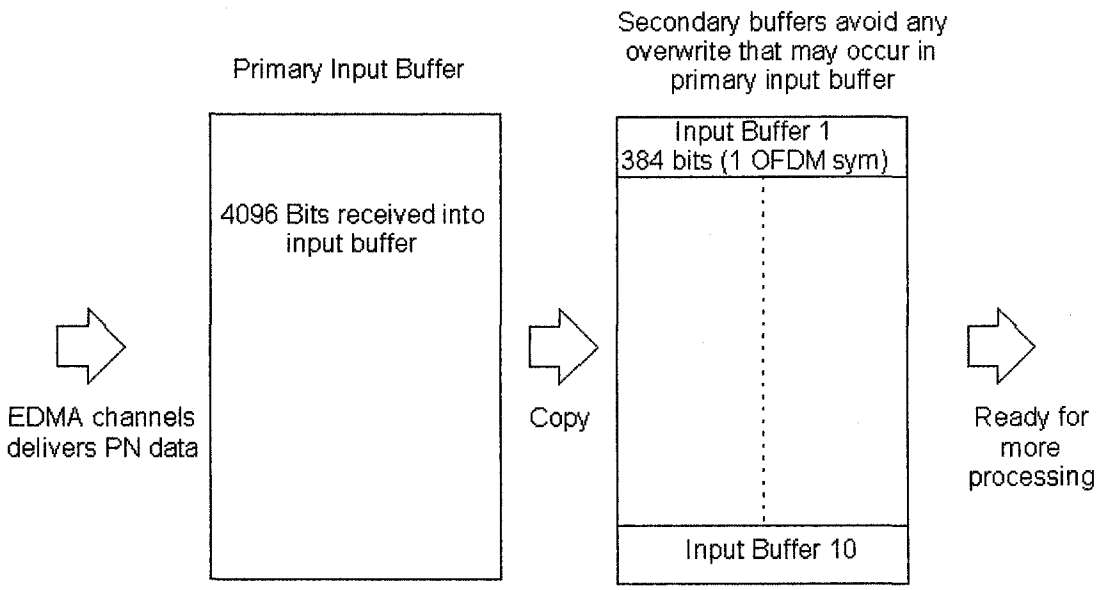


Figure 3-6. Data Arriving to L2 Cache Input Buffer from EDMA

Upon completion of data reception, the primary input buffer array will have filled up. To avoid being overwritten over with the same sequence, since the duplex McBSP is still active for data reception, the buffer is copied into smaller secondary buffers. Each secondary buffer stores the data in an equally partitioned memory segment, dividing the data up as illustrated in figure 3-6. This organizes the data for processing by grouping data bits such that an OFDM symbol is created from each data group. Once the data is

copied over to the secondary buffers, the CPU will begin processing the secondary buffers by extracting 6-bit binary words from the 32-bit element array in a serial manner and copying them to its very own 32-bit element as illustrated in figure 3-7. The objective of the copy is to prepare for constellation mapping, which is discussed in the proceeding section on constellation mapping.

Input Data: Serial to Parallel Conversion in Memory

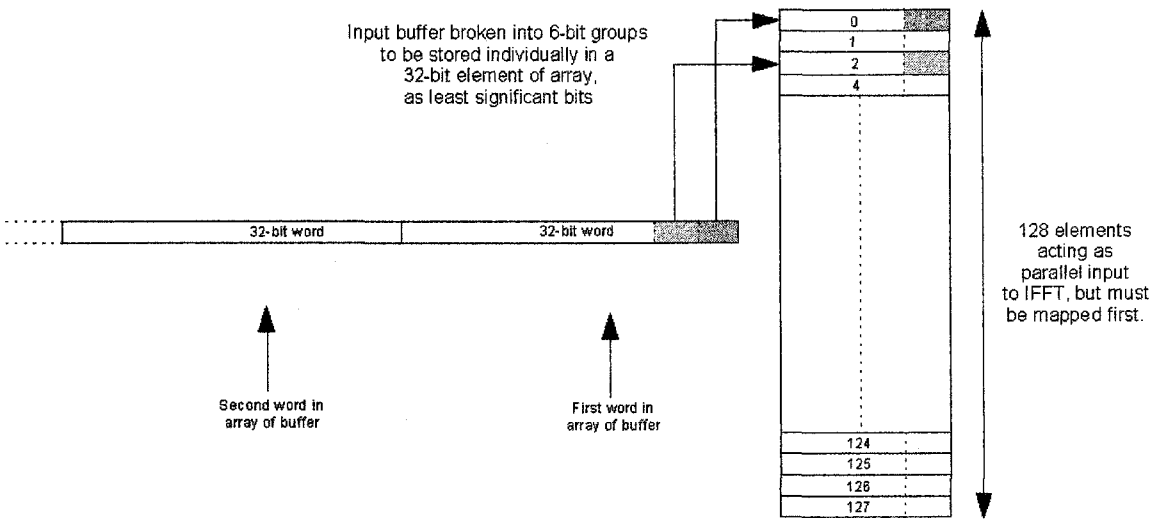


Figure 3-7. Input Data: Serial to Parallel Conversion in Memory

128 32-bit elements are used to for the 64 point FFT input because values each sequential pair of elements represent the real and imaginary value.

As shown in figure 3-8, all data output signals are 5Vpp from the CPLD, except for a 10Vpp output for the master clock reference. The main 25.175MHz master clock signal from the CPLD is 10Vpp because it sources from an onboard clock buffer used to drive all internal clock lines on the chip. The serial port permits a maximum voltage level of 5Vpp, if violated, damage is possible. To prevent damage, a 3dB attenuator is used to attenuate the level along with losses in SMA connectors and wires. Using a scope for verification, there is enough attenuation to correct the clock reference output level to 5Vpp.

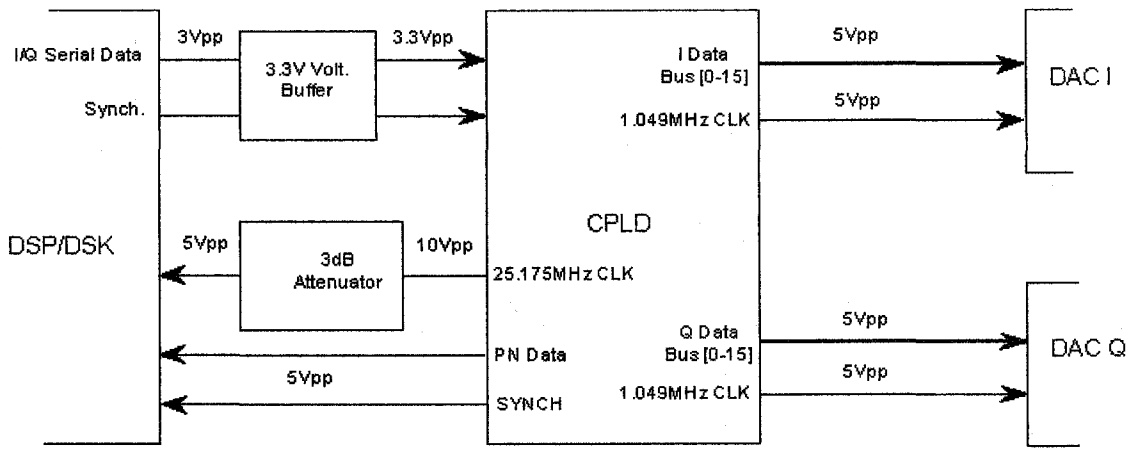


Figure 3-8. Hardware Voltage Interface

The data is transmitted out the CPLD is received by the DSP at a rate of 25.175Mbps, and is synchronized by referencing the timing off the CPLD clock. The McBSP on the DSP is configured to receive 4096 bits or equivalently 128 32-bit words (but only 3840 bits used), and is triggered by the synchronization pulse signal externally provided by the CPLD.

3.2.2 Constellation Mapping/De-mapping

The hardware system only employs rectangular 64QAM mapping to avoid reconfiguration difficulties of receiving longer PN data sequences through the DSP's serial ports, which is required for larger constellation experiments. To elaborate, in the given setup, only a single frame of 128 elements (32-bit words per element) worth of data is accepted for modulation at the DSP's serial port. With the data available from the single frame, fewer OFDM symbols can be constructed for larger constellations. Having fewer OFDM symbols reduces the QAM symbol data to constellation size ratio, and translates to reducing confidence in the experiment results.

In the example of 64QAM, 6-bits of data has a possible range from 0 to 63 in decimal format. Each value can be mapped to the 64 QAM constellation found in figure 3-9, as a

complex value point, spanning two dimensions in the real and imaginary axis. Other rectangular constellation sizes share the same pattern shown in figure 3-9, but with different dimensions. At the hardware level, all values that represent the constellation are stored in the DSP cache memory. Each constellation point requires two 32-bit memory locations for storage on the DSP, because each points needs to store the real (Inphase) and imaginary (Quadrature) value. Thus, 128 memory elements are required to store 64 constellation symbols.

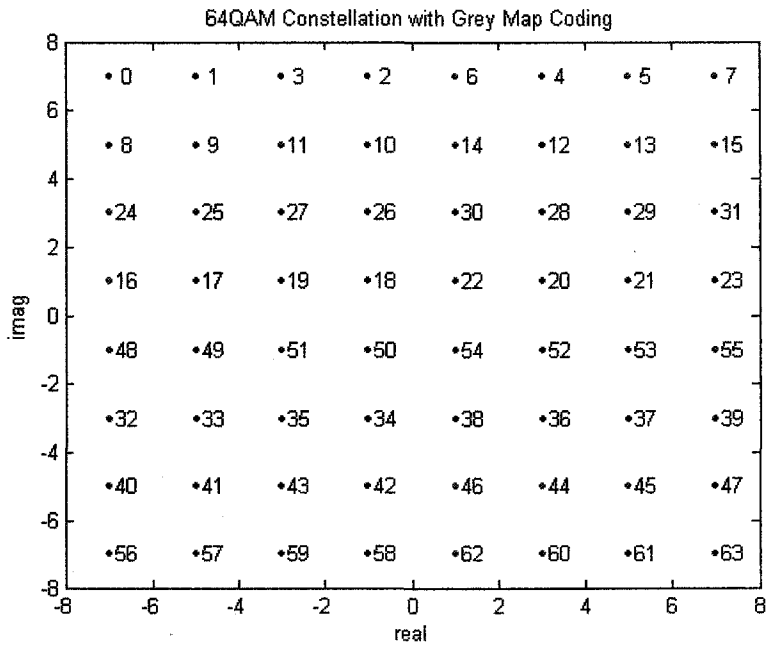


Figure 3-9. Example of Grey Code Mapping to Minimize Bit Errors
Values on map are in decimal values, which represent 6-bit binary words.

To minimize the bit error rate (BER), all constellations used in this study implement Grey-code mapping, where adjacent symbol binary sequences differ by 1 bit. Therefore, if constellation point demodulates into an adjacent neighbor's region, only a minimal single bit error occurs. Demodulating into further distant regions, beyond an adjacent neighbor's, has relatively lower probability and the coding scheme is not optimized to these regions. Simply summarized, Grey-coding is a simple method of improving BER performance of a system without incurring any costs or penalties.

Another way to boost BER performance can be accomplished by increasing the minimal distance between any two points on a constellation. However, the minimal distance is a function of the average power transmitted. The greater available output power transmitted, the higher the signal-to-noise ratio (SNR) at the receiver equating to better BER performance in demodulation.

In the receiver, to reverse the mapping of the binary data, the FFT output is de-mapped on the constellation. De-mapping ideally give the exact input binary sequence if the noise added through the system is below a certain acceptable threshold.

3.2.3 DSP IFFT/FFT

The Inverse Fast Fourier transform (IFFT), and Fast Fourier Transform (FFT) are the key functions in the OFDM modulation and demodulation process to mount data symbols onto and off carriers. They simplify the modulator/demodulator and permit the function to be performed in the digital domain.

FFT/IFFT algorithms are implemented for hardware processing because they are more computation efficient relative to DFT/IDFT, and accomplish the same objective. The fundamental conceptual structure giving FFT its efficient advantage is the butterfly computational unit, which can be found in [15] or in most DSP theory textbooks. With minimal modifications, the butterfly can also perform the IFFT. The FFT/IFFT size is proportional to the number of butterflies required. Each butterfly in radix-2 requires a complex multiply and two complex adds at two data memory locations. This does not include the memory for the twiddle factors (values representing discrete complex values on the unit circle), which are constants that must be computed only once or preloaded and stored in a table before performing the FFT/IFFT function(s). When a FFT/IFFT algorithm is applied to an input sequence, the output sequence is shuffled to a particular pattern. Anticipating this phenomenon, the input sequence can be rearranged with a 'bit-reversal' operation to achieve a properly ordered output sequence. Details on how algorithms for FFT/IFFT shuffle the output sequence are explained in [15].

Supplemental libraries from TI include functions that generate twiddle coefficients and perform the bit reversal operation needed to complete the IFFT/FFT functions. These are libraries are specific to the C67XX processors, and details of the library can be found in [32]. From the libraries, there are many FFT/IFFT methods/options available to choose from, such as: decimation in time (DIT), decimation in frequency (DIF), radix-2, and radix-4. In the hardware configuration, a radix-2 algorithm has been selected over radix-4 to maintain flexibility in the FFT size for possible future changes. To elaborate on the flexibility of the FFT size, radix-2 allows FFT sizes of base 2 {sizes: 2,4,8,16,32,64,.....}, while a radix-4 is constraint to sizes of base 4 {sizes: 4,16,64,256,.....}. However, speed is not equivalent between the two algorithms, for a 64-point IFFT (which is not considered a very large size) the computation difference is 810 (radix-2) and 761 (radix-4) instruction cycles, according to [32]. Using the available clock/cycle profiling tools in code composer, this claim made by TI has been verified in this study before applying the library functions. The verification process involves using a loop in the program and looping it a thousand times; this loops only involves the IFFT call and excludes the twiddle computation. Once the loop is complete, the number of instruction cycles is observed and divided to find the average cycle per 64-point IFFT execution. From this verification exercise an average of 824 instruction cycles is required per IFFT function, which is very close to the theoretical 810 cycles claimed by [32] on page 4-37. The mismatch between the two figures are caused by limitation and accuracy of the profiling tools used to count the instruction cycles, which requires additional clock cycles to save/load registries upon halting/starting at break points inside the DSP program.

When using TI's libraries, the latest version must be installed to avoid errors in the old libraries that comes default with CCS version 2.1.0. Relevantly, these errors are present FFT/IFFT used in this study. Without the updated libraries, the radix-2 FFT/IFFT functions consume excessive number of instruction cycles than claimed by TI. Applying the verification method described above, each IFFT process requires over 50000 instruction cycles.

To use the TI IFFT library functions in hardware, the twiddle factor and bit-reversal functions from the libraries must be called upon, to properly complete the IFFT computation. From several possible methods, here are the steps necessary and used to perform the IFFT in this DSP setup:

1. Generate the twiddle factors (These twiddle factors are only computed once and reused for all future IFFT functions calls).
2. Apply the bit reversal function N point IFFT inputs
3. Call the IFFT function from the library to perform the transformation
4. Normalize the output with respect to N

Outputs of the IFFT/FFT functions used remain in the same memory locations from where the inputs values were taken. In another possible arrangement commonly used, the outputs are placed in a different memory location, thereby doubling the memory usage but simplifying addressing and control. The array used as input to these transform functions consumes *2*N* memory locations because every input/output is a complex value for *N* inputs; each complex value requiring a pair of memory locations.

In the hardware model, an equivalent of 64 orthogonal carriers is used in the modulation scheme. Thus, a 64-point IFFT/FFT is applied to a group of 64 data constellations points to form/decode a single OFDM symbol in the modulation/demodulation process.

3.2.4 Data Scaling to 12-bits and De-scaling

More manipulation must be performed on the IFFT outputs by means of scaling to meet the input dynamic range constraint of the DAC resolution. In the receiver, to undo this scaling effect in the transmitter for recovery of the data, reversal of all scaling is performed before applying the FFT.

The algorithm to accomplish this scaling objective is described as follows:

1. Begin with a search through all IFFT outputs of the OFDM symbols generated to find the maximum and minimal values. (In an actual design the maximum and minimal is predetermined from all possible permutations of the data input sequences)

2. An offset based on the search is then added to all outputs to have the new minimal value at 0.

3. Outputs are scaled by multiplication by a constant to format the data properly for DAC use. For example, if 12-bits are available, the outputs must be within 0 to 4095, assuming the DAC is operating in binary mode versus other data formats such as 2's complement. In the experiments, the data is scaled from 0 to 4095 to utilize the complete 12-bits.

4. Before sending the data to the DAC, the floating-point values must be 'typecast' (converted from one type of word format to another in software) to fixed-point integer values to accommodate the resolution of the DAC. This format conversion requires truncation, but results in very small quantization noise error.

5. After completion of scaling, the real and imaginary values for the complex outputs are stored in their OFDM symbol groups at the corresponding output memory buffers, then transmitted out McBSP1 to the I and Q DACs.

For a visual summary of the scaling algorithm, a equivalent flow chart is provided in figure 3-10.

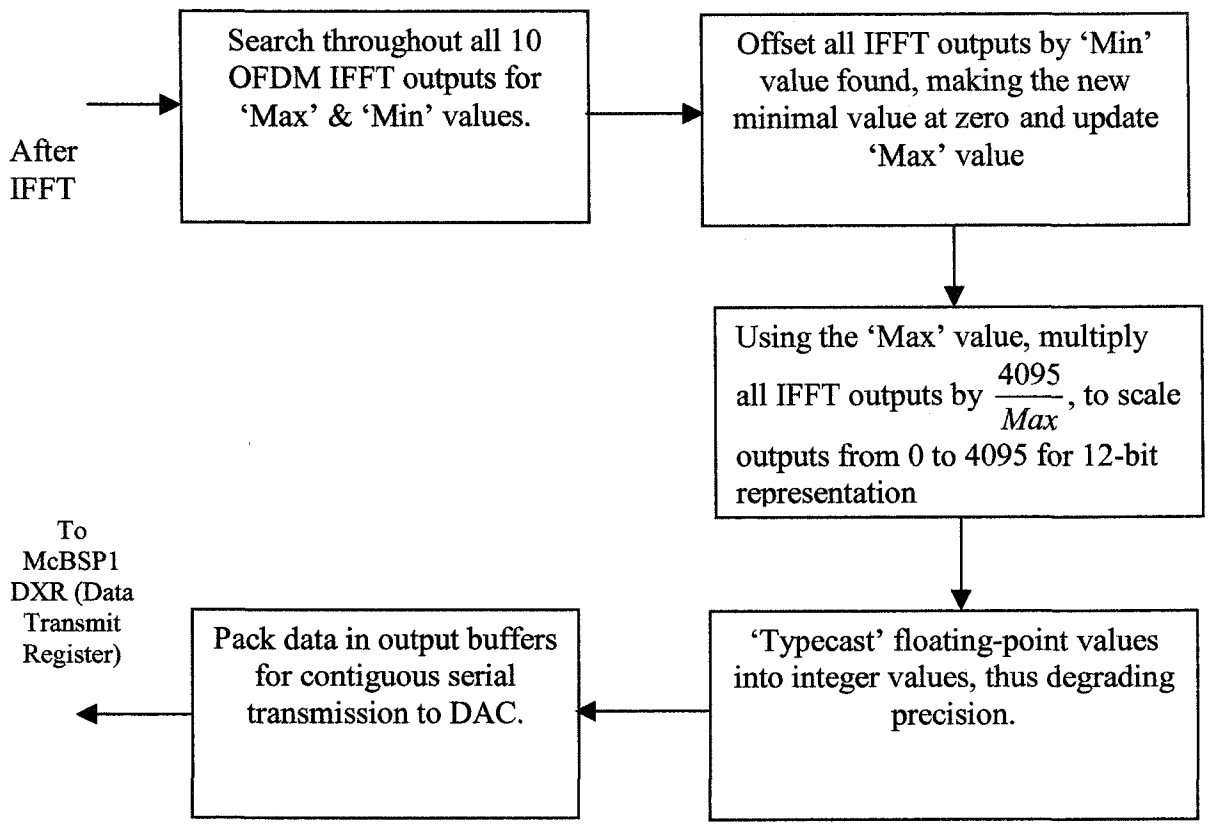


Figure 3-10. Scaling Algorithm Flow Chart

Both the offset value and scaling constant are stored and reused to reverse the scaling effects in data demodulation, before taking the FFT. More details on the 'data packing' are discussed in section 3.2.5 'DSP: Data Packing and Output Transmission'.

3.2.5 DSP: Data Packing and Output Transmission

Following data scaling, the data is packed in a contiguous fashion for seamless transmission. Packing is required since the serial port cannot be configured to transmit 12-bits of data from each 32-bit memory element or an integer multiple value when using the EDMA, which only supports 8, 16 and 32-bits. Lower data rate would result if data were not packed, because the transmitted sequence of data would contain worthless information.

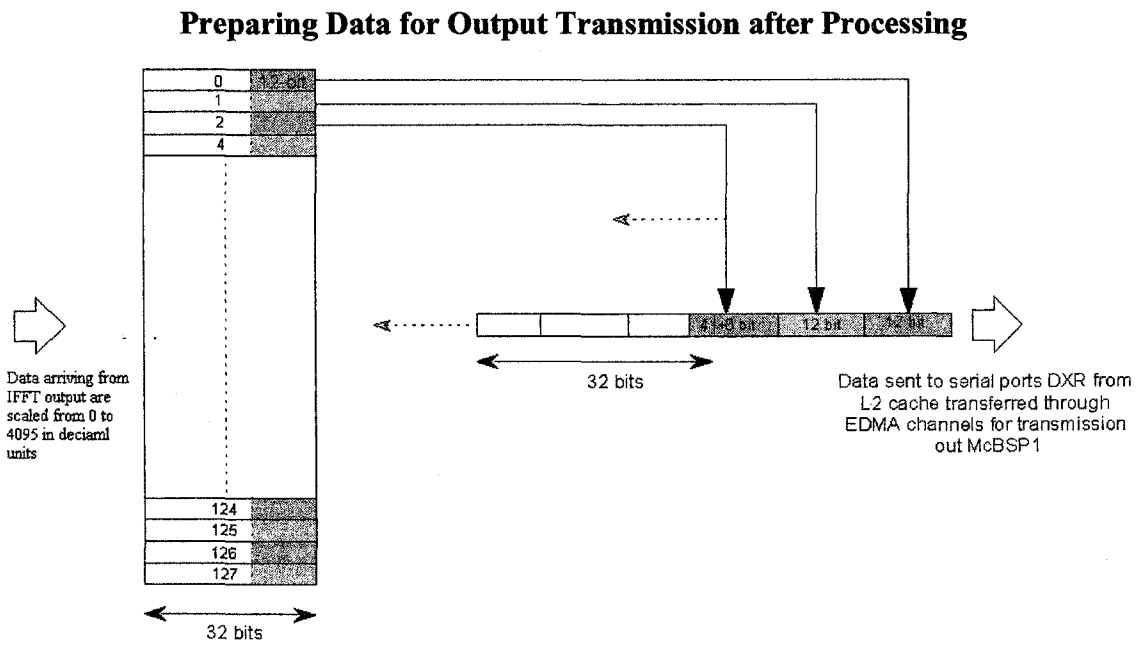


Figure 3-11. Data Packing into Output Buffer to Prepare for Serial Port Transmission

Figure 3-11 demonstrates how the scaled IFFT output array is packed into an output buffer/array of 32-bit elements. Note the I and Q buffers are packed in a fashion where they are interleaved as pairs in their 12 bit data blocks. Upon completion, the data is ready to be transported from the L2 cache to McBSP1 for transmission out the DSP. With external timing from the CPLD of 25.175MHz, the serial port transmits the data at a rate of 25.175Mbps. At this rate, the OFDM symbol rate is given by:

$$\text{OFDM}_{\text{sym}}\text{ rate} = \left(25.175\frac{Mbits}{\text{sec}}\right)\cdot\left(\frac{1}{64}\frac{OFDM_{sym}}{OFDM_{samples}}\right)\cdot\left(\frac{1}{12}\frac{OFDM_{sample}}{bits}\right)\cdot\left(\frac{1}{2}\right)$$

$$= 8.1945K\frac{OFDM_{sym}}{\text{sec}} \qquad (3\text{-}1)$$

Where:

- The '*1/2*' factor in equation 3-1 represents the reduced data rate due to the I/Q words sharing the same serial port.

The present configuration used does not take full advantage of both available serial ports on the DSP to increase the OFDM symbol rate. Thus, another possible configuration explored, is to use two serial ports (McBSP0 and McBSP1), where one port would be dedicated for the I component and the other for the Q component. This method would better utilize the resources available on the DSP and produce twice the data rate relative to the single port. However, a difficulty arises in this method because the output synchronization pulses and the data output times between the two ports are not aligned, and the time delay between the two ports would vary between full power DSK off/on reset. To elaborate on this phenomenon, the two ports operate in a staggered manner, where the beginning of the data transmission time instances are never simultaneous. There is a constant time between the transmission time instances of the ports, but this constant varies between DSP power ups. Designing a serial-to-parallel converter on the CPLD to predict the relative delay and realign the two signals is required, but this method is avoided in the configuration used in these experiments.

3.2.6 Interfacing DSP TO CPLD

As shown previously, figure 3-12 again displays the interface details between the DSP and CPLD for both the serial data (I and Q components) and frame pulse signals (Synch.), which carries the signals to the CPLD input pins. Please note, they do not directly feed into the CPLD, but rather pass through voltage buffers first. From experience, the buffering provides the drive necessary for the CPLD inputs.

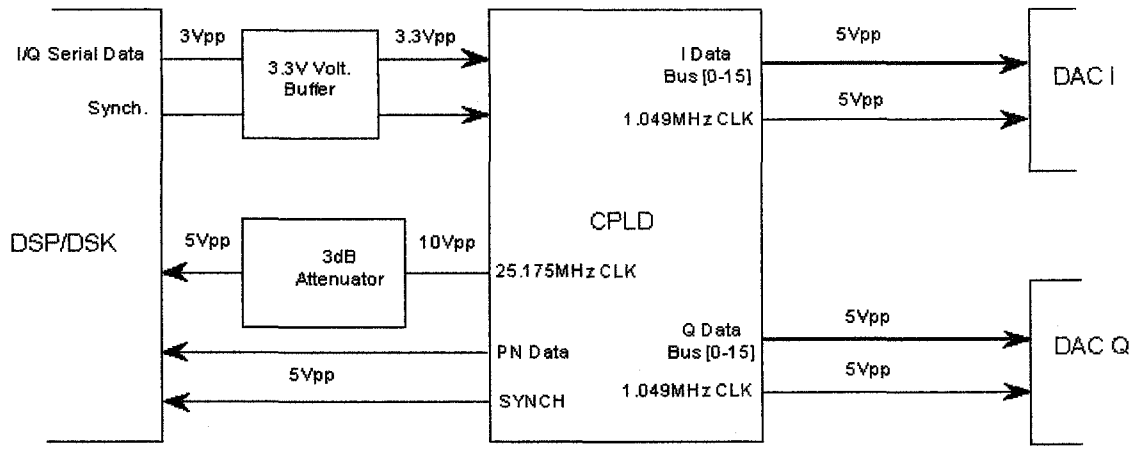


Figure 3-12. Hardware Voltage Interface

Theoretically and verified on a scope, the DSP peripheral outputs have approximately 2.5 to 3Vpp signals swings, which are adequate drive levels to the CPLD's input interface. From experiments, the output of the DSP levels degrades when driving the CPLD and eventually fails to meet the minimal 2.2Vpp required. Possibly the capacitive load of the input pins of the CPLD are too large and draw too much current from the DSP signals.

In summary, the serial port transmits the data and a synchronization pulse to the CPLD with voltage buffering to interface the two devices. Master timing is supplied by the CPLD to the DSP serial port because the timing of the data signals must match the serial-to-parallel converter that it must pass through. Buffering is not performed on the clock signal; instead attenuators are used to ensure the signal is 5Vpp.

3.2.7 Serial-to-Parallel Conversion

As I and Q data components arrive at the CPLD from the DSP in serial format, it is the responsibility of the CPLD to sequester the two, convert them to a parallel format, then pass them to the corresponding DAC with proper timing reference. Similar to the PN

generator, this design is described in 'Verilog' code in its own separate logic module[7], and synthesized with the PN generator onto the same CPLD chip.

Before performing any manipulation on the received signal at CPLD from the DSP, the signal may have a relative delay factor and realignment is performed to ensure any delay factor is eliminated. Since the data signal received is derived from the same 25.175MHz clock, then a flip-flop can be used to capture the input signal at every clock cycle to realign all input before performing the serial-to-parallel conversion. This realignment method incurs a one-clock cycle delay using this flip-flop technique.

Once realigned, the signal begins to undergo the serial-to-parallel conversion process through a 24 flip-flop shift register structure. From the DSP, alternating I and Q 12-bit words stream into the CPLD's serial-to-parallel shift register continuously and contiguously. For every 24 clock cycles the shift register sends its parallel output to DACs. The first parallel 12-bits are sent to the inphase channel DAC, and the following 12-bits are sent to the quadrature channel DAC. Please refer to figure 3-13 for illustration.

Illustration of Serial-to-Parallel Conversion

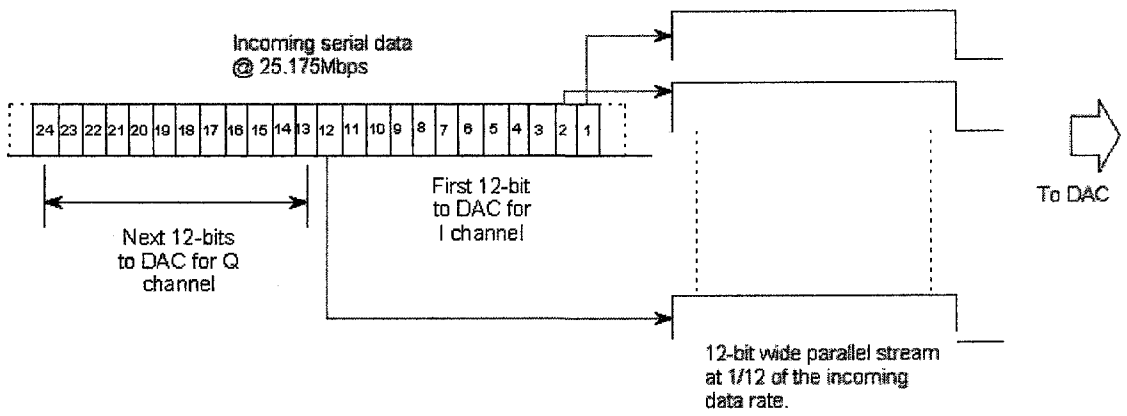


Figure 3-13. Serial-to-Parallel Conversion Performed in CPLD

[7] Verilog code for serial-to-parallel module is provided in appendix C.

In addition to performing the conversion, the CPLD supplies the clock to the DAC for timing reference. Importantly, the clock must be oriented such that the rising edge of the clock must be at the middle of DAC input data period. The DAC clock is obtained by dividing the CPLD master clock rate by 24, as shown:

$$f_{DAC} = \frac{25.175MHz}{24Bits/OFDM_{sample}} = 1.049MHz = f_{OFDM_{sample}} \quad (3\text{-}2)$$

The period of the reference clock is given by:

$$T_{DAC} = \frac{1}{f_{DAC}} = 953ns = T_{OFDM_{Sample}} \quad (3\text{-}3)$$

3.2.8 Transmitting OFDM Symbols Out Through I/Q DACs

From the serial-to-parallel conversion, the binary words in parallel format translate through the DACs to its corresponding analog levels to complete the OFDM symbol transformation. Once transformed, usually the analog waveform undergoes up conversion onto a carrier in a radio link (as an example) to transmit through the channel within a designated bandwidth assigned to the application.

In each I and Q channel, there is a dedicated DAC to perform the translation on the incoming 12-bit binary words to corresponding analog waveform levels. If incoming data to each DAC ranges from 0 to 4095 (decimal values representing binary data), then the complete dynamic range available on the DAC is utilized. Using the complete dynamic range maximizes the precision of data conversion, minimize quantization noise, and maximize the potential to increase the minimal distance between any two points on the constellation utilizing the full DAC output voltage swing available. In the hardware setup, 100% of the dynamic range is utilized, since IFFT output data values are scaled from integer values 0 to 4095, as discussed in section 3.2.4 'Data Scaling to 12-bits and De-scaling'.

Error Sources:

The peak output of the DAC in the hardware experiment has been calibrated to 4Vpp on a digital scope, referenced to 1M Ohms. This calibration method does not provide perfect accuracy, and DAC output error in the configuration is unavoidable. However, the error is visible in the results and used to simulate signal level mismatches at the receiver that may occur in typical systems. Selecting 4Vpp output swing for each DAC in the experiments provides a good signal swing for the ADC input range, and is suitable for the investigations throughout this study. With 4Vpp set at the transmitter, the demodulation computation uses this value when undoing the DAC output scaling range. Thus any mismatch error with either or both DAC(s) by not transmitting the expected 4Vpp, is attributed to the inaccuracies of the calibration and cause gain errors in the demodulator.

Gain error within the DAC[8] is another possible significant factor contributing to mismatch errors. From inspection of the THS5661 DAC specifications listed back in table 2-2, gain error of the DAC has the potential to be a significant contributor to mismatch error. The specified 1.3% FSR gain error equates to 0.052V @ 4Vpp DAC output. This gain error source is another reason for studying signal level mismatches at the receiver.

3.2.9 ADC Sampling at the Receiver Input

The ADCs in the hardware system are located in the HP54542A digital storage scope. Data through the I and Q channels are received by the scope and sampled at 25MSPS. Operating at 25MSPS is a suitable sampling rate chosen on this scope model because there is enough storage space to capture 10 OFDM symbols from the DSP/DAC at the given 1.049M $OFDM_{samples}$ per second output rate. In fact, there is more than enough storage space for 10 OFDM symbols, and the time marker (equivalent of 1 OFDM

[8] The DAC's gain error curve is nonlinear over the input/output range and must be measured from experiments (Converter gain error is described previously in section 2.5.2 'Gain Error'). Thus the nonlinear effect is difficult to model in the simulations. However, the nonlinear effects are not investigated in this study, rather the gain error contribution is assumed linear.

symbol period). Another reason for the selected sampling rate is to achieve an over-sampling ratio as close to an integer value as possible and to have a fine time resolution in the sampled data for processing purposes. The over-sampling ratio is defined as the following:

$$\text{Over-Sampling Ratio} = \frac{f_{ADC}}{f_{OFDM_{sample}}} \tag{3-4}$$

$$= \frac{25MHz}{1.049MHz}$$

$$= 23.83 \tag{3-5}$$

With fine time resolution and an over-sampling ratio near an integer value, data processing/re-sampling is less complicated[9]. The over-sampling performed on the incoming signal is necessary because there is no incoming synchronization signal. Thus, the asynchronous signal must be over-sampled to have enough time resolution to accurately determine the starting point in the signal of the OFDM symbol sequence. To begin processing the data captured by the scope, a search for the all zero period time maker transmitted by the DSP. Once found, the beginning or the first scope sample of the 10-symbol sequence is known. This first scope sample is used as the 0 sec reference time location, and thereafter every sample which follows is located at increments 40ns[10] on the time axis.

[9] Less complicated, to decimate to one data sample per OFDM sample, as close to time instance $\frac{T_{OFDM_{sym}}}{2}$, as shown in figure 3-27.

[10] 40ns is the period of the sampling rate 25MSPS, which is used at the ADC.

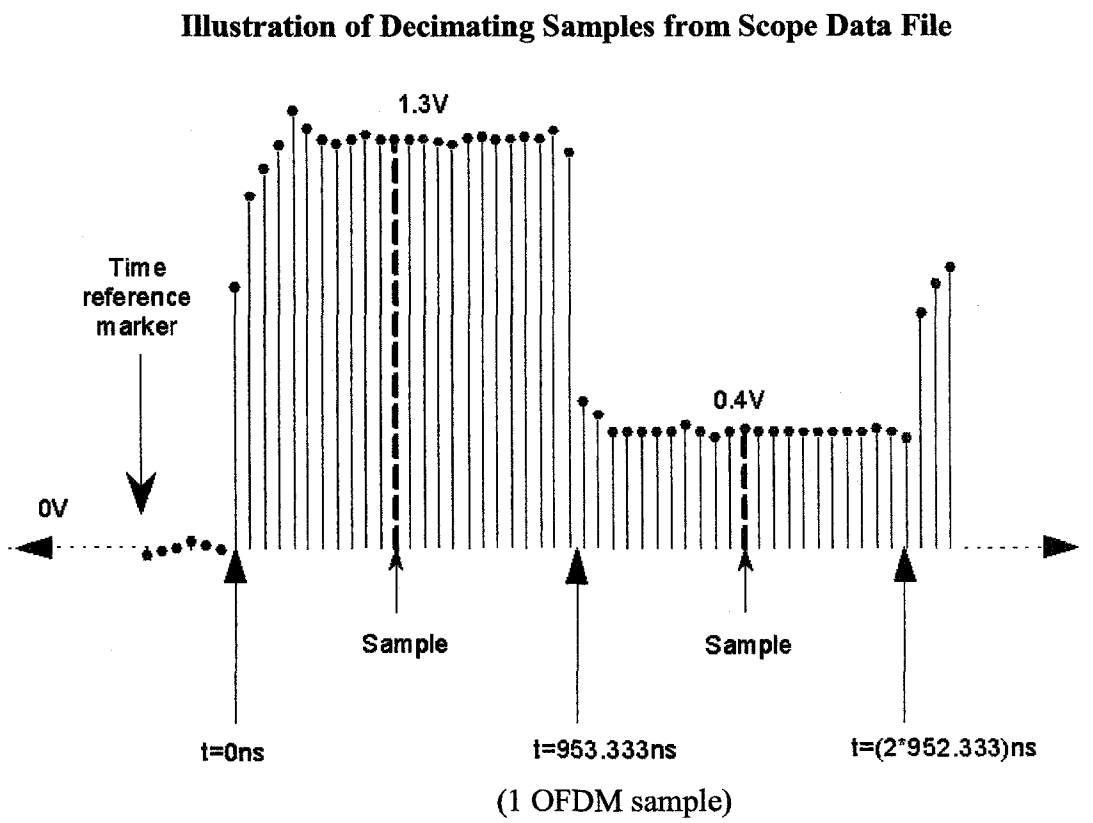


Figure 3-14. Illustration of How Demodulation Samples Decimated from Scope Data Files

Since the time period of each $OFDM_{sample}$ is 953.333ns, data samples are grouped together within the theoretical time boundaries, and then decimated by selecting the data sample closest to $\frac{953.333}{2} ns = 476.7 ns$ in each $OFDM_{sample}$. Please refer to figure 3-14 for an illustration of the decimation concept. Once the decimation is completed, the decimated samples are demodulated and figure of merits are computed. The details of the demodulation are discussed in the next section 3.2.10 'Receiver Demodulation'.

3.2.10 Receiver Demodulation

Once the scope has captured an I/Q data set for 10 OFDM symbols, storage media (floppy diskette) is used to transfer the data to a PC for demodulation and analysis. The main demodulation details have already been discussed in sections 3.2.2 to 3.2.4 with

their reciprocal blocks in the transmitter. For example, in the demodulation process the FFT must be performed to reverse the IFFT function, which were both discussed together. The complete summary of steps performed for demodulation are given by:

1. The first step (discussed in previous section 3.2.9 'ADC Sampling at the Receiver Input') is to decimate the scope-captured sampled sequence by a factor of 23.83. This is accomplished as follows:

    a. Data samples from the scope are analyzed and a search is performed for the first OFDM sample in the first OFDM symbol. To find this, the search targets the time marker interval (nothing is transmitted in that interval). At the end of the time maker, the next scope sample is used as a time reference.
    b. Samples used for demodulation are selected as close to the center of each OFDM sample period i.e. $t=\frac{T_{OFDM_{sym}}}{2}$. For example, if 23 samples are determined to correspond to the same OFDM symbol, the 12th sample is nearest to sample instance at $t=\frac{T_{OFDM_{sym}}}{2}$ is selected.
2. Because the scope is set to 'DC coupled' with the DACs, the DC level from the DAC are present in the samples. This DC offset is removed with a pre-determined value from calibration of the DAC with the scope.
3. Reverse scaling and reverse offset is performed on the data to undo all manipulations inflicted in the transmitter, after the IFFT function taken during modulation.
4. MSE_{wave} [11] is computed with de-scaled sampled values and theoretical values (at the IFFT output) stored from modulation.
5. FFT is applied to the decimated samples.

[11] MSE_{wave} is defined and discussed in section 3.7 Figure of Merit

6. QAM symbols are verified with the theoretical values stored from modulation (after constellation mapping). The results of the computation provide the number of QAM symbol errors that have occurred in the simulation.
7. Last step is to use the demodulated QAM symbol coordinates and theoretical QAM symbol coordinates, to compute the MSE_{cst} and SIR_{bit} [12].

3.3 Functions Performed by the DSP

3.3.0 Introduction

This section discusses certain functions and miscellaneous details that are important to DSP to meet its processing objectives:

1. Receive binary data from the CPLD through the DSP's serial port
2. Perform OFDM modulation
3. Store modulated symbols in onboard memory
4. Send modulated symbols out serial port to the CPLD serial-to-parallel data converter.

These functions discussed are important factors in DSP design, especially for real-time processing. Although, real-time processing is not an objective in the configuration of the hardware model, it could be a possible alternative for future works, in extension to this thesis. For flexibility, consideration is made to the present configuration in the DSP to easily switch to real-time mode with minimal modifications to the program. In this section, a discussion is made on a few functions performed by the DSP in this design and their roles in a time critical processing design.

3.3.1 EMDA Data Exchange with Peripherals Devices

In the hardware configuration, data transmitted and received through the serial ports of the DSP are serviced through the EDMA channels. If the EDMA channels are not

[12] MSE_{cst} and SIR_{bit} are discussed in section 3.7 Figure of Merit

utilized, the CPU would have to constantly service the ports and not be able to allocate more time to processing data and attend to other tasks. The EDMA channels alleviate the CPU from this task by transporting the data to/from the L2 cache, which is very important in real-time processing designs. A channel only needs to interrupt the CPU when it is ready at the beginning and end of each transfer, allowing the CPU to perform other tasks while transfer runs in the background. When the EDMA interrupts the CPU to signal the data has been delivered to the L2 cache, the CPU is then permitted to process the new data. Hardware interrupts (HWI) generated by the peripheral devices, or internal events, is a method to manage the requests made to the CPU and allow for real-time processing. The CPU halts the current execution flow and attends an asynchronous interrupt, after saving all machine states by saving the registers, servicing the interrupt, and then returning to its processing or algorithms it was performing with the restored registers. Each peripheral device has a dedicated EDMA channel, and its channel must be properly initialized and configured in a 2KB EDMA parameter table (PaRAM). The DSP/BIOS GUI in CCS potentially allows the EDMA configuration to be much easier and quicker than if it were to be done in C code.

3.3.2 Memory Management

In the DSP setup, to minimize the time utilized moving data between external memory sources, all symbols are processed from L2 cache upon data arrival from the EDMA. This method minimizes fetch time by the CPU to process the data versus storing the data in an external location through the external memory interface (EMIF), which requires a time costly EDMA transfer. The CPU can access the L1 cache quickest relative to the other memory location, but limited to the amount of information it can cache because of the small 8KB size. Thus, this memory block is used for data and programs that are most frequently called upon. With the different levels of memory available, strategic planning can optimize processing for less time to help meet real-time criteria. Please refer to figure 3-15 for an illustration of the memory hierarchy and the DSP/CPU's decision map performed when accessing the different levels of memory.

DSP/CPU: Memory Hierarchy Flow Structure

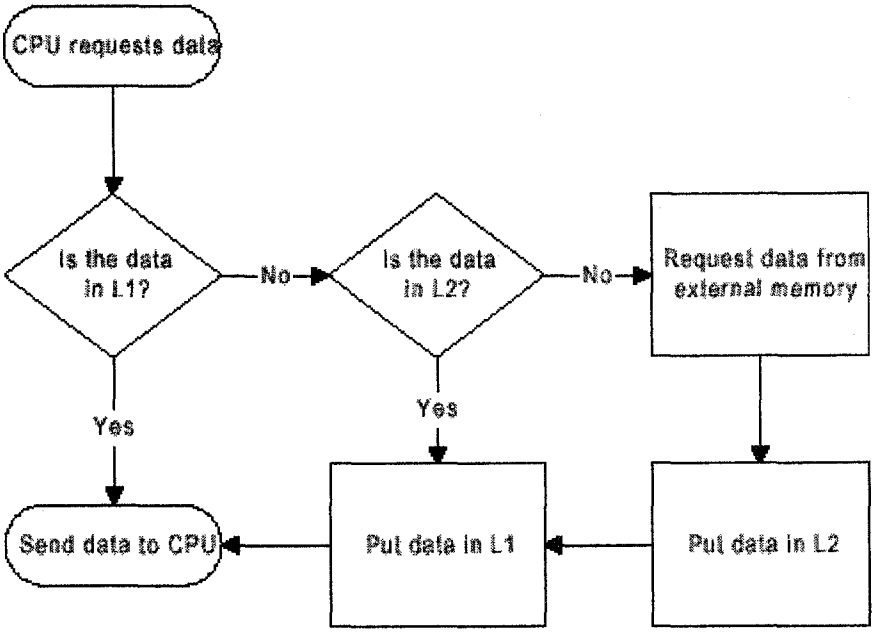


Figure 3-15. DSP/CPU: Memory Hierarchy Flow Structure

3.3.3 IFFT/FFT Optimized Code

When performing the OFDM modulation, there is no concern about the processing time to perform the IFFT/FFT. In the future, this maybe critical if processing were to be real-time. To prepare the present setup for such a possible demands, the IFFT/FFT functions used are created by optimized assembly code, which has greater potential to efficiently outperform higher-level 'C' code. To elaborate, creating a project in 'C' language in CCS, then compiling, the compiler tries to generate as many parallel instructions as possible, but is limited in its ability versus manually optimized assembly code. The time saving using the optimized assembly code can lead to vast performance gain in algorithms and functions within a program. Numerically, savings with optimized algorithms can reduce computation by as much as 20% less, according to [30], but is not guaranteed. Although not a concern in the hardware experiments done, the IFFT/FFT used in the thesis have already been optimized and compiled.

3.4 DSP Event Flow Control

This section describes how the order of events flow in the design used to meet its OFDM modulation objective. In figure 3-16, a flow diagram summarizes the order of events that occur.

During run-time various events can occur simultaneously or in a very short time span to each other. How the DSP handles each one is important, to keep events in order and flowing smoothly. The program written for the DSP begins with initialization of variables and aligning them in the L2 cache for addressing efficiency. Next, the EDMA channels and McBSP1 serial port go through proper initialization procedures, and then all interrupts are properly activated. With the DSP system prepared, McBSP1 is activated to commence receive and transmit activity. Upon activation, the interrupt handler takes over orchestrating incoming events, which stream in immediately after the serial port has been activated. The serial port 'receive' and 'transmit' functions operate independently but neither one can be stopped alone in an obvious manner[13]; only a complete serial port halt is permitted to disable both functions. While the serial port is waiting for its receive pulse and PN data, the transmitter is sending out an empty buffer and posting an interrupt every time it completes transmission. When an external synchronization pulse acknowledges McBSP1 data arrival, the serial port will send a message to the EDMA controller to begin transporting the incoming data at the McBSP1 'data receive register' (DRR), grouped into 32-bit elements, away to the L2 Cache. Upon reception of 128 elements, the EDMA will post a hardware interrupt to the CPU with its interrupt identification number. The interrupt service routine (ISR) will handle the HWI. Inside the ISR, a software interrupt (SWI) is called to handle the incoming data rather than trying to respond to the interrupt directly.

[13] It maybe possible to ignore/halt the operations of either the 'receive' or 'transmit' function in the serial port by not clearing the previous interrupt issued by the function. By doing so, the function does receive any servicing/attention from the CPU again, until the interrupt is cleared.

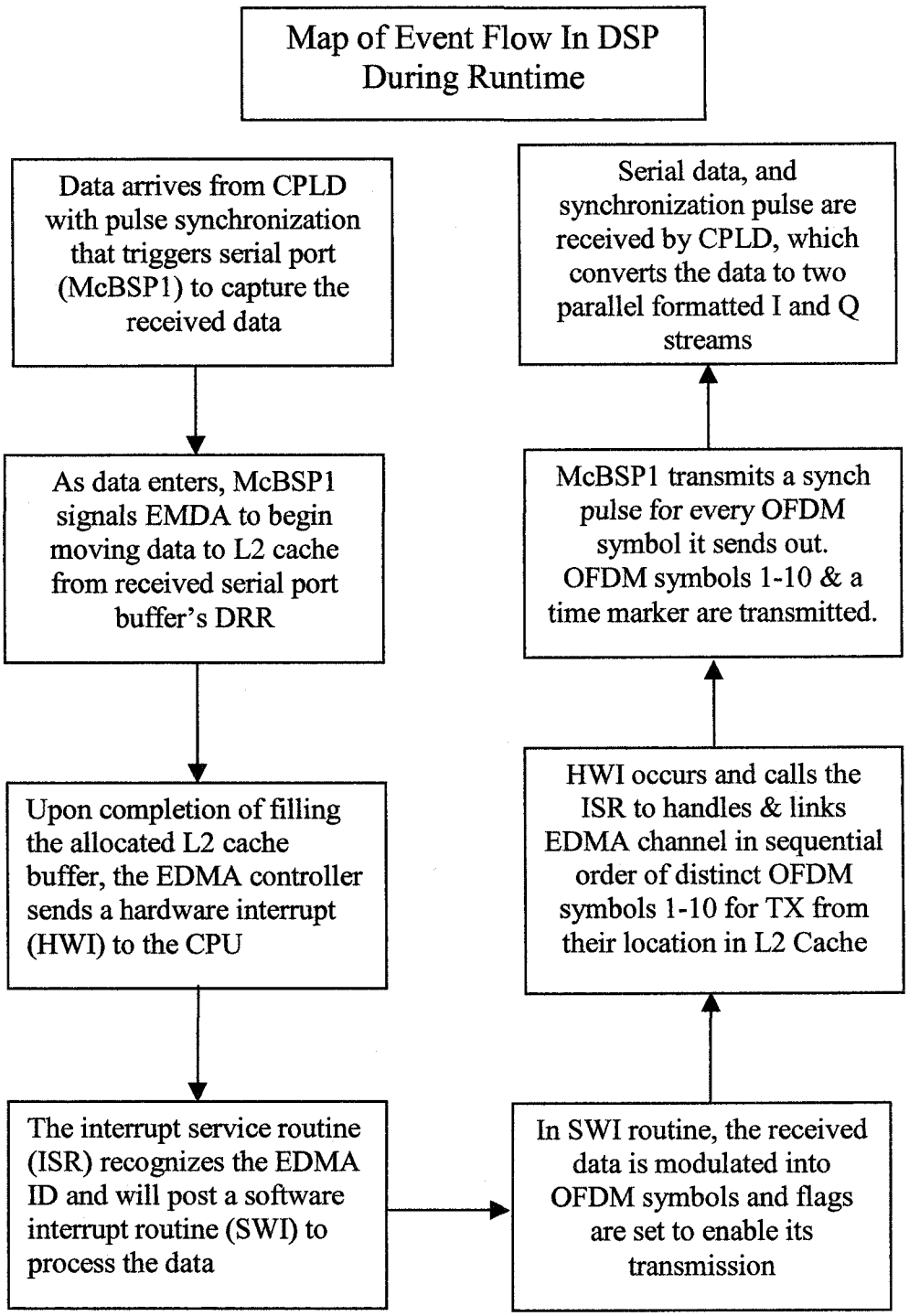


Figure 3-16. Event Flow Sequence In DSP During Runtime

Processing the lengthy data in the HWI ISR is avoided because HWI ISRs must be completed quickly before another interrupt occurs, which would permanently stop and drop the present processing task in the ISR (non-interruptible for correct execution, otherwise fragmented processing errors occur). Using the SWI routine to process the data will not interfere with HWI events. SWI are interruptible threads and are lower in priority to HWI, allowing HWI to run without interference. In the program written, the SWI posted processes input data, forming 10 OFDM symbols for output. When completed, a flag will enable a section of code in the ISR, which sequentially links the OFDM output symbol buffers with the EDMA channel 14 to the McBSP1 for transmission between interrupts.

3.5 Software Model Introduction

The second model flow diagram, shown in figure 3-17, represents the software model used to perform simulations contained within the software itself. If the filtering blocks were excluded, then this model is identical to the previous hardware model, but only presented in a different fashion in this section. One purpose this model serves is to emulate the complete hardware model when configured properly to match the hardware setup. In extension, the other objective of the model is to isolate each analog factor examined in the hardware section, examine relative I/Q delay caused by filtering, and ADC saturation caused by both DC offset and gain error. Also, a combinational case of all factors is observed in simulation. As applied in the hardware experiments/simulation, MSE_{wave}, MSE_{cst}, and SIR_{bit} gauge the performance of each simulation.

The simulations in software are more flexible and scalable than hardware, thus longer data sequences with larger constellations can be relatively easier and quicker to reconfigure than the hardware setup. Taking advantage of the scalability, the hardware experiments will only perform 64QAM constellation modulation, while the software simulations will include 16, 64 and 256QAM per each simulation case and applies longer PN data sequences that are discussed in details further on in this software model segment.

The software model specification can be found in table 3-3. They are presented differently than the hardware specification because of the various constellation and investigation configurations in the software model. In contrast to the hardware details discussed, here in sections 3.5 and 3.6, the software model details do not deeply divulge into data manipulation and processing, since its low level operations within the computer processor is not of interest.

Model 2: Software model

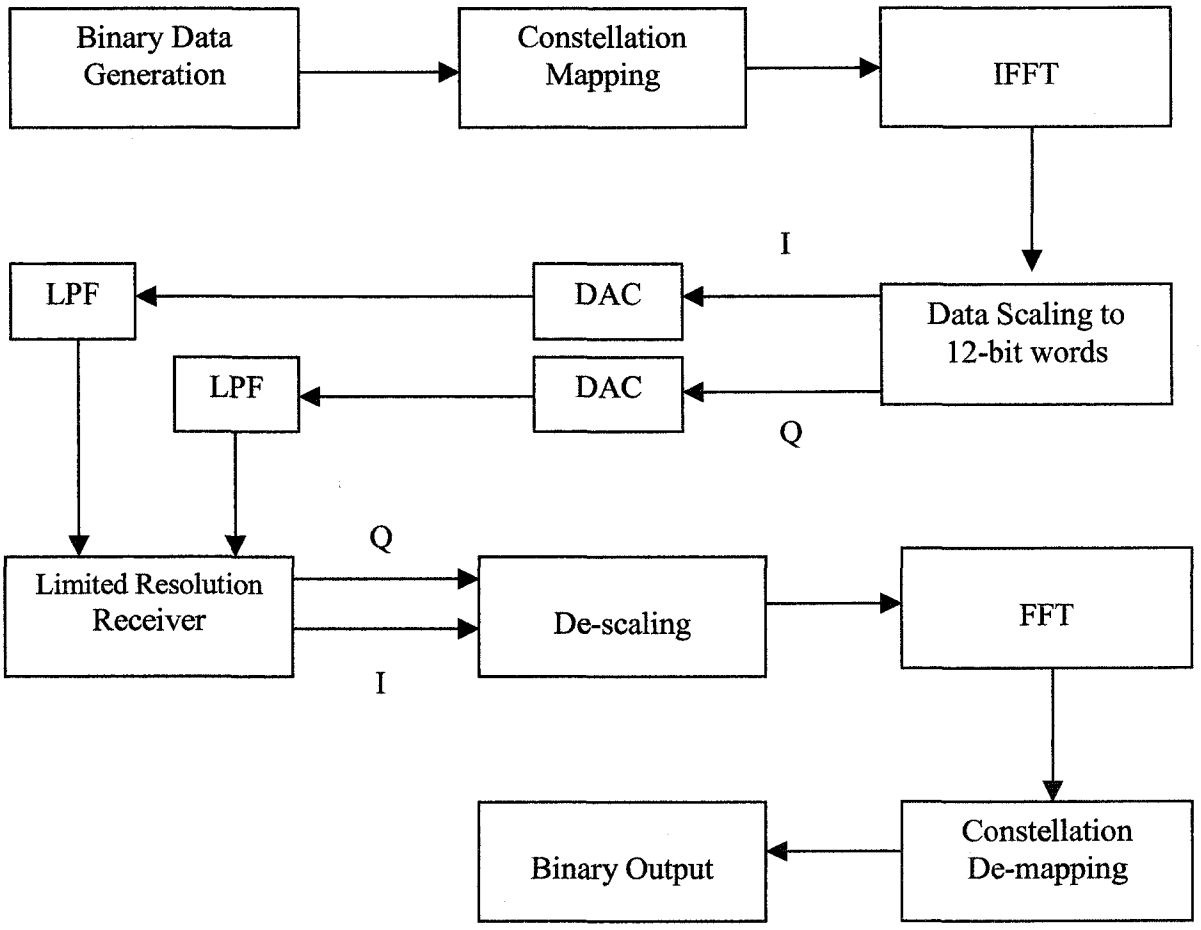


Figure 3-17. Software Model

Table 3-3. Table for Software Model Summary

Item	Value	Comments
Net PN sequence length	Various *Please refer to table 3-4 for details*	Higher order LFSR used for larger constellations because longer data sequences required for comparable test between each constellation
N-IFFT/FFT	64	Equates to 64 orthogonal carriers
Constellation	16QAM 64QAM 256QAM	Three different constellations used
Average Number of OFDM samples per constellation point	40	For more extensive results than hardware
OFDM Symbol rate	8.1945K $OFDM_{sym}$/sec	This rate matches the hardware data rates.
DAC resolution	12-bits	DAC set to operate in binary mode
DAC output swing	Default 4Vpp Variable	Can be varied to cause gain error and/or I/Q amplitude imbalances
ADC Resolution	Varied from 6 to 11-bits over 8V	This is varied to observe performance versus reduced ADC resolution.
ADC sampling	At time instant $T_{DAC}/2$	Sampled at mid-point of each OFDM sample in the analog wave.
LPF[14]	7^{th} Order Filter	From experiment, this is near highest (9^{th}) order digital FIR filter without causing QAM symbol demodulation errors. Linear phase FIR filter has equiripple response.
Expected DAC voltage at receiver	4Vpp	Receiver will use this value to undo the scaling of the DAC output

[14] For more details on low pass filter see table 3-5

3.6 Software Model Block Description

3.6.0 Pseudo Random Data Generation

Like the hardware PN generator, the software model utilizes LFSR structures to generate PN binary sequences for the simulations. However, the LFSRs are described in Matlab language rather than in Verilog, as previously done in the hardware to synthesize into digital logic. The difference in the descriptive language is irrelevant to the objective in both cases. Both are able to describe the exact structures and produce the same binary sequence, which is important in the simulations performed to mimic the hardware model. A summary of the various PN generators used in the simulations are summarized below in table 3-4.

Table 3-4. Summary Table of PN Generator used for Experiments and Simulations

	LFSR nth order	Polynomial equation[15]	Total PN Bits available	Total Used
16 QAM (Software simulation)	17^{th}	$x^{17}+x^3+1$	131072	12800
64 QAM (Hardware experiments and corresponding simulation)	12^{th} and 13^{th} combined	12^{th} Order: $x^{12}+x^7+x^4+x^3+1$ 13^{th} Order: $x^{13}+x^4+x^3+x+1$	12288	7680
64 QAM (Software simulation)	17^{th}	$x^{17}+x^3+1$	131072	76800
256 QAM (Software simulation)	17^{th}	$x^{17}+x^3+1$	131072	409600*

**PN sequence from LFSR repeated and continued for data use in modulation*

[15] Equations are taken from [9]

The longer PN data sequences are used in software simulation cases to help deliver greater confidence in the simulation results. However, to mimic the hardware experiments, the exact PN sequence is replicated in the software simulation by using the same 12^{th} and 13^{th} order polynomials in the LFSR configuration[16].

In each software simulation case, the 17^{th} order LFSR provides an adequate random binary sequence to form an average of 200 data symbol points per each reference constellation point. The objective by doing this is to provide an equivalent and fair data length between the simulations of the three constellation sizes used.

3.6.1 Mapping Binary Data Sequence to Constellation

No different than the hardware model, the hardware experiment's equivalent simulation applies the exact same 64QAM constellation mapping scheme. However, the pure software simulations explore further into other constellation sizes to observe their sensitivity to all analog interface factors examined in this study. The new set of rectangular constellations includes the following sizes: 16, 64, and 256 QAM, which all feature grey-code mapping. Non-rectangular constellation configurations are avoided because of demodulation complexities introduced.

3.6.2 DSP IFFT/FFT

Using the IFFT/FFT functions in Matlab only requires a single standard function call and consideration to computation load is not applicable because processing resources is not a limiting factor. Thus, the floating point IFFT and FFT algorithm details in Matlab are masked and not important to simulation results.

[16] The hardware setup uses both a 12^{th} and 13^{th} order LSFR to generate a shorter PN sequences relative to the 17^{th} order LFSR used in software simulation. By using the lower order PN generators, mapping the binary data to QAM data symbols ensures there is full coverage and more even distribution on the constellation.

Similar to the hardware model, to modulated data onto 64 orthogonal carriers, a 64-point IFFT/FFT is applied to 64 constellations points to encode/decode a single OFDM symbol in the modulation/demodulation process.

3.6.3 Data Scaling and De-scaling

All the simulations emulate the 12-bit DACs in the transmitter. Therefore the same scaling and search algorithms are performed on the IFFT outputs for all OFDM symbols in the simulation to accommodate the 12-bit dynamic range of the DAC's input. Thus the scaling specifications remain unchanged at 100% of the DAC dynamic range and 4Vpp output is utilized, by scaling the IFFT output data values to 0 to 4095. De-scaling performed in the demodulator reverses all scaling factors, offsets, and DAC scaling applied before performing the FFT to recover the data.

Unlike hardware experiments, mismatches between the DAC output swing and receiver demodulation constant to undo the DAC scaling do not exist, and must be fabricated. To emulate the mismatch errors found in the hardware system, the output swing of the DACs can be varied as a parameter away from 4Vpp, inflicting gain error and/or I/Q amplitude imbalances in the system.

3.6.4 Low Pass Filtering

In a typical system, the signal transmitted through a channel must be band-limited to constraint itself to an allocated bandwidth to avoid interference with other signals occupying different section of the frequency spectrum. To accomplish signal containment within a specified bandwidth, analog filters are employed at both receiver and transmitter ends of a communication system. In addition, filters also have other objectives, which include:

1. Band-limiting signals
2. Remove excessive noise outside passband; isolate signal

3. Prevents aliasing before sampling

In the software model, only one pair of filters for I and Q filters are shown and needed because the signals do not pass through an AWGN channel.

By experimentation, a 7th order low pass filter (LPF) was designed for the software model using Parks-McClellan method. This method is an algorithm developed for design of an optimal equiripple FIR filters with linear phase response. A summary of the filter specifications is listed in table 3-5 and a frequency response plot is shown in figure 3-18.

Table 3-5. Summary Table for FIR Filter Design used in Software Simulations

FIR filter specification	Value	Units
Filter order	7	None
Filter Sampling Frequency (*fs*)	1.049 * 9 = 9.44	MHz
3dB cutoff	1.049 ($1 / T_{DAC\_CLK}$)	MHz
Normalized 3dB cutoff	0.22	Hz
Normalized Stopband	0.54	Hz
Transition band width [From 3dB cutoff to Stopband]	1.51	MHz
Attenuation in Stopband	Less than 35	dB
Inband ripple	0.01	dB
Stopband ripple	0.1	dB

Frequency Response of 7th Order Filter used in Simulations

For the thesis, a FIR filter solution is selected over other digital filter types for its linear phase properties. The resultant FIR low pass filter specifications used in the simulations would limit bandwidth with maximum attenuation in the stopband and quickest transition band slope without causing demodulation QAM constellation symbol errors. Figure 3-19a and 3-19b demonstrates the filter output using DAC output as the input signal to the filter, with a 9th order and a 7th order FIR filter. The 9th order filter has a sharper

transition band and stopband attenuation than the 7th order filter, which is selected for the software model.

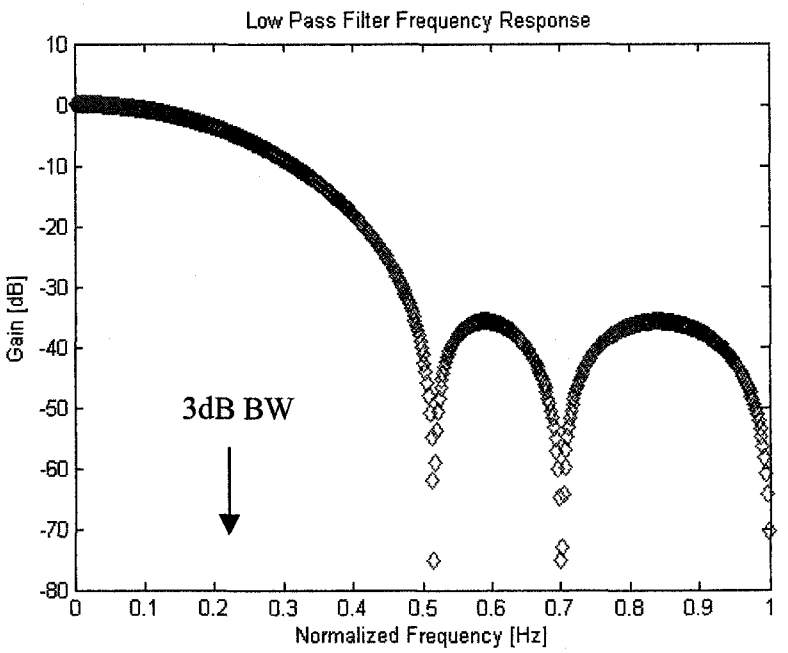


Figure 3-18. Frequency Response of LPF used in Simulations
1Hz represents normalized *fs/2*.

The better abilities to attenuate and block out the higher frequency components with the 9^{th} order filter relative to 7^{th} order filter, produces a greater overshoot (dashed line in figures 3-19a and 3-19b) with respect to the ideal digital step signal (solid line) waveform. The overshoot is known as amplitude distortion, and is caused by removal of higher frequency components in the signal, which shares a responsibility in forming the step-shaped waveform. To show an example of the effect of the amplitude distortion causing demodulation QAM symbol errors, the constellation plot is shown for the 9^{th} order FIR filter in figure 3-20.

The sampling frequency of the digital filter is set to nine times the 3dB bandwidth value, which provides fine time resolution between samples at the output of the filter due to the high sampling rate.

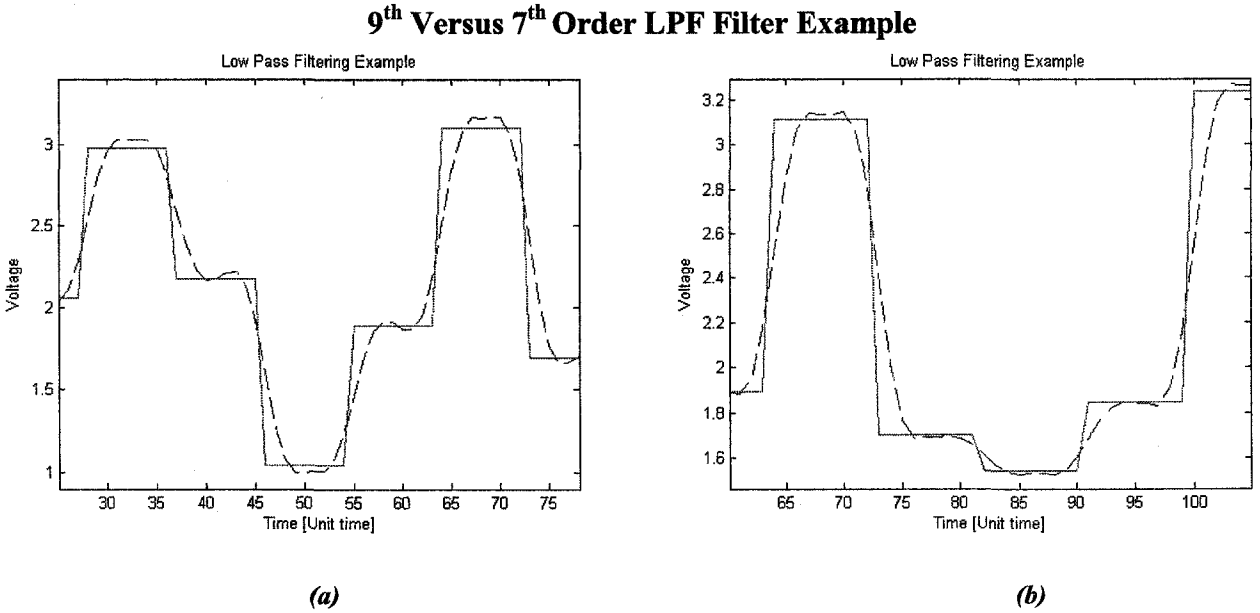


Figure 3-19. Amplitude Distortion Caused by: (a) 9th Order LPF Filter (b) 7th Order LPF Filter, that is used in Thesis.

In both figures: Dashed lines for post filter signal. Solid line is DAC output; it is an analog waveform that appears as digital steps.

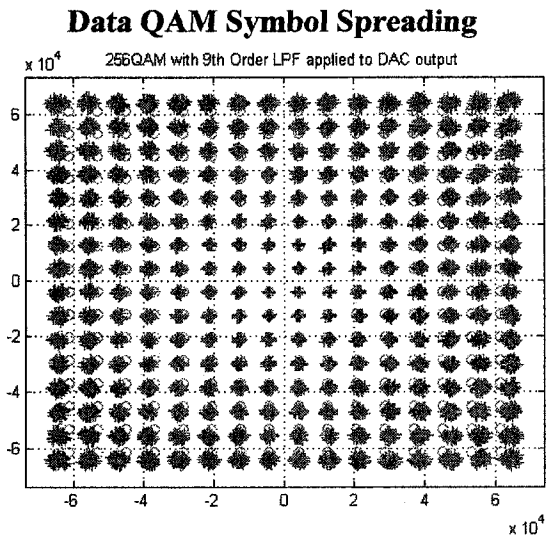


Figure 3-20. Data QAM Symbol Spreading on Constellation Caused by 9th Order LPF with the same specifications as the example output in figure 3-19a.

In this thesis, delay[17] between the I and Q channels of the OFDM baseband signal is of interest, while the amplitude mismatch is covered in the 'I/Q amplitude imbalance' case. In the simulation I/Q delay model, it assumes either the I or Q channel is sampled at the ideal time, while the other has an offset from the ideal instance because of the delay factor. More details on the sampling time are discussed in the following ADC section.

3.6.5 Receiver Input: ADC

In the hardware equivalent software simulation model, resolution of the ADC in the software model replicates the ADC characteristics found in the HP54542A scope by providing 8-bit of resolution. In further investigations using pure software simulations, resolution is isolated as a factor, and the ADC varies between 6 to 11-bits.

In simulations, re-sampling the baseband signal at the receiver is straightforward. Software provides complete control of sampling time instances, thus unlike the hardware experiments, over-sampling the waveform[18] is not required. In all simulation cases, except when relative delay factor is examined, the received signal is sampled at time $t=\frac{T_{OFDM_{sym}}}{2}$. When simulation include relative delay factor between the I and Q channel, one channel is sampled at the ideal time $t=\frac{T_{OFDM_{sym}}}{2}$, while the other is sampled at an offset from the ideal time.

3.6.6 Demodulation

Similar to the hardware demodulation, the software model applies the same procedure to demodulate the data. These demodulation steps are listed in order as:

[17] Parallel analog filters in a typical system have the potential to cause relative I/Q delay and amplitude mismatches caused by tolerance variance in manufacturing of the components.
[18] In simulation, the waveform is discrete, but spline interpolation is applied to produce a continuous time signal.

1. Data samples to undergo demodulation range approximately from –2 V to 2 V, but not exact if the DAC in transmitter not at ideal 4Vpp and/or because of quantization noise due to ADC.
2. Samples used for demodulation are selected at the center of each OFDM sample period i.e. $t=\frac{T_{OFDM_{sym}}}{2}$, unless delay factor is introduced.
3. Reverse scaling and reverse offset is performed on the data to undo all manipulations inflicted in the transmitter post-IFFT.
4. MSE_{wave} is computed with de-scaled sampled values and theoretical values stored from modulation, at the IFFT output.
5. FFT is applied to the de-scaled samples.
6. QAM symbols are verified with the theoretical values stored from modulation, after constellation mapping. This provides an absolute value on the number of QAM symbol errors that have occurred in the simulation.
7. Using both the 'demodulated QAM symbol locations' and 'theoretical QAM symbol location values' saved from modulation, MSE_{cst}, and SIR_{bit} are computed

3.7 Figure of Merit

As discovered from the hardware experiments and corresponding simulations, noise equivalent sources degrade demodulation performance. To measure and gauge how much performance degradation the system has experienced, different figures of merit are computed and are discussed in this section to help quantify the results.

To begin, the result of all simulation models are evaluated at two points in the system. The first point is the analog waveform in the receiver to determine the amount of distortion in the waveform. The second point of interest is on the constellation, after FFT

demodulation to determine how distortion due the analog factors translates to interference noise that displaces constellation points away from its ideal demodulation location. Equations on how each figure of merit is calculated is presented and discussed in the following subsections.

3.7.0 Waveform MSE

The purpose of finding the mean squared error (MSE) of the waveform is to determine distortion of the analog waveform before demodulation. Performing such calculations provides a meaningful method to quantify distortion error in a fundamental form. All factors including DC offset, gain error, I/Q amplitude imbalance, ADC saturation, quantization error from limited resolution in converters, and channel filter delay contributes to distortion. Using ideal waveform sampled values and theoretical values, the MSE is computed with the following equations 3-6 and 3-7.

The MSE for any continuous time domain waveform given by:

$$MSE_{wave} = \frac{\int_0^T \left| x_{act}(t) - x_{ideal}(t) \right|^2}{\int_0^T \left| x_{ideal}(t) \right|^2} \tag{3-6}$$

Where:

- t is a variable representing time
- T is the end-time of the waveform used for computation
- x_{act} is the continuous waveform received at the receiver
- x_{ideal} is the ideal continuous waveform the receiver would receive without any distortion

In the discrete domain, the MSE for any sampled waveform is given by:

$$MSE_{wave} = \frac{\sum_{n} |x_{act}(n) - x_{ideal}(n)|^2}{\sum_{n} |x_{ideal}(n)|^2} \quad (3\text{-}7)$$

Where:

- n is the sample index of the waveform. n=1,2,3.....
- x_{act} is the sampled waveform received at the receiver
- x_{ideal} is the ideal sampled waveform the receiver would receive without distortion

3.7.1 Constellation MSE, SIR, SIR_{bit}

On the constellation, using data QAM points, the MSE, SIR, and SIR_{bit} are computed as figure of merits used to gauge performance. All figures of merit mentioned here are related to each other, only mathematically manipulated to be different in value, but follow the same trend with respect to the data.

For m number of OFDM symbols, MSE of the constellation at the FFT output is given by:

$$MSE_{cst} = \frac{\sum_{m}^{K} \sum_{n=1}^{N} \left| S_{n,m-actual\_RX} - S_{n,m-ideal\_TX} \right|^2}{K \cdot N \cdot |d_{\min}|^2} \quad (3\text{-}8)$$

Where:

- N is the IFFT/FFT size
- K is the number of OFDM symbols
- m is the OFDM symbol index. m = 1,2,3....K
- n is the QAM point index . n = *1,2 ,3...N*

- $S_{n\text{-}ideal\_TX}$ is the ideal coordinates of n^{th} constellation QAM point transmitted within the m^{th} $OFDM_{sym}$.
- $S_{n\text{-}ideal\_RX}$ is the received coordinates of n^{th} constellation QAM point received within the m^{th} $OFDM_{sym}$.
- d_{min} is the minimal distance between any two points on the constellation

Another of figure of merit computed from the constellation data points is the signal-to-interference ratio. Ideally, the SIR would be infinite if there would be no noise sources in the system to displace the data points on the constellation from the ideal constellation symbol coordinates. The equations to find SIR for K number $OFDM_{sym}$ is given by:

$$SIR = 10LOG_{10} \frac{\sum_{m=1}^{K} \sum_{n=1}^{N} \left| S_{n,m-ideal\_TX} \right|^2}{\sum_{m=1}^{K} \sum_{n=1}^{N} \left| S_{n,m-actual\_RX} - S_{n,m-ideal\_TX} \right|^2} \quad (3\text{-}9)$$

Where:

- N is the IFFT/FFT size
- K is the number of OFDM symbols
- m is the OFDM symbol index. $m = 1,2,3....K$
- n is the QAM point index . n = 1,2 ,3...N
- $S_{n\text{-}ideal\_TX}$ is the ideal coordinates of n^{th} constellation QAM point transmitted within the m^{th} $OFDM_{sym}$.
- $S_{n\text{-}ideal\_RX}$ is the received coordinates of n^{th} constellation QAM point received within the m^{th} $OFDM_{sym}$

Similar to the SIR equation, but normalized to the number of bits in the constellation-mapping scheme, the SIR with respect to a bit for K number of $OFDM_{sym}$ is given by:

$$SIR_{bit} = 10LOG_{10}\left(\frac{\sum_{m=1}^{K}\sum_{n=1}^{N}\left|S_{n,m-ideal\_TX}\right|^2 / J}{\sum_{m=1}^{K}\sum_{n=1}^{N}\left|S_{n,m-actua\_RX} - S_{n,m-ideal\_TX}\right|^2}\right) \quad (3\text{-}10)$$

Where:

- N is the IFFT/FFT size
- K is the number of OFDM symbols
- J is the number of bits with respect to M-QAM constellation size, such that $2^J=M$
- m is the OFDM symbol index. m = 1,2,3....K
- n is the QAM point index . n = 1,2 ,3...N
- $S_{n\text{-}ideal\_TX}$ is the ideal coordinates of n^{th} constellation QAM point transmitted within the m^{th} $OFDM_{sym}$.
- $S_{n\text{-}ideal\_RX}$ is the received coordinates of n^{th} constellation QAM point received within the m^{th} $OFDM_{sym}$.

3.7.2 Degradation Curves

Besides the interference from the non-ideal analog factors, thermal noise is another factor in any communication systems. To relate the interference caused by the non-ideal analog factors with the thermal noise factor, degradation plots are created to observe the degradation trend due to interference from non-ideal factors for different signal-to-noise scenarios. Each degradation curve expressed in decibels, is extracted from the ratio of the possible "signal to interference-plus-thermal noise" to the "signal-to-noise ratio", as given by:

$$\left[\frac{S/[N_o+I]}{S/N_o}\right]=\left[\frac{1}{1+\frac{S/N_o}{S/I}}\right] \tag{3-11}$$

Once extracted from equation 3.11, the degradation factor is given by:

$$Deg=\left[1+\frac{S/N_0}{S/I}\right] \tag{3-12}$$

Where:

- N_o is the average noise power spectral density (W/Hz) of the thermal noise (white Guassian distributed)
- I is the average interference noise power
- S is the average signal energy per bit

Chapter 4

Simulation Results

4.0 Overview

This chapter is divided into two main sections. First, section 4.1 presents the simulation results obtained using the hardware setup. These results are derived from the empirical data gathered and then evaluated from the hardware experiments preformed, which are then matched by the software simulation model. The next section 4.2 deals with the results obtained using only the software simulator. In each of the following subsections from 4.2.0 and 4.2.4, simulations results are presented for scenarios isolating each analog factor of interest, and are presented in their sequential order:

1. DC offset error in received data samples, with/without ADC saturation
2. I/Q amplitude balanced gain error at receiver, with/without ADC saturation
3. I/Q amplitude imbalance gain error at receiver
4. Varying ADC Resolution at receiver
5. Relative I/Q delay sampling at receiver's ADC, with low pass filtering before ADC.

Last of all, section 4.2.5 deals with the results of a combination of the above five 'isolated factors' listed. Several simulations in this section are performed by varying

relative I/Q delay factor, while the other four factors listed above remain at a fixed value.

4.1 Hardware Experimental Results

4.1.0 Experimental Setup Overview

Before divulging in to the results, a brief overview entails the details on the steps taken from the beginning to the end of the hardware experiment.

A hardware experiment performed involves the following order of steps:

1. PN data sequence is generated on the Altera CPLD using digital logic that is synthesized to a linear feedback register
2. The binary data is transferred to the DSP via a serial port interface at a rate of 25.175Mbps
3. A total of 128 32-bit words (4096 bits) is captured and modulated on the floating-point DSP[19] to generate 10 OFDM symbols
4. To begin the data processing, the DSP groups the received binary data to 6-bit words and store them 32-bits memory locations
5. On a 64-QAM constellation, the grouped 6 bit words are mapped into symbols values
6. Once mapped, the key function to OFDM modulation is applied to data symbols; a 64-point IFFT is applied to 64 symbols, which equates to 64 orthogonal carriers.
7. The IFFT outputs consist of the I/Q components. These components are scaled to accommodate the available 12-bit resolution on the DACs used
8. From the DSP, the I/Q components are transferred out in a serial manner to the serial-to-parallel data converter on the CPLD, which separates the I/Q

[19] Results are not dependent on the on the DSP model used (Texas instruments C6711) and any other floating-point DSP could be substituted in the system to perform the modulation without changing the results found. Also, any future revisions to the present C6711 will unlikely to impact to results, since revisions usually change processor speed, chip size, cost, and power consumption, but the computational results would not change.

components and provides serial-to-parallel conversion before sending the data to the I/Q DACs

9. The I/Q DACs translate the digital words to its corresponding analog levels, thus constructing the continuous OFDM time waveform
10. From the DAC outputs, the analog signals are applied to a 4-channel digital sampling scope, only using two of the four channels.
11. At a rate of 25MSPS, the scope captures the I/Q signals in 32768K bytes data space (enough for all 10 OFDM symbols) at a time and stores the data, which is transferred to a PC for demodulation in Matlab software.
12. In Matlab, before demodulation, the data samples from the scope are decimated down from 23.8-to-1 ratio, such that there is one data sample per OFDM sample.
13. After decimation, the data samples undergo demodulation that begins with reverse scaling, and then the FFT function is applied to recover the QAM data points.
14. To generate more data, the complete hardware system resets, and the CPLD is reconfigured with another LFSR structure. Then steps 1 to 13 are repeated to gather results for 20 distinct OFDM symbols, which equates to one data set.

In each data set, the MSE_{wave}, MSE_{cst}, and SIR_{bit} computation is performed to measure the quality of the signal received and its ability to accurately demodulate data QAM symbols on the constellation. With a collection of empirical data sets, MSE_{wave}, MSE_{cst}, and SIR_{bit} values are averaged. These averages are then reproduced in software by matching each error source values. With near matching results in software, the following factors including: DC offset, gain error, I/Q imbalance, and DAC/ADC resolution is quantified numerically and individually as contributors of error sources. Also, a visual inspection of the constellation plots between simulation and experiments is used as a method of verification. By recreating the hardware values, the software simulation cross-verifies the presence of the specific analog factors mentioned above and the accuracy of the software simulation model.

Important hardware experimental configuration details are provided in table 4-1, while table 4-2 summarizes the processing details in the experiments.

Table 4-1. Hardware Experimental Setup Detail Summary

Item	Details
Analog Impact examined in system	1. DC offset caused by: ▪ Internal DAC offset error ▪ Calibration inaccuracies 2. I/Q amplitude imbalances caused by DAC output mismatches, thus leading to gain mismatches. 3. Limited DAC/ADC resolution of 12-bit DACs and 8-bits ADCs.
Number of OFDM symbols per data set	20 OFDM symbols
Number of experimental sets performed	15
Total Number of OFDM symbols used throughout the experiments	300 OFDM Symbols
Constellation Size	64QAM
IFFT/FFT Size	64
DAC resolution	12-bits
DAC output swing	4Vpp+calibration error
Receiver demodulation expects DAC output swing	4Vpp (To undo scaling imposed by I/Q DACs)
DAC output data rate	1.049Mbps
ADC Receiver Resolution[20]	8-bits
Receiver sampling rate[21]	Sampling at 25 MSPS

[20] This refers to the resolution of the HP54542A sampling oscilloscope
[21] Refers to the sampling rate used by the HP54542A sampling oscilloscope

Table 4-2. Hardware Experimental Processing Details

Item	Details
Sample Decimation	Only sample closest to $\frac{T_{OFDM_{sym}}}{2}$ used to calculate all performance merits (MSE_{cst}, MSE_{wave}, SIR_{bit}).
MSE_{cst}	This is computed post FFT using the demodulated QAM symbols and corresponding reference QAM symbol (values from transmitter block after constellation mapping) *Please refer to figure 4-1 to data location '2' used for calculations.*
MSE_{wave}	This is computed using de-scaled sampled values from ADC compared to ideal waveform levels directly from IFFT outputs in transmitter without scaling. *Please refer to figure 4-1 to data location '1' used for calculations.*
SIR_{bit}	Computation data is same as MSE_{cst} *Please refer to figure 4-1 to data location '2' used for calculations.*
QAM Errors	All demodulated QAM symbols and bits are verified to be correct with transmitted sequence. This includes the first FFT output bin, which represents the DC component of the OFDM time waveform.
Miscellaneous	• No coding • No cyclic prefix • No pilot channels

4.1.1 Empirical Data Results and Matching Hardware Results in Software

In this section, a brief explanation is presented as to how the DAC settings and DC offset error in software simulations are determined to match the hardware system. The target of the matching exercise is to have software model matched to the hardware setup, based on the MSE and SIR_{bit} averages shown in table 4-3.

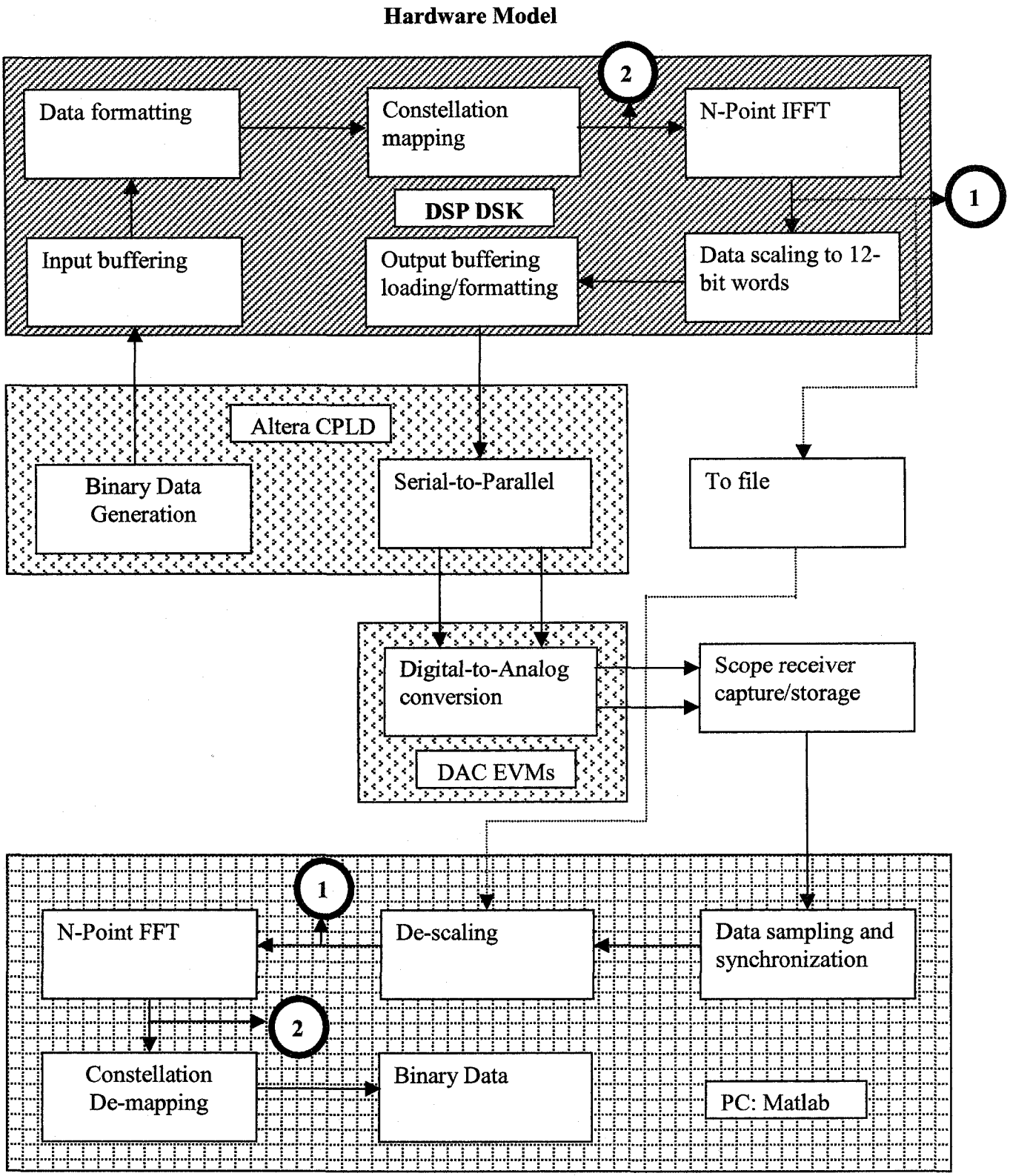


Figure 4-1. Hardware Model

Number labels map the location the data value used in computation (refer to table 4-2)

Table 4-3. Summary of Hardware Experiment Results

Description	MSE Waveform	MSE Constellation	SIR_{bit}
	[%]	[%]	[dB]
Empirical average of all data sets	0.081	0.82	23.11
Worse performance values among all data sets	0.089	0.9470	22.74
Best performance values among all data sets	0.076	0.5406	23.39

To begin the matching process, 8-bits of resolution receiver is set in the software model, with no DC offset, and perfect DAC 4Vpp setting. The simulation results from this setup show SIR_{bit}, MSE_{cst}, and MSE_{wave} values better than the hardware empirical averages. To degrade performance to reach a closer match, both DAC output swings were adjusted away from its ideal output levels.

In table 4-4, results of various DAC output combinations are tabulated and are used to help estimate a region of DAC values that maybe possible for matching, before introducing DC offset. In the table, two best cases are highlighted (Case 1: DAC I/Q output are 4.09V & 4.10V; Case 2 DAC I/Q outputs are 3.91V & 3.91V), based on best performance match with hardware empirical averages for either possible positive or negative gain error in the hardware system. With the two starting points, positive and negative DC offset is added by applying an estimated value based on the voltage step[22] value of an ADC word, then tuned to refine the match. At the moment, the DC offset is assumed balanced in both the I and Q channel, but later adjusted to demonstrate it is unbalanced.

[22] This value is given by: $ADC\ Voltage\ Step = \frac{2A}{2^N} = \frac{2*2V}{2^8} = 0.0316V$, where A is the input peak amplitude limit allowed by the ADC, and N is the number of resolution bits.

Table 4- 4. **Results for Hardware System's Equivalent Simulation**

SW: Max Output Voltage Swing		MSE Waveform		MSE Constellation		SIR-bit	
Q Channel DAC	I Channel DAC	MSE Waveform in SW simulation	% Difference between SW and HW empirical average	MSE Constellation in SW simulation	% Difference between SW and HW empirical average	SIR-bit	% Difference between SW and HW empirical average
[Vpp]	[Vpp]	[%]	[%]	[%]	[%]	[dB]	[%]
4.08	4.05	0.05	35.43	0.55	32.98	24.89	7.71
4.08	4.06	0.06	31.36	0.59	28.48	24.61	6.49
4.08	4.07	0.06	26.17	0.63	22.88	24.28	5.07
4.08	4.08	0.06	20.99	0.68	17.34	23.98	3.77
4.08	4.09	0.07	14.81	0.73	10.73	23.65	2.32
4.08	4.10	0.07	8.77	0.79	4.20	23.34	0.99
4.09	4.05	0.06	28.15	0.59	27.85	24.41	5.64
4.09	4.06	0.06	24.07	0.63	23.50	24.16	4.55
4.09	4.07	0.07	18.89	0.67	18.13	23.86	3.26
4.09	4.08	0.07	13.83	0.71	13.09	23.59	2.08
4.09	4.09	0.07	7.65	0.76	6.72	23.28	0.75
4.09	4.10	0.08	1.73	0.82	0.51	23.00	0.47
4.10	4.05	0.06	20.12	0.68	16.57	23.94	3.59
4.10	4.06	0.07	16.05	0.72	12.07	23.71	2.60
4.10	4.07	0.07	10.86	0.77	6.46	23.44	1.44
4.10	4.08	0.08	5.80	0.81	0.94	23.19	0.36
4.10	4.09	0.08	0.25	0.87	5.68	22.91	0.85
4.10	4.10	0.09	6.30	0.92	12.21	22.65	1.98

Table 4-4 Continued. **Results for Hardware System's Equivalent Simulation**

SW: Max Output Voltage Swing		MSE Waveform		MSE Constellation		SIR-bit	
Q Channel DAC	I Channel DAC	MSE Waveform in SW simulation	% Difference between SW and HW empirical average	MSE Constellation in SW simulation	% Difference between SW and HW empirical average	SIR-bit	% Difference between SW and HW empirical average
[Vpp]	[Vpp]	[%]	[%]	[%]	[%]	[dB]	[%]
3.90	3.90	0.09	15.19	0.69	16.41	23.91	3.48
3.90	3.91	0.09	9.63	0.61	25.06	24.39	5.53
3.90	3.92	0.08	2.96	0.56	32.32	24.83	7.45
3.90	3.93	0.08	2.35	0.52	36.96	25.14	8.78
3.90	3.94	0.07	9.38	0.49	40.17	25.37	9.77
3.90	3.95	0.07	14.57	0.46	43.49	25.61	10.84
3.91	3.90	0.09	6.91	0.84	2.46	23.05	0.27
3.91	3.91	0.08	1.48	0.80	2.66	23.27	0.69
3.91	3.92	0.08	5.19	0.75	8.77	23.55	1.91
3.91	3.93	0.07	10.37	0.71	13.65	23.79	2.94
3.91	3.94	0.07	17.53	0.65	20.24	24.14	4.44
3.91	3.95	0.06	22.59	0.62	24.99	24.40	5.59
3.92	3.90	0.08	1.85	0.77	5.70	23.41	1.29
3.92	3.91	0.08	7.28	0.73	10.82	23.65	2.34
3.92	3.92	0.07	13.83	0.68	16.93	23.96	3.67
3.92	3.93	0.07	19.01	0.64	21.80	24.22	4.81
3.92	3.94	0.06	26.17	0.59	28.40	24.60	6.47
3.92	3.95	0.06	31.23	0.55	33.15	24.90	7.75

DC offset has a more significant impact on MSE_{wave}, than on MSE_{cst}, and SIR_{bit}. The gain error and I/Q imbalances has the opposite effect i.e. its effect is more pronounced on MSE_{cst} and SIR_{bit} than MSE_{wave}. With this known fact, the gain error and DC offsets are strategically tweaked to match the experimental empirical averages as close as possible. In summary, from the matching process, four possible scenarios are drawn and are presented in table 4-5. Each possible case is a possible representation of what is occurring in the hardware system, but between the four there is not enough evidence to conclude which exact case is occurring[23].

Table 4-5. Summary of Hardware Matching Cases

Description	MSE Waveform [%]	MSE Constellation [%]	SNR bit [dB]
Hardware: Empirical average of all data sets	0.081	0.82	23.11
Software Possible Case 1 ▪ Q DAC 3.91Vpp ▪ I DAC 3.94Vpp ▪ DC offset 0.01V	0.0806	0.789	23.32
Software Possible Case 2 ▪ Q DAC 3.92 ▪ I DAC 3.94, ▪ DC offset -0.0065V	0.812	0.804	23.23
Software Possible Case 3 ▪ Q DAC 4.05Vpp ▪ I DAC 4.06Vpp ▪ DC offset 0.0091V	0.076	0.798	23.26
Software Possible Case 4 ▪ Q DAC 4.05Vpp ▪ I DAC 4.07Vpp ▪ DC offset -0.013V	0.079	0.824	23.13

[23] The ambiguity exist because the error in the calibration of the DAC output voltages and the 0V level in the received samples from the sample-scope cannot be determined to be either positive or negative from the measurements.

4.1.2 Constellation Matching Results

As a visual confirmation, figures 4-2a and 4-2b are constellation plots of the data set for 20 OFDM symbols for simulation case 2 and a hardware data set. Important observation details seen between the two constellations are the spread of the data QAM symbols around the reference locations and the displaced QAM symbols representing the DC component of the OFDM symbol. The spread factor in both constellations visually appears to be in close proximities, thus confirms the I/Q imbalance are within reason, since I/Q imbalance causes the spread as shown later in section 4.2.2 'Effects on I/Q Amplitude Imbalances'. Also by visual inspection, the QAM symbols representing the DC components are on both figures share similar displacements distances from its ideal reference locations. Note, the sign of the DC offset distortion in the I/Q channels is important in the displacement direction of the DC value QAM symbols. To match DC distorted QAM symbols on the constellation diagram in figure 4-2a, to the adjacent figure 4-2b, negative DC distortion is applied to the Q channel, while positive is applied to the I channel.

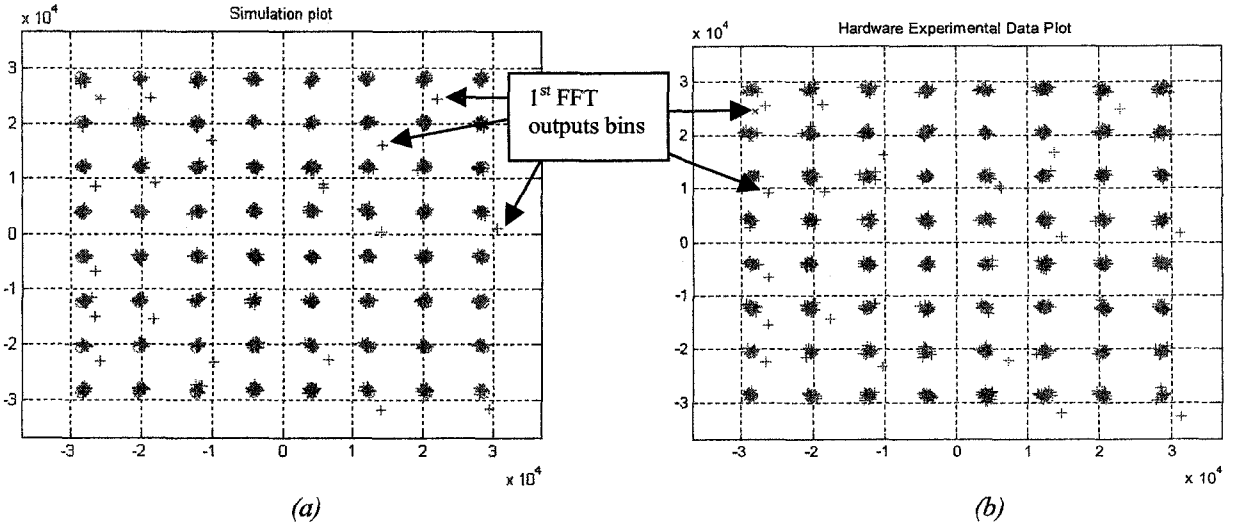


Figure 4-2. Simulation Plot *(a)* of Equivalent Hardware System Settings and Constellation Plot *(b)* of a Set of Data Collected in the Hardware Experiments

Simulation Plot (a) **Settings: Q DAC 3.91V, I DAC 3.94V, DC offset -0.01V(Q), +0.01 V (I). Each data set consist of 20 OFDM symbols, where the '+' markers represent the QAM data symbols from all 64 FFT output bins OFDM symbol.**

This unbalance DC offset error is due to the calibration mismatch error of the 0V reference on the scope, between the two DAC channels.

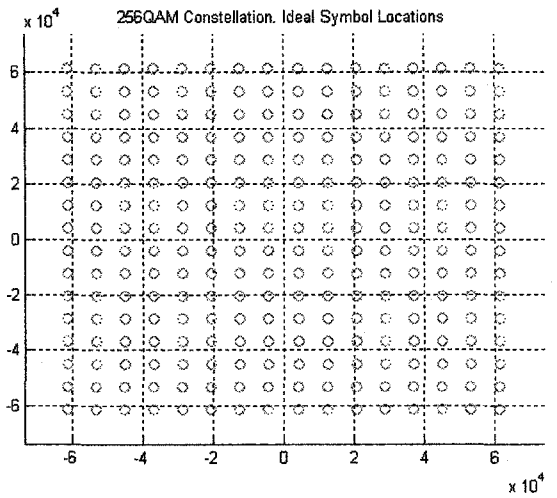


Figure 4-3. Demonstration of Ideal/Reference Symbol Locations on Constellation Plots

Scaling of both axis in the figure are integer multiples of 4096, which simplifies hex values used in the DSP C program. To help understand constellation plots presented before and in the rest of the results chapter, the reference symbol locations are marked with a 'o', as shown in figure 4-3, while data QAM symbols are marked with a '+' (not shown in this figure).

In figures 4-4 to 4-6, the hardware results have been plotted to show the empirical characteristics for the 15 trials performed. They provide a visual confirmation that the results have stable values between trials and there are no obvious protruding bad data set(s).

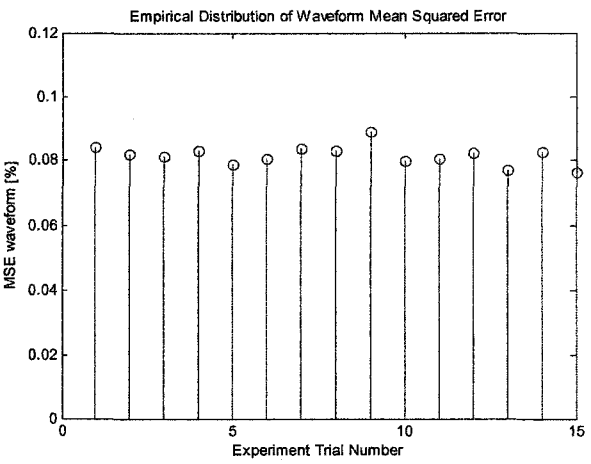


Figure 4-4. Empirical Plot of MSE_{wave}

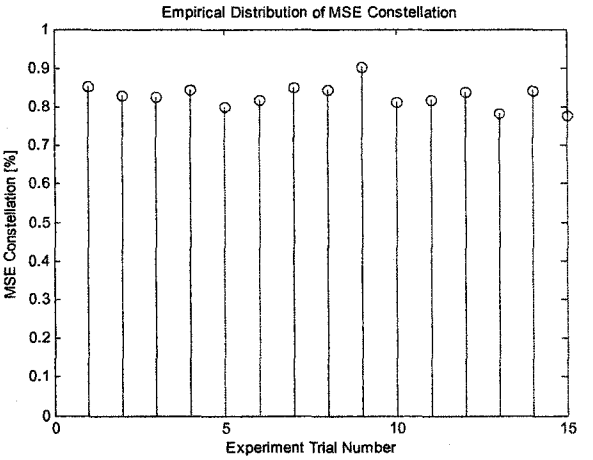


Figure 4-5. Empirical Plot of MSE_{cst}

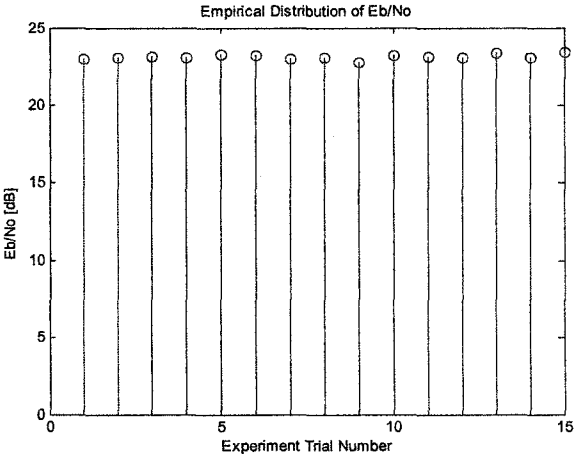


Figure 4-6. Empirical Plot of SIR_{bit}

One final remark about error distribution before ending the hardware results, the constellation error distribution on a histogram is not of a Guassian or any other known/obvious form, because DC offset, gain error, and I/Q imbalance each engender their own unique form. More details on each of these distributions are discussed in the isolated simulation cases, from sections 4.2.0 to 4.2.2.

4.2 Pure Software Simulations

This section presents the results from the various simulation cases performed purely in a Matlab software environment. The order of presentation begins with isolation cases of each analog factor, then a combination case of all analog factors combined. The specific order of results is as follows:

1. DC offset
    a. DC offset with no ADC saturation
    b. DC offset with ADC saturation
2. Gain error
    a. Gain error with no ADC saturation
    b. Gain error with ADC saturation
3. I/Q amplitude imbalances
4. Variable receiver resolution
5. Filtering phase delay causing relative I/Q delays
6. Combination case of all above factors listed

A refresh of the important simulation details are provided in table 4-6 and the corresponding model is presented in figure 4-7.

Table 4-6. General Software Simulation Details Applicable to all Cases

Description	Value/Details
OFDM symbols generated to constellation size ratio	A ratio of 200 Symbols per each reference constellation point, in each simulation.
Total Number of OFDM symbols	• 50 (16 QAM constellation) • 200 (64 QAM constellation) • 800 (256 QAM constellation)
IFFT/FFT	64
Time Instance of Receiver Sample	Exactly at $\frac{T_{OFDM_{sym}}}{2}$ used to calculate all performance metrics (MSE_{cst}, MSE_{wave}, and SIR_{bit}). Not applicable to delay simulations
MSE_{cst}	This is computed post FFT using the demodulated QAM symbols and noiseless QAM symbols values saved from the modulation section after constellation mapping. *Please refer to figure 4-7 to data location '1' used for calculations*
MSE_{wave}	This is computed using de-scaled sampled values from ADC compared to ideal waveform levels directly from IFFT outputs in transmitter without scaling. *Please refer to figure 4-7 to data location '2' used for calculations*
SIR_{bit}	Computation data is same as MSE_{cst} *Please refer to figure 4-7 to data location '1' used for calculations*
Miscellaneous	• No coding • No cyclic prefix • No pilot channels • No channel noise; data feed directly from transmit to receive

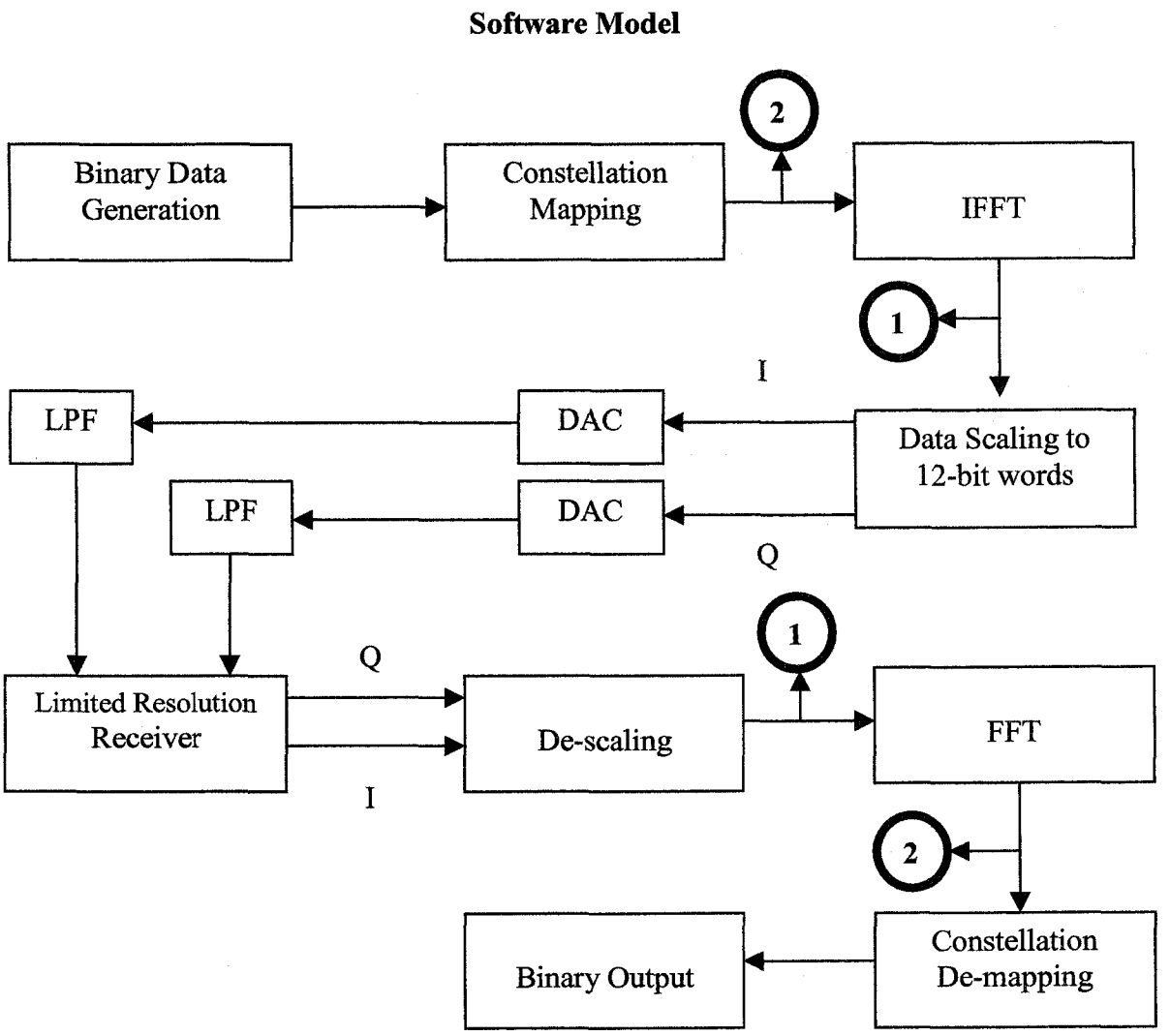


Figure 4-7. Software Model

Number labels map the location the data value used in computation (refer to table 4-6)

4.2.0 DC offset

Part I: DC Offset Analysis with no ADC Saturation

In this first part of the investigation, the OFDM system is probed for maximum DC offset[24] tolerance allowable in the received samples. To determine these tolerance values, the software simulation model removes all non-ideal factors in the system to isolate the

[24] Maximum allowable DC offset until QAM symbol errors occur, which belong to the first FFT output since they are sensitive to DC distortion.

effects of DC offset introduced at the receiver. Table 4-7 presents specific key setup details important to this simulation and table 4-8 summarizes the results for the maximum allowable DC offset in each constellation scheme before experiencing QAM symbol errors from the first FFT output bin. The simulations also verify the other FFT output bins are not susceptible to DC offset error, assuming the ADC has an infinite input range.

Table 4-7. First Setup Details for DC Offset Analysis

Item	Value or Detail(s)
Constellations	• 16 QAM • 64 QAM • 256 QAM
DAC resolution	12-bits
DAC output swing	I/Q fixed at 4Vpp
Receiver demodulation expects I/Q DAC output swing	4Vpp
DC offset	• Varied per each constellation simulation case to find its tolerable threshold before QAM symbol errors occur. When they do occur, they involve only the first FFT output bin/QAM symbol. • DC offset is introduced in a balanced manner in both I/Q channels.
ADC Resolution	Infinite. No information loss in the 12-bit DAC words received. Therefore DC offset does not affect the effective number of bits of the system, since saturation of the ADC does not occur due to DC offset.
Receiver Sample Decimation	At time instant $\frac{T_{OFDM_{sym}}}{2}$, which is at the sample closest to the center/between the beginning and end of an OFDM sample
LPF Filtering	None used

According to simulations results, using the same average power transmitted between the different constellation sizes, the larger constellation schemes are more sensitive to DC offset. To explain this phenomenon, the first point in the receiver FFT output representing the DC component in the OFDM symbol is sensitive because the DC offset imposes distortion onto this point.

Table 4-8. Results: Maximum DC Offset Levels Allowable Before QAM Symbol Errors[25]

	Maximum DC offset absolute value before QAM symbol errors	Maximum DC offset percentage before QAM symbol errors	MSE_{wave} @ Max. DC offset	MSE_{cst} @ Max. DC offset	SIR_{bit} @ Max. DC offset
	[V]	[%]	[%]	[%]	[dB]
16QAM	0.030	0.75	0.3039	0.7570	19.07
64QAM	0.011	0.28	0.0707	0.7412	23.72
256QAM	0.006	0.15	0.0164	0.7045	28.83

Thus this QAM symbol/point is displaced from its original reference point on the constellation, and potentially demodulated into the wrong region on the constellation. The tolerable QAM symbol displacement is less in the larger constellation schemes, because the minimal distance between any two-constellation points is shorter, making the larger constellation more sensitive to DC offset.

Figure 4-8 demonstrates the first points in each FFT output shifting away from the reference symbol location on the constellation as DC offset distortion increases. The FFT output that represents the DC point moves in a 45 degree diagonal manner on the constellation map since the gain error is equivalent in both I/Q channels that maps to the two dimensions on the constellation, real and imaginary, causing a equivalent displacement in both axis. If the first FFT output point is not considered in the results, then the data QAM symbols on the constellation are not susceptible to symbol error due to DC offset distortion, assuming the ADC has an infinite input range to eliminate the possibility of reduction of resolution.

[25] All FFT outputs in the receiver are considered in the results.

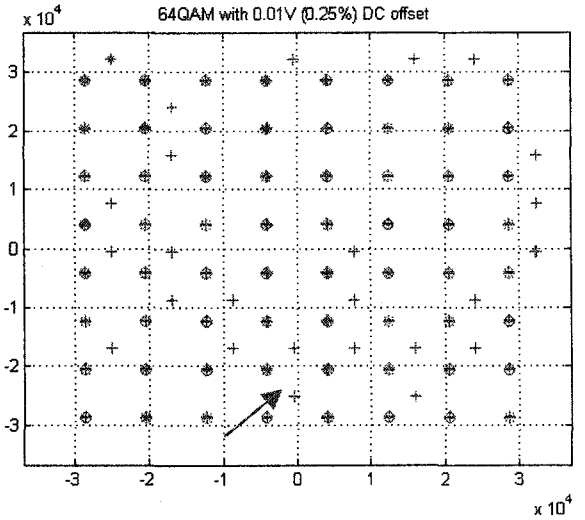


Figure 4-8. Illustration of First FFT Output Bin Displacement due to Balanced DC Offset

All QAM data markers not located in the ideal 'o' position are the QAM symbols representing the DC component after FFT taken. The arrow shows the 45-degree displacement direction of the QAM symbols representing the DC component in each OFDM symbol.

Again, in the simulations, the DC offset is assumed balanced in both the I/Q channels. In actual systems, there is a high likelihood the DC offset from the ADC is unbalanced, but the sign off the offset is likely to be the same due to the same transistor design between data converters in the I and Q branches. Also LO leakage would contribute to DC distortion to both the I and Q components during down conversion. In effect, the unbalanced DC factor displaces the first FFT output QAM symbol in different angle rather than the 45 degrees seen in the previous figure for balanced DC offset.

The error distribution on the constellation caused by DC offset only involves the QAM symbols from the first FFT output bin, while other bins experience zero error magnitude from their ideal coordinates. For static DC offset, the displacement magnitude on the constellation is constant from OFDM symbol-to-symbol. Therefore, on a histogram plot, the errors would all reside in the same bin, resulting in a single vertical line.

Part II: DC Offset Analysis with ADC Saturation

With the maximum values DC offset found before occurrence of QAM errors considering all FFT outputs, and verifying DC offset does not have any obvious impact on the rest of the FFT bins, this second investigation observes if the DC offset has any significant impact on demodulation if the ADC were to susceptible to input signal saturation. In table 4-9, the setup details for this investigation are disclosed.

Table 4-9. Second Setup Details for DC Offset with ADC Saturation Analysis

Item	Value or Detail(s)
Constellations	• 16 QAM • 64 QAM (Results show only for 64QAM) • 256 QAM
DAC resolution	12-bits
DAC output swing	I/Q fixed at 4Vpp
Receiver demodulation expects I/Q DAC output swing	4Vpp
DC offset	• Varied to maximum DC offset permitted before QAM errors as found and shown in table 4-8 for each constellation. • DC offset is introduced in a balanced manner in both I/Q channels.
ADC Resolution	9-bit ADC over 4Vpp input selected. Thus any DC offset could possibly affect the effective number of bits of the system by saturating the input range of ADC.
Receiver Sample Decimation	At time instant $\frac{T_{OFDM_{sym}}}{2}$
LPF Filtering	None used

The results shown in figure 4-9 for 64QAM, show the serious SIR_{bit} degradation when all FFT output bins are accounted for. However, excluding the first bin results in a constant SIR_{bit}, since the first FFT output bin is very sensitive to DC offset. Therefore, for the magnitude explored, the DC offset error does not have any obvious impact to SIR_{bit}. This

is because the ENOB reduction is minuscule and the OFDM sampled waveform at the ADC occasionally saturates due to its distribution that lies mostly within the ADC input range. Results for the other two constellations are not shown, since the results are similar to 64QAM, shown below.

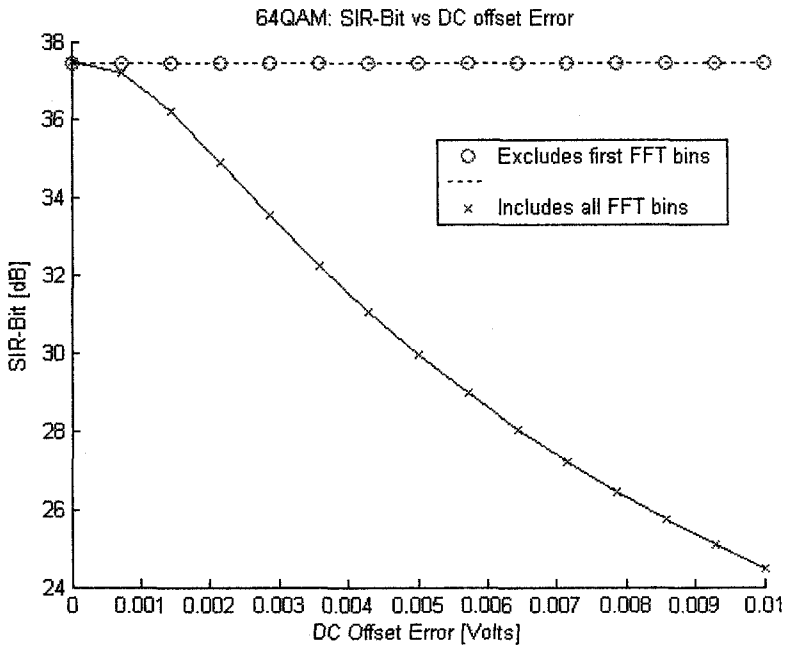


Figure 4-9. 64QAM: SIR_{bit} vs DC Offset Error

4.2.1 Effects of Gain Error

Part I: Gain Error without ADC Saturation

This section discusses the results and elaborates on the effects of the I/Q balanced amplitude mismatch error between the DAC output levels and the receiver's demodulation constant used to undo the DAC scaling. Mismatch between the I and Q channels generate a gain error in the signal. This method of simulating gain error can represent various sources in an OFDM transceiver such as LO drive level error, inability for perfect AGC compensation, and ADC gain error. Therefore, these investigations observe the effects of improper compensation for the newly scaled constellation caused by gain error.

Simulations performed in gain error investigations isolate the balanced DAC to receiver mismatch effects from all other analog factors in the system. Output voltage swings of the DACs are varied together for both I and Q channels, to control gain error magnitude imposed onto the system. The input range of the ADC is set to 8V with infinite resolution, to avoid input saturation due to positive gain error. Table 4-10 summarizes key configuration details specific to this investigation.

Table 4-10. First Setup Details for Gain Error Analysis

Item	Value or Detail(s)
IFFT/FFT size	64
Figure of Merits	MSE_{cst} and SIR_{bit} computation do not include first FFT output bin, since they are sensitive to DC offset errors and are not of interest assuming the first FFT output bin does not carry data.
DAC resolution	12-bits
DAC output swing	I/Q Varied around 4Vpp together
Receiver demodulation expects I/Q DAC output swing	4Vpp
DC offset	0 V
ADC Resolution	Infinite. Therefore no information loss from the 12-bit DAC words received due to saturation.
Receiver Sample Decimation	At time instant $\frac{T_{OFDM_{sym}}}{2}$
LPF Filtering	None used

By introducing gain error in the system, the data demodulates with reference to a scaled constellation. To be precise, constellation scaling occurs when the QAM symbols on the constellation shift as if the ideal/original reference constellation points contracted or expanded.

Using +/- 10% gain error values, an illustration of constellation scaling is provided in Figures 4-10 and 4-11. They demonstrate how the constellation symbols expand and contract due to positive and negative gain errors. If the constellation should be scaled by gain error, the newly scaled constellation needs the receiver to accurately divide up the demodulation regions on the constellations as precisely as possible. The division of

regions is based on bisecting distances between points on the constellations. However, if the gain error is uncompensated, where the divisional boundaries on constellation refer to an incorrect constellation scale, the demodulated data symbols are displaced by interference from the gain factor.

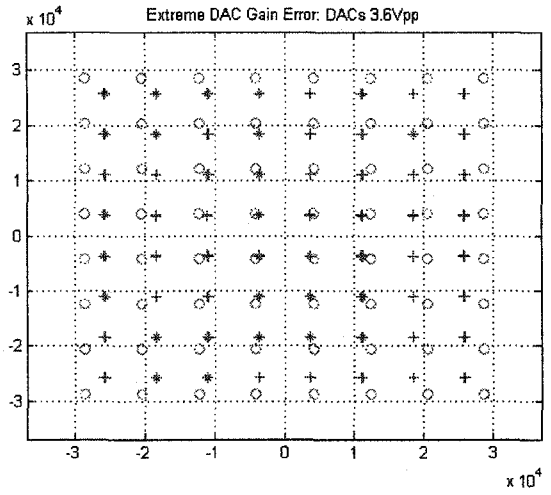


Figure 4-10.
Constellation Illustration of -10% Gain Error

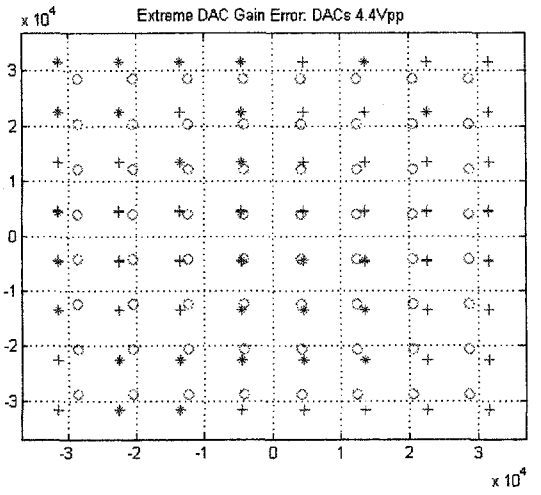


Figure 4-11.
Constellation Illustration of +10% Gain Error

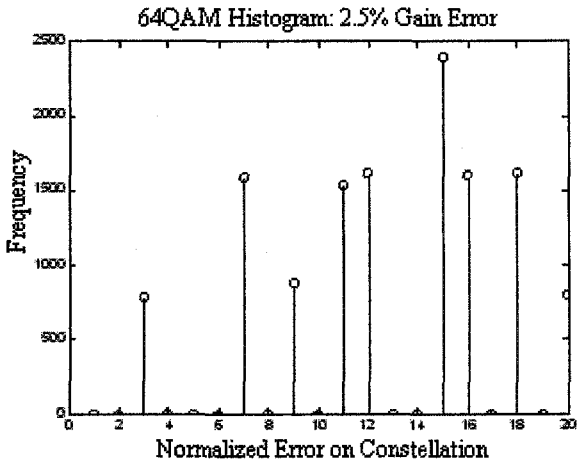


Figure 4-12. Histogram of Normalized Errors for 64QAM: 2.5% Gain Error

Uncompensated gain creates a predictable constellation error distribution on the histogram because the error is a function of the distance between the new and old constellation. Using 64QAM as an example, the distribution is plotted in figure 4-12, proving that it is not Guassian or any other common distribution curve commonly known.

Observing the mean square error in figure 4-13, the MSE_{wave} versus gain error is plotted in a monotonic growing manner with respect to gain error magnitude increase.

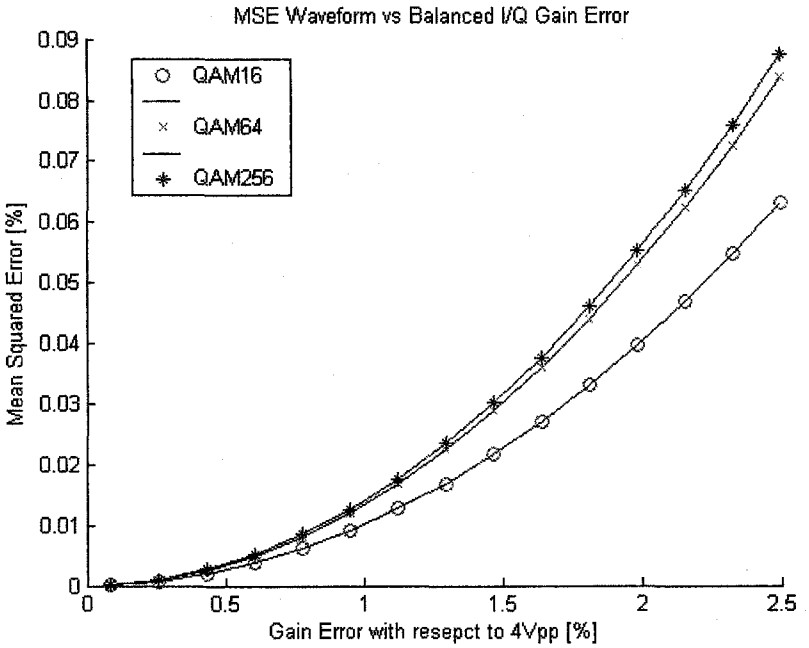


Figure 4-13. MSE_{wave} versus Gain Error

While the MSE_{wave} monotonically increases with gain error that is uncompensated in the receiver, MSE_{cst} and SIR_{bit} performance degrades as shown in figures 4-14 and 4-15.

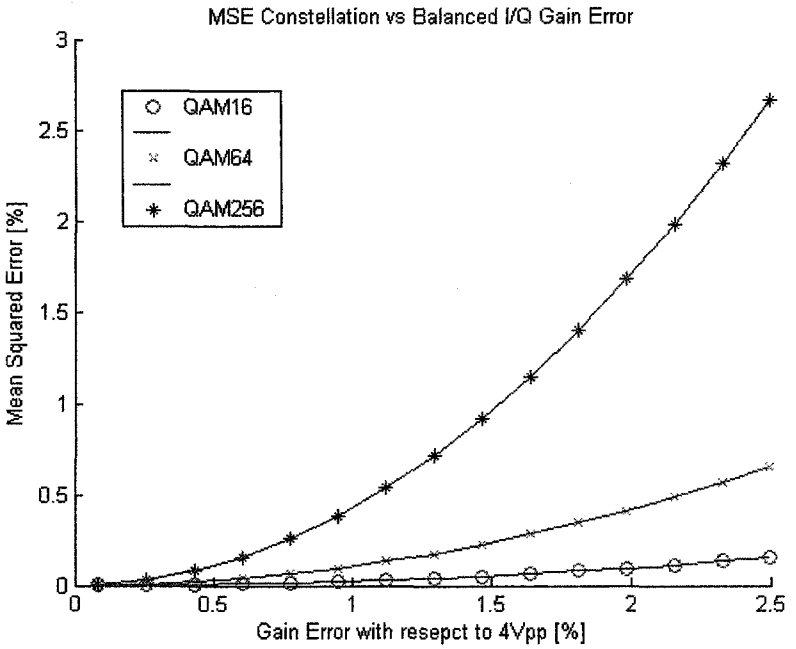


Figure 4-14. MSE_{cst} versus Gain Error

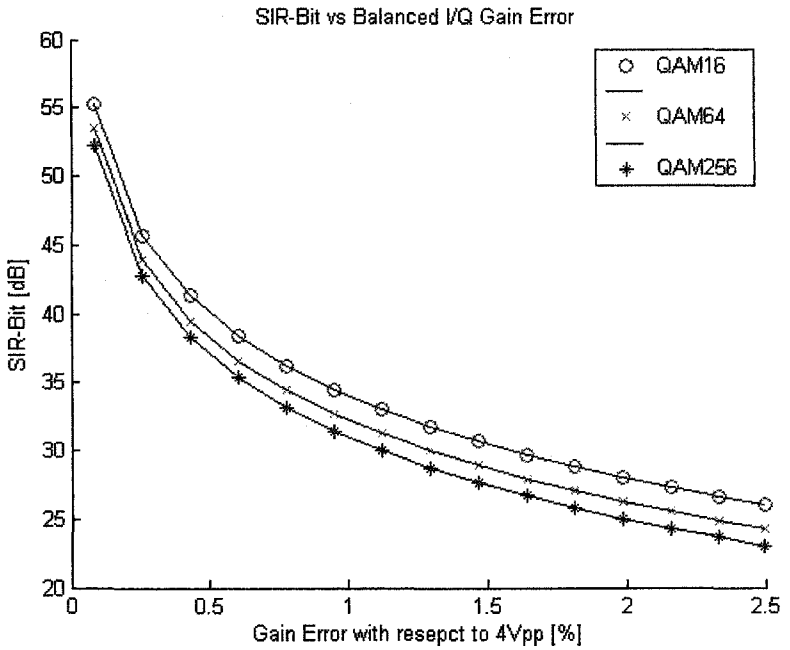


Figure 4-15 SIR_{bit} versus Gain Error

In summary, the results for gain error show the significant demodulation performance benefits for minimizing gain error.

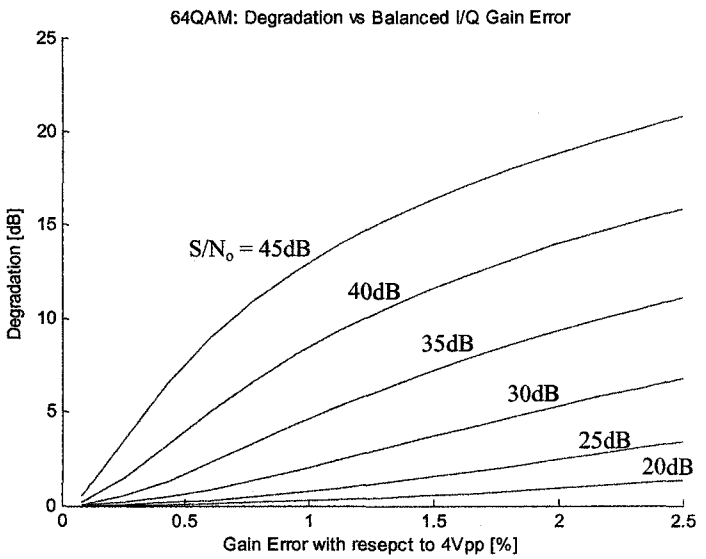


Figure 4-16. 64QAM: Degradation versus Gain Error

Figure 4-16 shows the degradation curves for S/N_0 values from 20dB to 45dB. Shown in this figure, the degradation becomes more severe with respect to increasing gain error/interference noise, experienced in the higher S/N_0 curves.

Part II: Gain Error with ADC Saturation

In a second setup used to observe the impact of uncompensated gain error in both I/Q channels, which assumes the full ADC range is utilized before gain error occurs. With such an assumption, it is interesting to investigate if the gain error saturating the ADC input has a significant impact. To perform the investigation the following setup details in table 4-11 are applied to the model.

For the gain error magnitudes explored, the results shown in figure 4-17 do not show any significant impact of ADC saturation due to uncompensated gain error on SIR_{bit} performance. To justify using same reason as found in 'ADC saturation by DC offset', the sampled OFDM time waveform is not impacted by the saturation because it does not

Table 4-11. Second Setup Details for Gain Error Analysis with ADC Saturation

Item	Value or Detail(s)
IFFT/FFT size	64
Figure of Merits	MSE_{cst} and SIR_{bit} computation do not include first FFT output bin, since they are sensitive to DC offset errors and are not of interest assuming the first FFT output bin does not carry data.
DAC resolution	12-bits
DAC output swing	I/Q Varied around 4Vpp together
Receiver demodulation expects I/Q DAC output swing	4Vpp
DC offset	0 V
ADC Resolution	9-Bits. Therefore saturation occurs when I/Q DAC output voltages exceed 4Vpp.
Receiver Sample Decimation	At time instant $\frac{T_{OFDM_{sym}}}{2}$
LPF Filtering	None used

always consume its full peak-to-peak range, assuming no intentional clipping involved at the DAC/transmitter. In other words, the OFDM time signal distribution mainly lies within the ADC range for the majority of the time, and occasionally extends to peak values. When it does reach its peak values, there is distortion from clipping if the ADC input range is not wide enough, but clipping has minimal effect for the magnitudes of gain error explored in this section.

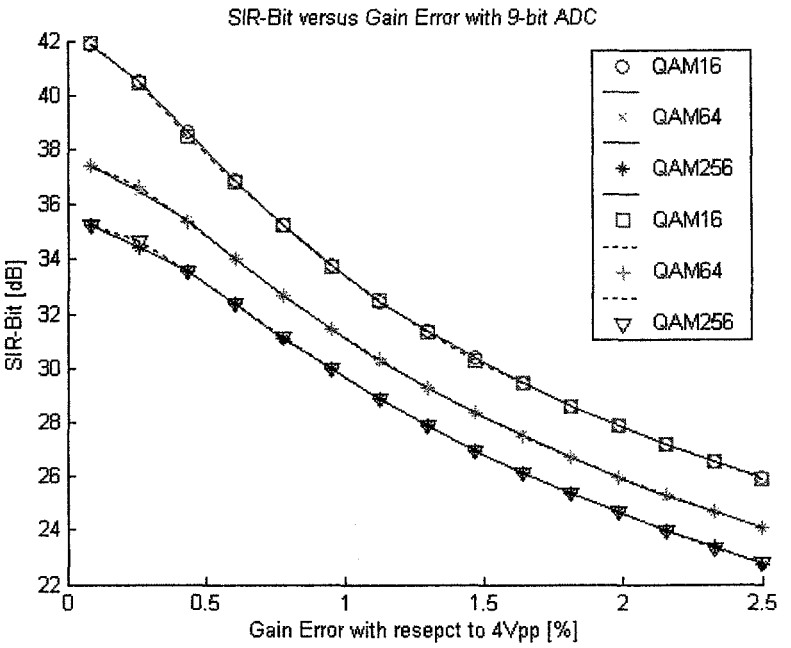


Figure 4-17. Impact of ADC Saturation due to Gain Error: SIR_{bit} vs Gain Error

The first three symbols in the legend box and the solid lines represents simulations performed without encountering ADC saturation limitations. The last three symbols in the legend box connected by dashed lines represent simulations with a limited 4Vpp input range to the ADC, and saturation is a factor.

4.2.2 Effects on I/Q Amplitude Imbalances

In this simulation case, I/Q amplitude imbalances are caused by mismatched DAC outputs, which are not compensated in the receiver. Thus the amplitude imbalance results in a gain error at the receiver, in either one or both channels. These I/Q amplitude imbalance results are applicable and simulate various non-ideal sources in the receiver.

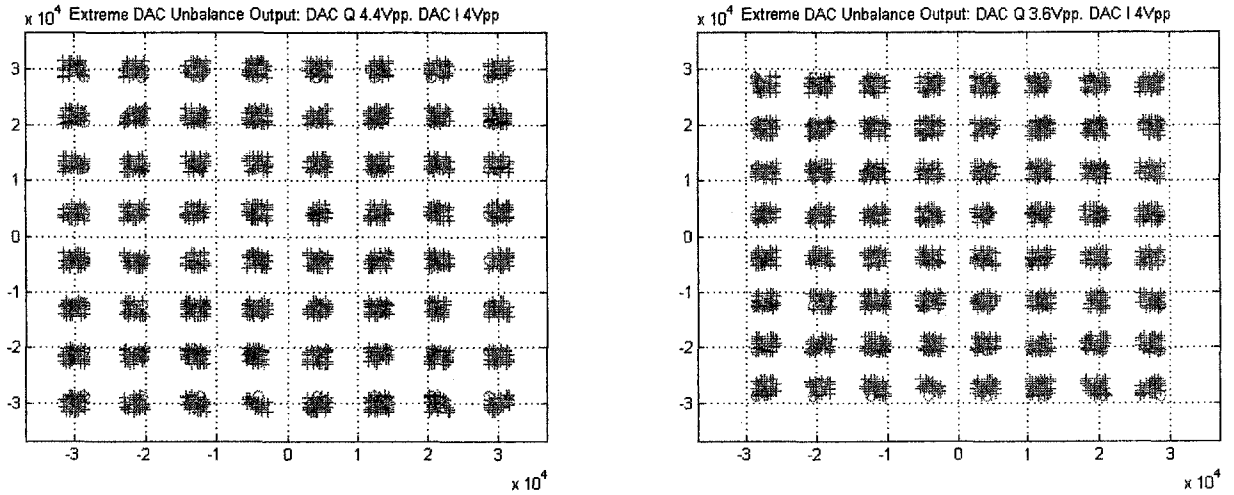
Possible sources include unbalanced LO drive levels feeding the I/Q mixers during down conversion, mixer mismatches, I/Q branch filtering attenuation mismatches (performed before ADC sampling), uneven I/Q ADC gain error, and unmatched DAC output swings in the transmitter.

In the simulation model, gain error sources are caused by intentional discrepancies between the Q DAC output levels and the receiver's demodulation constant used to undo the DAC scaling. To simulate I/Q amplitude mismatch, a gain error is introduced only to the Q channel. Although the I DAC is at its ideal level in this simulation, it may not be in an actual system. Table 4-12 summarizes key the configuration details specific to this investigation case.

Table 4-12. Simulation Settings for I/Q Amplitude Imbalance

Item	Value or Detail(s)
IFFT/FFT size	64
Figure of Merits	▪ MSE_{cst} and SIR_{bit} computation do not include first FFT output bin, since they are sensitive to DC offset errors and are not of interest assuming the first FFT output bin does not carry data ▪ SIR without normalization to a bit is also computed for cross verification with [10], which also excludes the first FFT output bin.
DAC resolution	12-bits
DAC output swing	Q DAC varied around 4Vpp. I DAC fixed at 4Vpp.
Receiver demodulation expects I/Q DAC output swing	4Vpp
DC offset	0 V
ADC Resolution	Infinite. Therefore no information loss in the 12-bit DAC words received.
Receiver Sample Decimation	At time instant $\frac{T_{OFDM_{sym}}}{2}$
LPF Filtering	None used

The following figures 4-18a and 4-18b are constellation plots for *+/-10%* unbalanced DAC voltage output error. They demonstrate how QAM data points/symbols spread around its ideal reference locations on the constellation map. Depending on positive or negative Q channel DAC error with respect to 4Vpp, there is evidence of constellation scaling with respect to the expected reference points, and data QAM symbols spread around this scaled constellation. Positive error in the Q channel DAC shifts the spreading clusters outwards, while negative gain shifts them inwards. The spreading phenomenon is caused by inter-carrier interference (ICI) due to the amplitude imbalance after the FFT transform in the OFDM demodulator. Thus each data QAM point is displaced with interfering components from other QAM points mounted on orthogonal carriers. This interference noise on the constellation does not have a Guassian distribution nor does it conform to any other know distribution forms, as shown in figure 4-19.



(a) *(b)*

Figure 4-18. Illustration of Extreme I/Q Imbalance:
(a) Positive Q DAC Error. (b) Negative Q DAC Error.

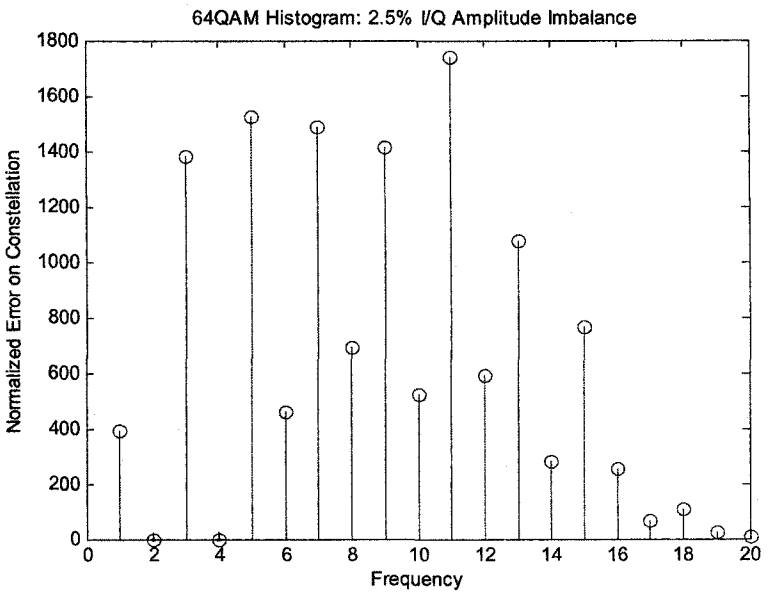


Figure 4-19 Histogram of Normalized Errors for 64QAM: 2.5% I/Q Amplitude Imbalance

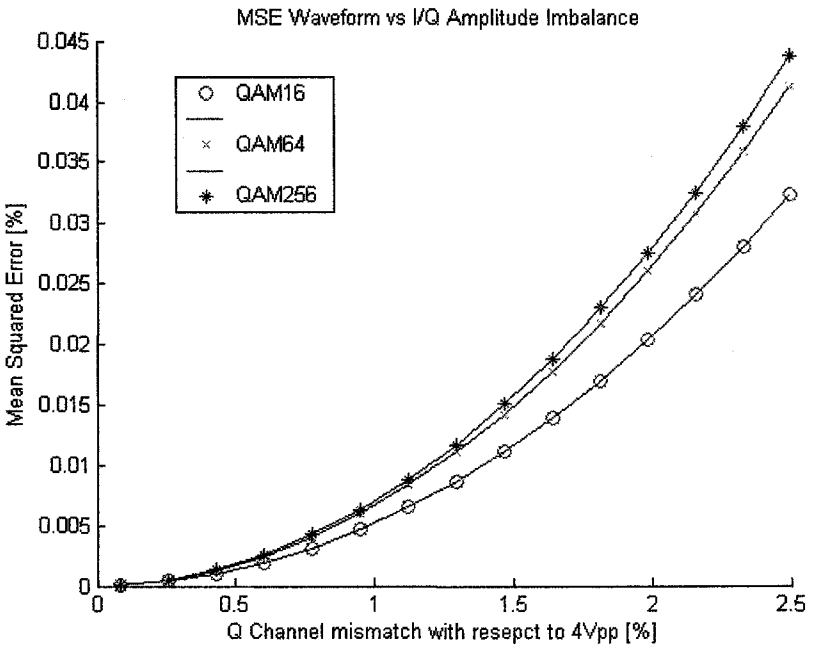


Figure 4-20. MSE_{wave} versus I/Q Amplitude Mismatch

Examining the MSE_{wave}, as the Q channel DAC output is tuned away from the ideal 4Vpp towards +/- 0.1V (2.5% I/Q error), MSE_{wave} increases monotonically, as shown in figures 4-20. For the post FFT transform performed on the received samples, the MSE_{cst} also increases monotonically, while the SIR_{bit} decays, as show in figure 4-21 and 4-22.

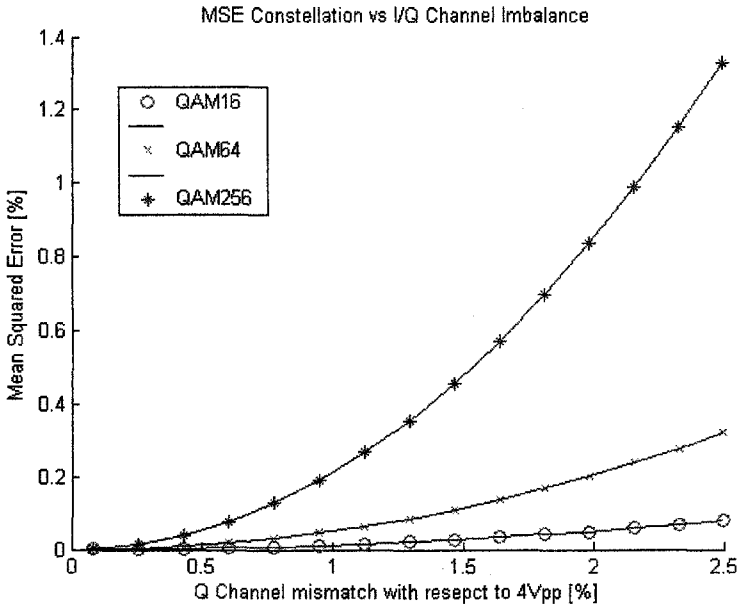


Figure 4-21. MSE_{cst} versus I/Q Amplitude Imbalances

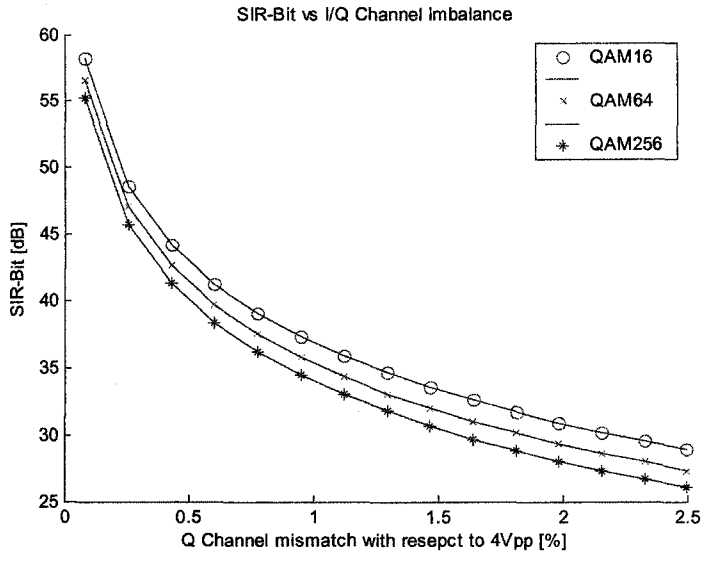


Figure 4-22. SIR_{bit} versus I/Q Amplitude Imbalances

In comparison to the gain error results in section 4.2.1 'Effects of Gain Error', the pattern for MSE_{cst} and SIR_{bit} performance graphs are similar, but the magnitude of MSE_{cst} is

approximately double that of the unbalanced case, as compared between figure 4-14 and figure 4-21. Similarly, SIR_{bit} curves in each constellation scheme figure 4-22 is approximately 3dB better than the previous 'gain error' case figure 4-15. As found in gain error case, the SIR_{bit} performance of the system is quite sensitive to I/Q amplitude imbalances, with both cases monotonically decaying with increasing error magnitude.

The degradation curves in figure 4-23, demonstrates the greater degradation in the higher S/N_0 scenarios, because the interference-to-thermal noise ratio is higher.

Reference [10] also investigates the effects of I/Q amplitude imbalances on QPSK-OFDM-QAM systems. This reference performs the mathematical analysis of the I/Q amplitude imbalance and relates it to the signal to inter-channel interference (SCSI) ratio, which is another notation equivalent to SIR used in this thesis. The analysis carried out in [10] expresses I/Q balance as β in dB and defines α in the following manner:

$$\alpha = \frac{10^{\beta/20} - 1}{10^{\beta/20} + 1} \tag{4-1}$$

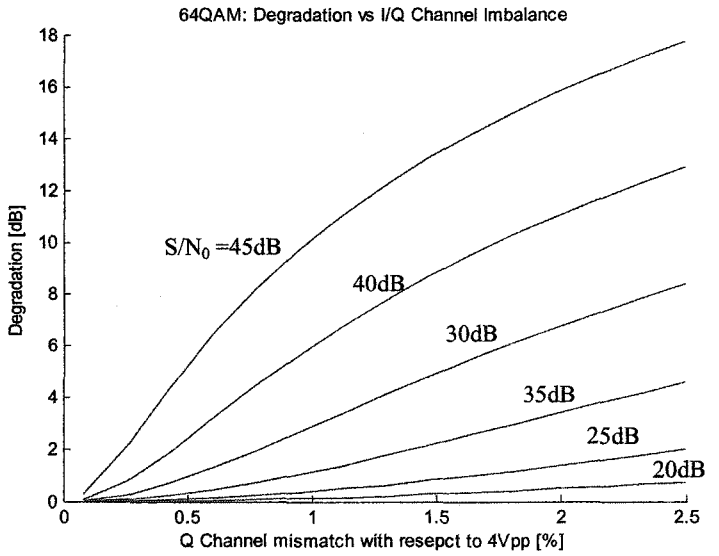


Figure 4-23. Degradation versus I/Q Amplitude Imbalances

With α, the SICI can be expressed as:

$$SICI = 20LOG_{10}(\frac{1}{|\alpha|}) \qquad (4\text{-}2)$$

$$SICI \equiv SIR$$

Where:

- SIR is the signal-to-interference ratio used in this section to express the inter-channel interference in a different notation than found in [10], where the equation is extracted.
- SIR is defined in equation 3-9, back in section 3.7.1.
- It also assumes the I/Q imbalance error has equal magnitude of the opposite sign in both I/Q channels.

Using equation 4-1 to plot the theoretical results and comparing it to simulation results as presented altogether in figure 4-24. Both curves for all simulations and theoretical overlap, leading to the conclusion that the simulation results are in excellent agreement with theory [10].

4.2.3 DAC/ADC Resolution

Reduced resolution in the receivers may apply to situations where lower resolution ADCs could reduce hardware costs. Selection of the ADC resolution is usually based on a tradeoff between cost and performance. To investigate the effects of relative resolution in the system, the ADC is now limited. The DAC is fixed at 12-bits over 4V and the ADC is varied between 6-11 bits over a 8V input range, while other parameters are set to ideal values. ADC input saturation is not a factor in this simulation case, and a 4Vpp input range could have been used, but it does not change the results collected.

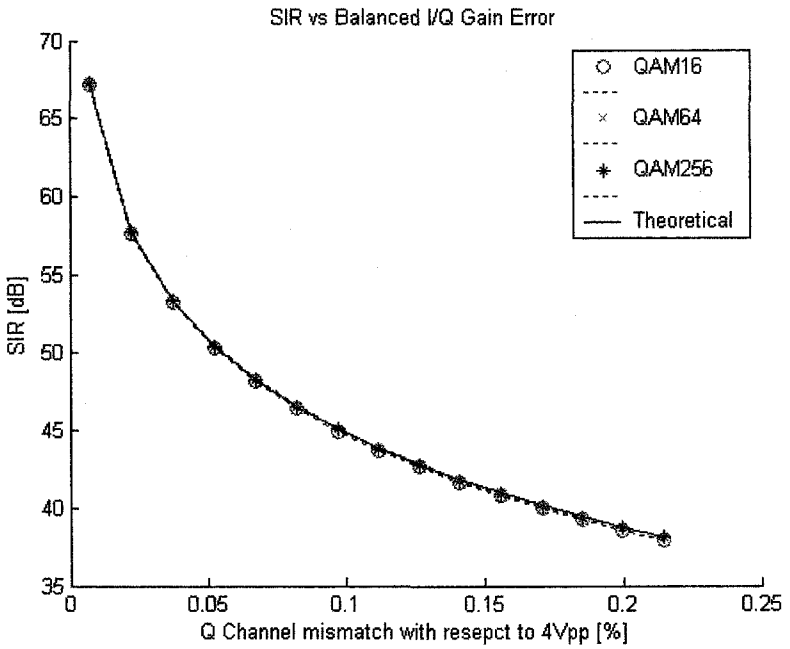


Figure 4-24. SIR Theoretical versus Simulation

Q channel DAC is tuned away from 4Vpp in one direction

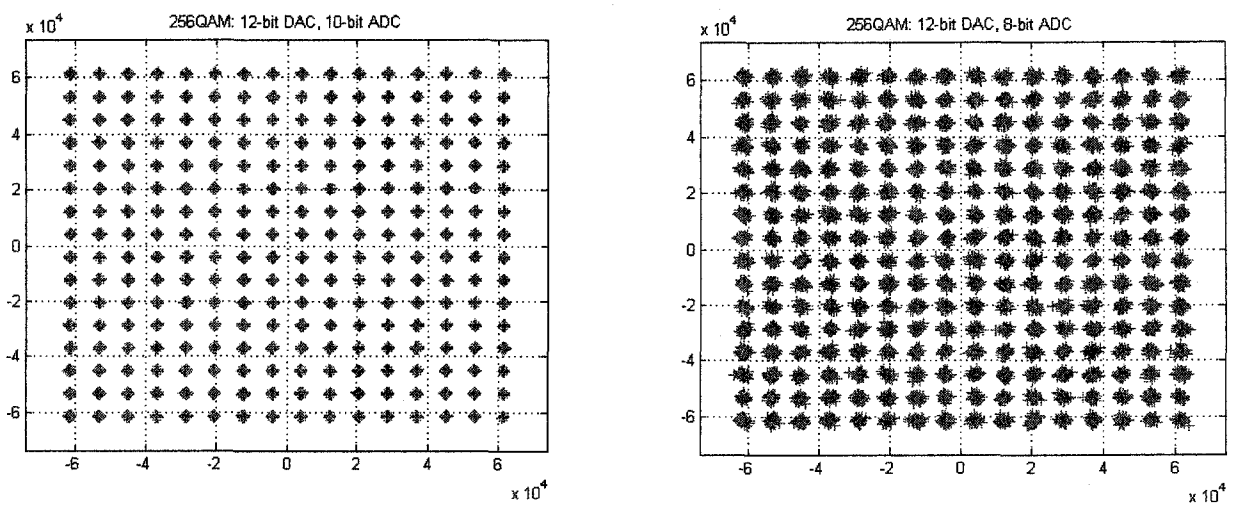
I channel DAC is tuned away from 4Vpp in equal magnitude in the opposite direction relative to Q.

The resolution can be adjusted by reducing a single resolution bit in this section for comparability with respect to the DAC resolution. Please refer to table 4-13 for a summary of the key configuration details of the software model, for this investigation.

In the simulation, reduction of resolution in the ADC of the receiver causes data QAM symbols on the constellation to spread around the ideal reference points due to loss in precision at the receiver, resulting in quantization noise. This data QAM symbol spreading effect and degradation with precision loss in the number of bits is demonstrated in figures 4-25a and 4-25b, for 10-bits and 8-bits over 8V in the receiver's ADC.

Table 4-13. Setup Details for Effective Resolution

Item	Value or Detail(s)
IFFT/FFT size	64
Figure of Merits	MSE_{cst} and SIR_{bit} computation do not include first FFT output bin, since they are sensitive to DC offset errors and are not of interest assuming the first FFT output bin does not carry data.
DAC resolution	12-bits
DAC output swing	I/Q set at 4Vpp
Receiver demodulation expects I/Q DAC output swing	4Vpp
DC offset	0 V
ADC Resolution	Varied between 6-11 bits over an 8Volts. For comparable resolution to DAC, add one bit to compensate for the 8Volt range in comparison to 4V range of DAC. Example: 12-bit DAC over 4 V is equivalent to 13-bit ADC over 8V input range.
Receiver Sample Decimation	At time instant $\frac{T_{OFDM_{sym}}}{2}$
LPF Filtering	None used



(a) (b)

Figure 4-25. Illustration of ADC Resolution Degradation at: (a) 10-Bits (b) 8-bits

The probability density function of the error signal due to quantinzation appears to be nearly Guassian; this is demonstrated as a visual in figure 4-26, which shows a histogram plot of the normalized errors on the 64QAM constellations given 12-bit DAC and 10-bit ADC resolution.

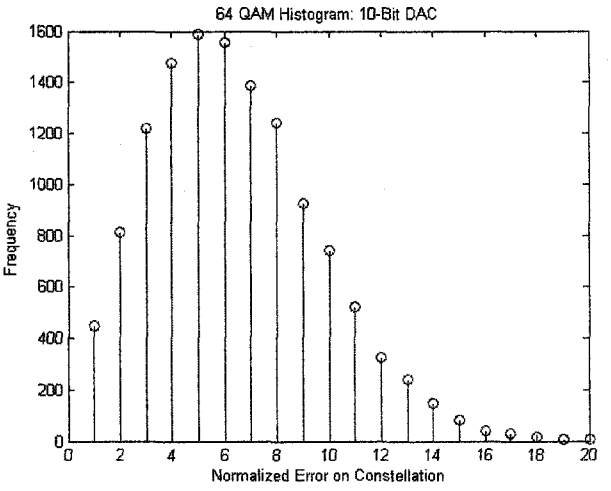


Figure 4-26. Histogram of Normalized Errors for 64 QAM: 12-bit DAC and 10bit-ADC.

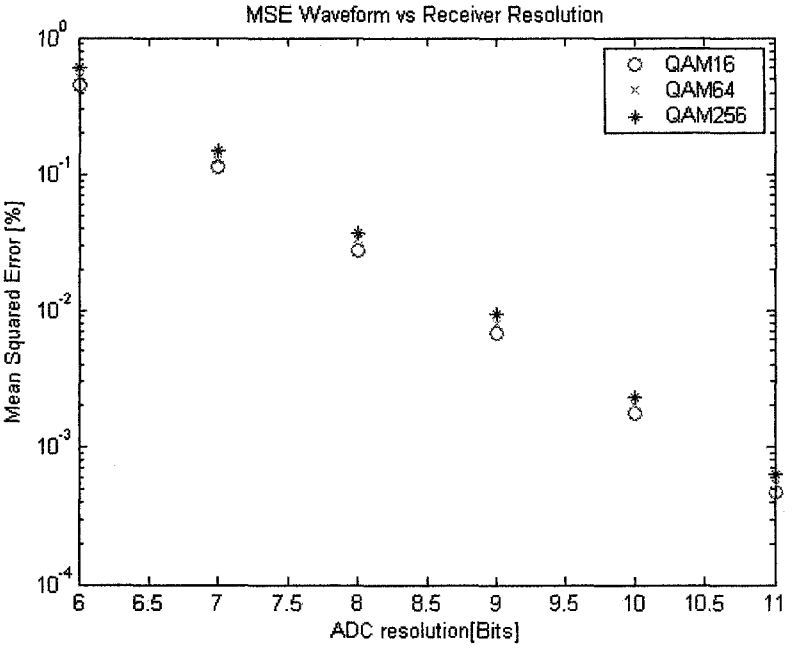


Figure 4-27. MSEwave versus Receiver Resolution

Figure 4-27 shows the MSE_{wave} versus ADC resolution. As expected, the mean square error decreases monotonically with the number of ADC bits; close inspection of this figure indicates that the mean squared error decreases by a factor of 4 for each added bit of resolution.

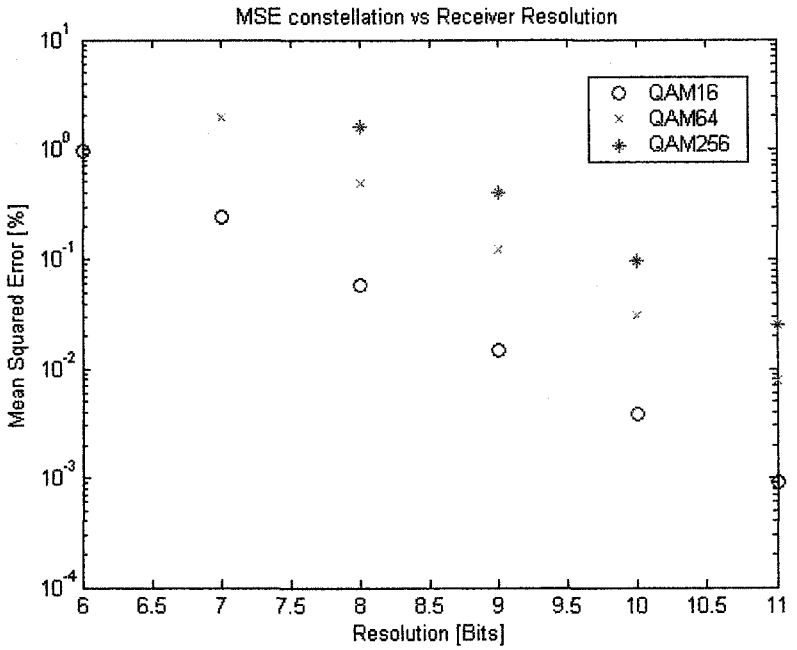


Figure 4-28. MSE_{cst} versus ADC Resolution
No data points are plotted if QAM symbol errors occur in simulation

Figure 4-28 plots the monotonic increase of MSE_{cst} as ADC resolution decrements. For every bit of resolution lost, the MSE_{cst} increases by a factor of four times its previous value. SIR_{bit} decreases as the receiver resolution is reduced as shown in figure 4-29. In the figure, a 6dB SIR_{bit} difference is observed between each 1-bit change in receiver resolution, regardless of constellation size. This 6dB loss phenomenon per resolution bit is equivalent to the fundamental principles of a single converter, which is mathematically explained section 2.5.5 'Quantization Noise'. However, different to the analysis on the data converter alone to predict the SNR given the resolution, this simulation based on the OFDM model includes the FFT transform function before determining the SIR_{bit} and/or SNR. Thus, with the results in this investigation, it appears both the data converter alone and the this OFDM system that includes both the FFT transform (a linear function) and a

data converter share similar properties and responses to resolution changes, assuming everything else is ideal as described in this simulation setup.

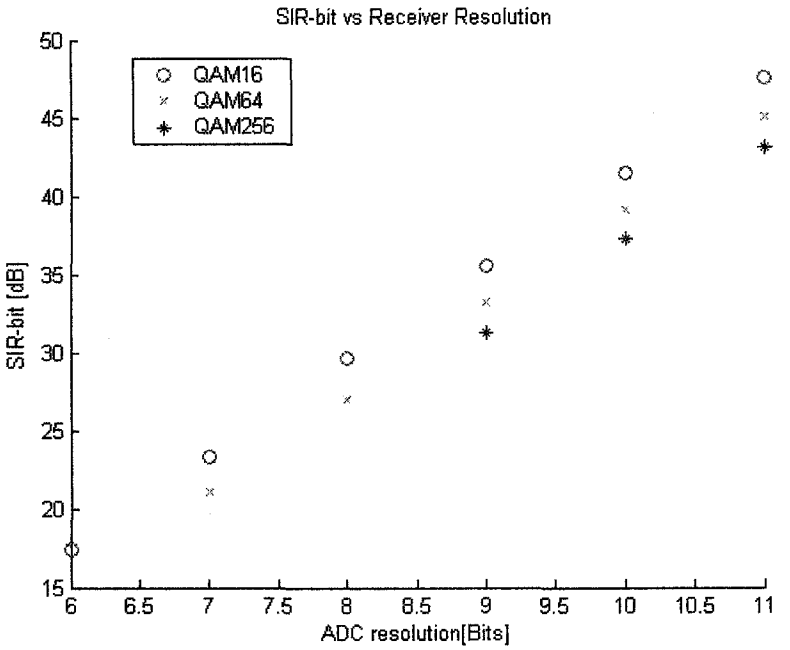


Figure 4-29. SIR_{bit} versus ADC Receiver Resolution

The degradation curves shown in figure 4-30, expresses the degradation sensitivity to S/N_o with less resolution in the receiver.

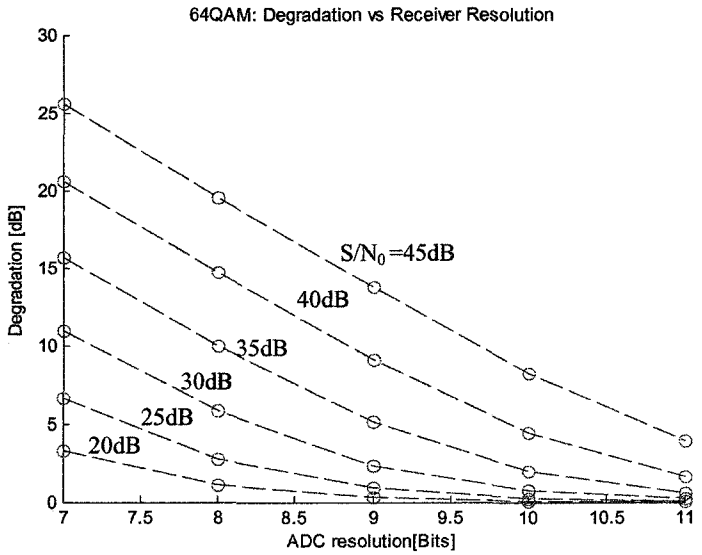


Figure 4-30. Degradation versus Resolution

4.2.4 Phase Delay from Filtering

Phase mismatch between the inphase and quadrature channels are caused by two main sources:

1. Phase mismatch caused by the inability of the two individual LOs[26] to differ exactly by an orthogonal 90 degrees when driving the down conversion mixers for the I and Q branches causes I/Q components leaking into each other after down conversion.
2. Parallel I/Q branch filtering (performed after down conversion) phase delay mismatches still preserves I/Q isolation of each component in each I/Q branch. However, the phase delay caused by filtering displaces sampling time instances from its ideal sample location causing amplitude distortion in the sampled values. This results in amplitude distortion that is dependent on filter characteristics and filter delay mismatch magnitudes.

In this thesis, only on relative delay caused by filtering tolerances is considered.

Even if the I/Q filters are identical, waveform distortion due to the bandwidth limitations imposed by the filters is another non-ideal factor. Low pass filtering without some form of amplitude compensation degrades the MSE_{wave}, MSE_{cst}, and SIR_{bit} because of the amplitude distortion caused by the filter, as discussed in section 3.6.4 'Low Pass Filtering'. Depending on characteristics of the filter, it is possible to at least partially compensate for the effects of the filter if negative or positive gain were applied to the analog signal. This compensation effect is observed in the next section, where all analog factors are combined into the same simulation case. In table 4-14, the key software model settings listed are used to observe the impact of I/Q phase imbalance caused by filtering.

[26] LOs are parts of an I/Q demodulator, which is intended to extract the complex envelope of the received signal.

Table 4-14. Setup Details for Relative I/Q Filtering Delay

Item	Value or Detail(s)
IFFT/FFT size	64
Figure of Merits	MSE_{cst} and SIR_{bit} computation do not include first FFT output bin, since they are sensitive to DC offset errors and are not of interest assuming the first FFT output bin does not carry data.
DAC resolution	12-bits
DAC output swing	I/Q 4Vpp
Receiver demodulation expects I/Q DAC output swing	4Vpp
DC offset	0 V
ADC Resolution	Infinite. Therefore no information loss in the 12-bit DAC words received.
Receiver Sample Decimation	I channel: At time instant $\frac{T_{OFDM_{sym}}}{2}$ Q channel: At time instant $\frac{T_{OFDM_{sym}}}{2}$ + *(variable delay)*
LPF Filtering	I/Q channel passes through a matching pair of 7^{th} Order FIR filters with equiripple response and 3dB bandwidth at $f_{OFDM_{sample}}$

With the low pass filtering implemented in the system and sampling the Q channel at a delayed time instance of: $(\frac{T_{OFDM_{sym}}}{2})$+ *(variable delay)*, the performance begins to deteriorate because the delayed samples of the signal move further away from the ideal levels/values. An illustration of the delay concept is shown in figure 4-31. The 'ideal sample' location represent $(\frac{T_{OFDM_{sym}}}{2})$, while the 'delayed sample' markers represent $(\frac{T_{OFDM_{sym}}}{2})$+ *(variable delay)*.

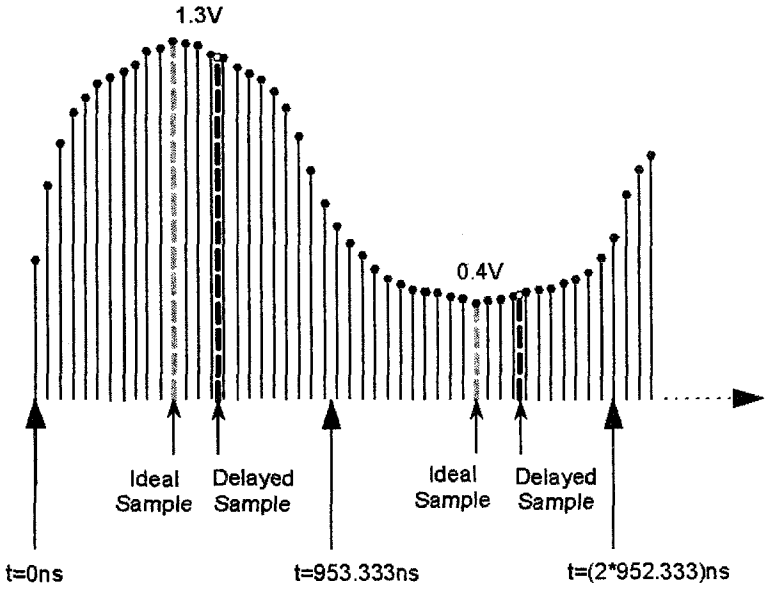


Figure 4-31. Illustration of Filtering Causing Delayed Sampling Time Instances In simulation samples are discrete, 3rd order polynomial line interpolation used to overcome time resolution

Combining the gain error and I/Q imbalance distribution characteristics, the error distribution on the constellation is not of a Guassian form either, as show in figure 4-32.

As first demonstrated in, figure 4-33, the waveform distortion gauged by MSE_{wave} exponentially spurts up near 29% I/Q phase delay[27]. Before reaching this delay value, the MSE_{wave} remains fairly constant, due to the quasi-stable level in this region of the analog wave with respect to the ideal transmitted levels.

[27] I channels sampled at $\frac{T_{OFDM_{sym}}}{2}$, while Q channels sample at a delayed percentage from $\frac{T_{OFDM_{sym}}}{2}$

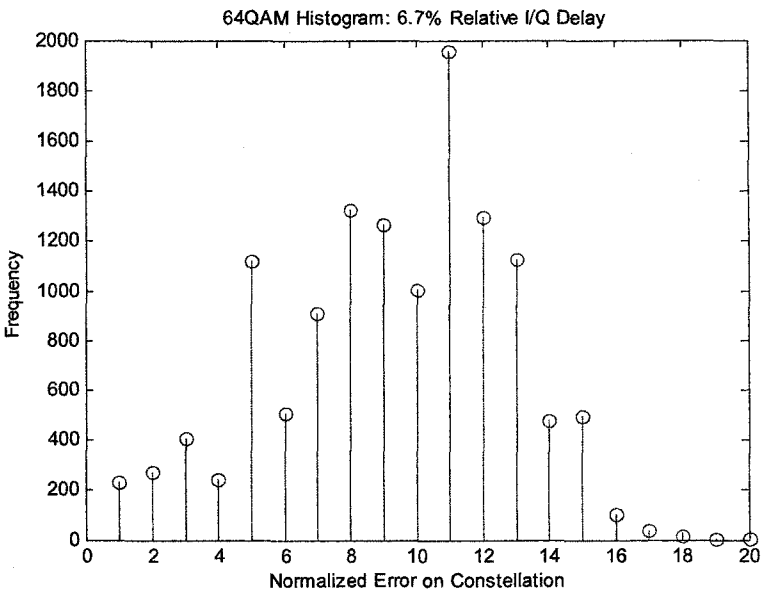


Figure 4-32. Histogram of Normalized Errors for 64QAM: Relative I/Q Delay

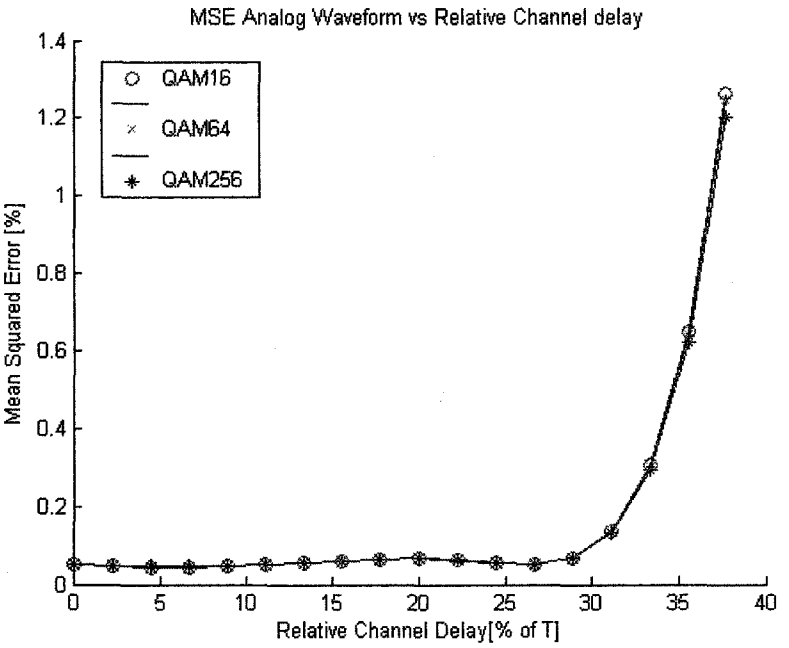


Figure 4-33. MSE_{wave} versus Sampling Q Channel at Relative Delay Time Instance

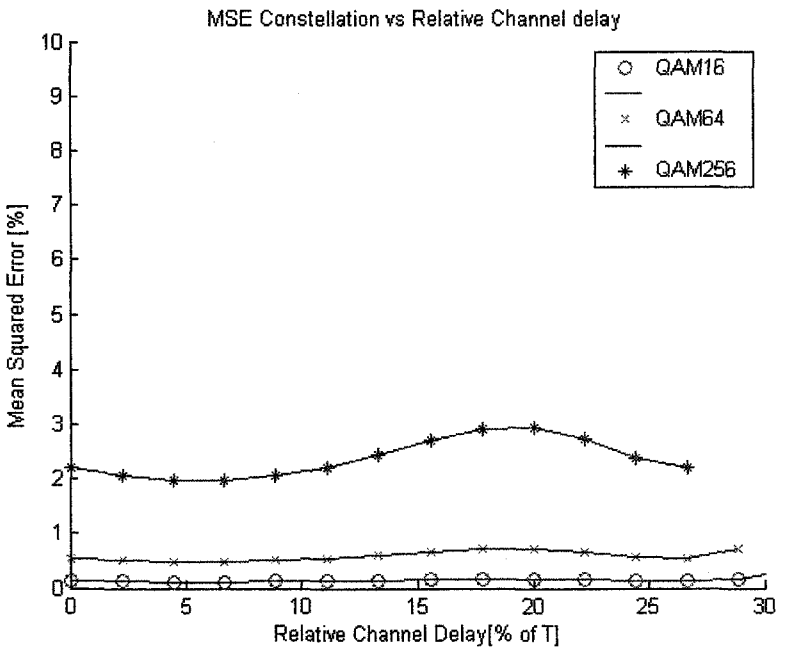


Figure 4-34. MSE_{cst} versus Sampling Q Channel at Relative Delay Time Instance
No data points are plotted if QAM symbol errors occur in simulation

The MSE_{cst} curve in figure 4-34 remains fairly constant before 29%. Approaching 29%, data QAM symbol errors begin to occur in the larger constellations first.

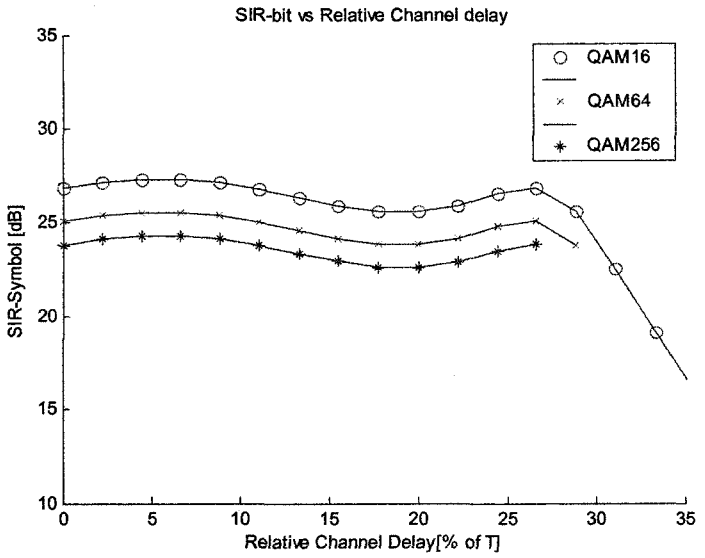


Figure 4-35. SIR_{bit} versus Sampling Q Channel at Relative Delay Time Instance
No data points are plotted if QAM symbol errors occur in simulation

Conforming to the MSE_{cst} results, the SIR_{bit} plots in figure 4-35 shows deterioration beginning before 29% relative delay.

Overall, the results for I/Q relative delay given the filter used show signal quality and demodulation is tolerable against a substantial amount of phase delay before significant signal distortion breaking down. Quantitatively, tolerances of up to 10% delay results in approximately 0.5dB of variance in SIR_{bit} for all constellation sizes observed. 10% percent tolerances typically exceed all manufacturing tolerances, and its unlikely for relative delay in parallel I/Q branch filtering to exceed this level. If it should, then the performance penalty in term of SIR_{bit} can be extracted from the corresponding figure for design estimations.

4.2.5 Combination Of Factors

This section discusses the results for a combination case of analog factors that includes: gain error, I/Q amplitude imbalance, transmitter/receiver resolution, and relative channel delay caused by low pass filtering. This combination case highlights the fact that each source of error does not necessarily contribute to performance degradation, as interference on the constellation, by superposition when other factors are present.

With several non-ideal factors considered, there are several variables in the system model. Since the objective of the case is to prove the combine net effect and not provide results for design guidelines, several variables are fixed. To begin describing the details of the simulation, the system receiver ADC is set to 10-bits over $8V^{28}$, while the transmit DAC has 12-bits over its output voltage. To factor in gain error and I/Q amplitude imbalance, the Q-channel DAC was set to 3.92Vpp, while the I-channel DAC output was 3.94Vpp. This equates to -2% gain error, and 0.5% (0.02V) I/Q amplitude imbalance

[28] 10 bits over 8V (Equivalent to 9 bits of resolution over 4Vpp for comparison to DAC expected 4Vpp) creates a ADC input range wide enough to avoid any input saturation due to the non-ideal analog factors introduced, and it is assumed that any OFDM would budget margin in a system to avoid saturation. With this assumption, this ADC saturation factor is not of interest and is excluded in this combinational case. The resolution in ADC has been selected for 10 bits because this would allow for a significant amount of relative I/Q delay in the simulations before QAM symbol errors.

with respect to 3.92Vpp. The values were selected within close proximities to the possible hardware simulation results. The negative DAC error is arbitrarily selected over the positive error. From the DC offset simulation results, in the most sensitive case, the max tolerable DC offset permitted in 256QAM is determined to be 0.15%, assuming all FFT output bins are considered. In this combination case 0.050% is introduced, budgeting margin for other factors before possible QAM symbol errors, excluding the first FFT output bin in analysis. A summary of all the model configuration details for this combinational case is presented in table 4-15.

Table 4-15. Setup Details for Combination Case

Item	Value or Detail(s)
IFFT/FFT size	64
Figure of Merits	MSE_{cst} and SIR_{bit} computation do not include first FFT output bin, since they are sensitive to DC offset errors and are not of interest assuming the first FFT output bin does not carry data.
DAC resolution	12-bits
DAC output swing	I: 3.94Vpp Q: 3.92Vpp
I/Q Amplitude Imbalance Error	0.5% with respect to Q DAC output
Gain Error	2% with respect to 4Vpp and Q DAC output
Receiver demodulation expects I/Q DAC output swing	4Vpp
DC offset	0.002 V (0.050% with respect to 4Vpp)
ADC Resolution	10-bits over 8 volts to avoid any ADC input saturation.
Receiver Sample Decimation	I channel: At time instant $\frac{T_{OFDM_{sym}}}{2}$ Q channel: At time instant $\frac{T_{OFDM_{sym}}}{2}$ + *(variable delay)*
LPF Filtering	I/Q channel passes through matching 7th Order FIR filters with equiripple response and 3dB bandwidth at $f_{OFDM_{sample}}$. (Same filter used in 'Filtering Delay' simulations)

First of all, observing the MSE_{wave} plot in figure 4-36, the curve characteristics are noticeably similar to the results in the isolation of 'delay' factor performed in the previous section. However, the curve begins to rapidly rise earlier than before for all QAM constellation sizes. Instead of spurting up near 29%, it begins near 25%, due to greater impact of the combined analog factors. The exponential increase in distortion quickly leads to QAM symbol errors in the simulation at approximately 27% relative delay for 256 QAM as shown in following figures. In the absolute sense accounting all combination factors, MSE_{wave} is less than 0.01% for relative delay up to 10%, and fairly constant before 25% delay, in all constellation schemes.

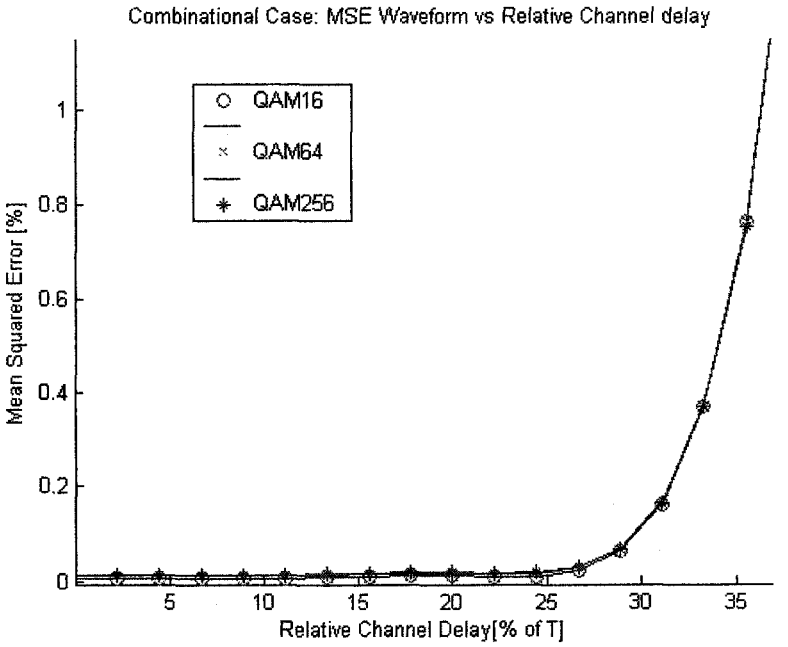


Figure 4-36. Combination Case: MSE_{wave} versus Relative I/Q Channel Delay

In figure 4-37, the MSE_{cst} computed and plotted to express the greater sensitivity to relative I/Q channel delay for this combination case versus I/Q phase delay alone. Similar to the phase delay simulations, the combinational factors demonstrate greater MSE_{cst} magnitude for the larger constellation schemes.

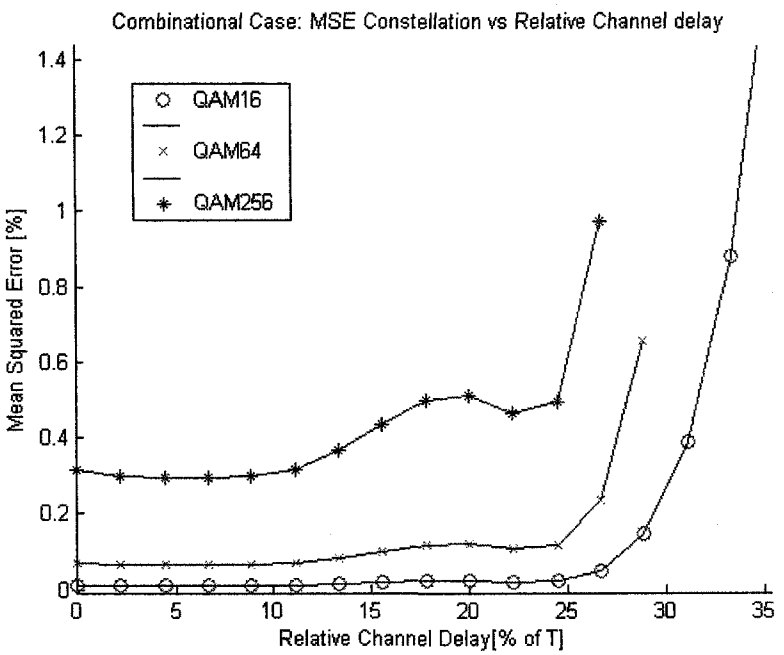


Figure 4-37. Combination Case: MSE_{cst} versus Relative I/Q Channel Delay.

Next, the plot for SIR_{bit} are shown in below figure 4-38 and demonstrate less than 1dB change for less than 10% relative I/Q delay in this combinational case.

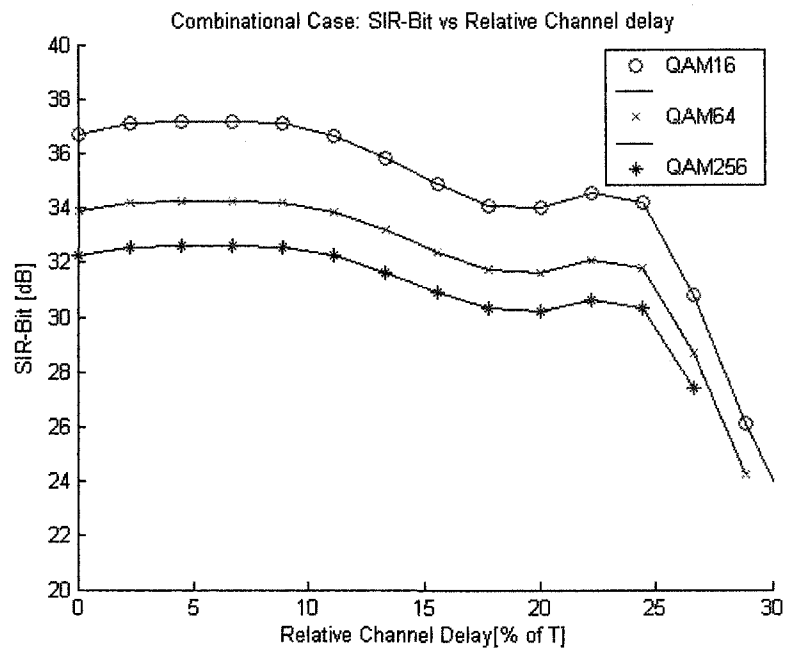


Figure 4-38. Combination Case: SIR_{bit} versus Relative Channel Delay.

In the above plots for the specific simulation case, the gain error is –2%. This negative gain improves SIR_{bit} performance in this specific case as proven in figure 4-39. In this figure, the SIR_{bit} curves for ideal 4Vpp I/Q DAC outputs a plotted for comparison to the non-ideal DAC output curves, which demonstrates a tremendous 9 to 10dB performance difference in the various constellation schemes. Thus in such scenario, negative gain error improves performance to compensate for amplitude distortion caused by filtering. However, too much negative gain error eventually degrades performance, thus limitations apply. To elaborate, the attenuation of the higher frequencies from filtering cause overshoot and undershoot relative to the ideal waveform levels because of the lower sinusoidal components dominating. Thus negative gain error in receiver reverses this effect, and improves SIR_{bit} and MSE_{cst} performances to a certain degree.

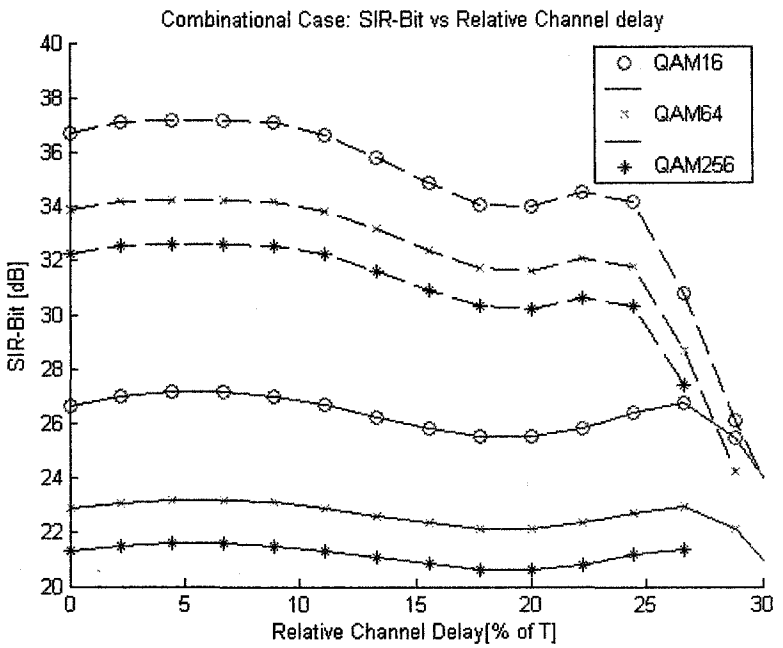


Figure 4-39. First Combination Case Modified: SIR_{bit} versus Relative Channel Delay.

Solid Curves - I/Q DAC outputs at 4Vpp. All other variables remain the same as previous combinational simulation.

Dashed Curves - Original combination case; they are the same curves as previous figure 4-38.

Going back to the original combination configuration, then reducing the ADC resolution from 10-bits to 9-bits, the results in figure 4-40 show SIR_{bit} degradation of less than the 6 dB, as observed in the 'resolution' isolation case. Thus, the ADC quantization noise imposed upon the distorted waveform does not have the same magnitude of impact as it does with a perfect waveform.

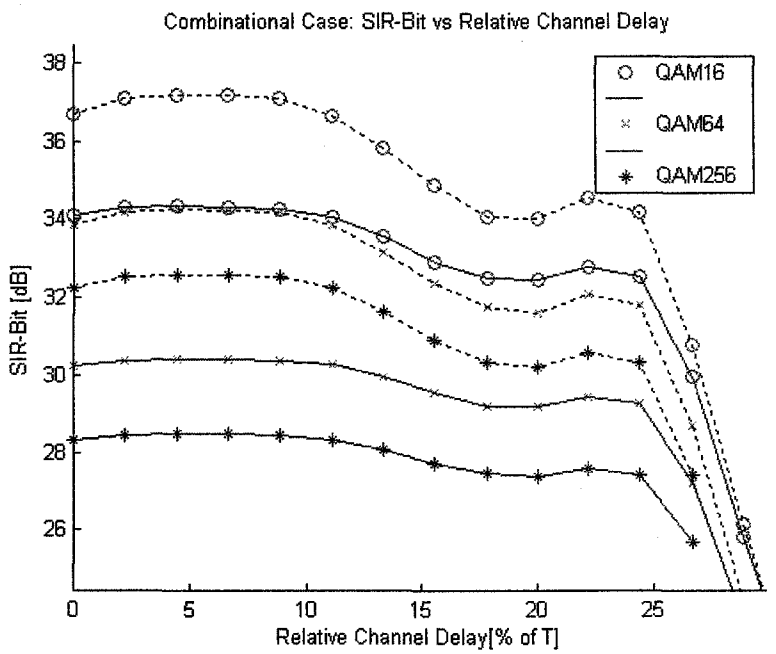


Figure 4-40. Second Combination Case Modified: SIR_{bit} versus Relative Channel Delay.

Solid Curves - ADC Resolution reduced to 9-bits from 10-bits. All other variables remain the same as original combinational simulation.

Dashed Curves - Original combination values; they are the same curves as figure 4-38.

Chapter 5

Conclusion and Future Works

5.0 Overview

This section concludes the important finding in the thesis work for both the hardware experiments with corresponding simulations, and the various software simulation cases that are used to investigate non-ideal analog factors on OFDM signaling. After the conclusions and before ending this chapter, a suggestion is discussed on possible future studies to build upon the contributions made in this thesis.

5.1 Hardware Experiments and Equivalent Simulation Conclusions

An elaborate OFDM hardware system is used to observe the impact of interference/noise due to gain error, I/Q amplitude mismatch, fixed DAC/ADC resolution, and DC offset. From all the experimental trials, the average of each figure of merit is computed and were matched in simulations. By doing so, the hardware system is successfully emulated in simulations just by adjusting the error sources mentioned above, which proves the expected analog factors mentioned to exist and to have an impact on

receiver/demodulation performance in terms of mean squared error waveform (MSE_{wave}), mean squared error constellation (MSE_{cst}), and signal-to-noise ratio normalized to a bit (SIR_{bit}).

5.2 Pure Software Simulation Conclusions

5.2.0 DC Offset

DC offset error imposed onto the sampled data, in the I/Q channel in the receiver, shifts the first FFT output bin in each OFDM symbol on the constellation. Too much DC offset eventually displaces these QAM points to be demodulated incorrectly, but there is no obvious impact on the rest of the FFT output bins from 2 to 64.

5.2.1 Gain Error

Uncompensated balanced gain error in the I/Q channels causes constellation scaling, which translates to interference onto the data symbols in demodulation process, where 'uncompensated' refers to demodulation using another scaled constellation for reference. There are significant gain/improvements to MSE_{cst}/SIR_{bit}, for minimizing uncompensated gain error as much as possible.

5.2.2 DC offset or Gain Error causing ADC Input Saturation

DC offset and gain error magnitudes, in separate investigation, for up to the values explored do not have a significant impact by saturating the ADC input due to the nature of the OFDM time waveform, which only occasionally occupies the peak ranges of the ADC. This assumes the signal consumes the full input range of the DAC/ADC converters without saturation/clipping before introducing DC offset, and the first FFT output bin is not used to carry data.

5.2.3 I/Q Amplitude Imbalance

MSE_{cst}, and SIR_{bit} performance results share the same pattern as 'gain error' simulations except MSE_{cst} is only half the magnitude and SIR_{bit} is 3dB better, due to gain error occurring only in the Q channel. Also, the I/Q amplitude imbalance results agree with the theoretical curves presented in [10] and [40].

On the constellation, I/Q amplitude balances causes QAM data symbol spreading around its ideal demodulation location point because of inter-carrier interference crossing into each carrier at the FFT transform stage. There are significant improvements to MSE_{cst}/SIR_{bit}, if I/Q amplitude imbalance is minimized as much as possible.

5.2.4 ADC Resolution Reduction

When the ADC resolution in the receiver is less or equal to the DAC in the transmitter, system SIR_{bit} decreases by 6dB for every bit of precision lost in the ADC. This is similar to SNR degradation characteristic for a single data converter such as an ADC. However, the input signal of the ADC does not saturate, the results are likely to differ if the input were a Guassian distribution, where intentional clipping is introduced in the transceiver to optimize available dynamic rage of the data converters.

5.2.5 Relative I/Q Delay

In delay simulations, where a 7^{th} order LPF is included in both I/Q channels, and relative delay in sampling time instance is varied, the delay effect disturbs demodulation performance because of amplitude distortion is caused in the delay sampling time instance. In the simulations results, tolerances of up to 10% delay results in approximately 0.5dB of variance in SIR_{bit} for all constellation sizes investigated.

5.2.6 Combination of Analog Factors

In summary from the results in the combination of factors case, non-ideal analog factors combined in typical OFDM systems do not necessarily combine in a linear manner or by superposition using simulations results from simulations that isolate these factors.

5.2 Possible Future Study

To complement the accomplishments made in this thesis and in this area, there are opportunities in the future to investigate other OFDM analog interfacing factors that include:

1. The internal nonlinear effects ADC/DAC converter gain error
2. Introduce optimization of the DAC/ADC dynamic range by applying intentional signal clipping to present model, the re-investigate results.
3. Phase noise imposed onto I/Q signals at down conversion from the oscillator
4. Various analog filter designs and the effects of relative I/Q delay mismatches caused by them.

The investigations would involve simulations and/or hardware experimental experiments to characterize and quantify the impact of each factor listed.

Appendix A

A1: DSP Program Source Code

In this section, the DSP C program is provided. Simplifications to variable declaration and initialization have been made and noted to reduce space. The objective in presenting the code is to focus on the concepts in serial port and interrupt initialization, data processing and event handling in the hardware routine section

```
/*************************************************************************
 *Written by Chiko Lee 2003
 *-McBSP1 will receive PN sequence
 *-The data will be processed and 10 OFDM sym will be formed
 *-McBSP1 R component
 *NOTE: Do not use Restart form Debug menu
 *      Instead always reload program to reset
\*************************************************************************
 Begin with header files
*******/
#include <std.h>
#include <swi.h>
#include <log.h>
#include <stdio.h>

#include "edmacfg.h"
#include "main.h"
#include <csl.h>
#include <csl_cache.h>
#include <csl_edma.h>
#include <csl_irq.h>
/*-------------------------------------------------------------------------*/

/* declare DSP/BIOS objects created with the configuration tool */
extern far SWI_Obj SwiMain;
```

```
extern far LOG_Obj LogMain;
extern far SWI_Obj swiProcess;
extern far LOG_Obj trace;

/*******************************************************/
/* These are the interrupt codes for the EMDA channesl that
 * service McBSP1
 */
#define TCCINTNUM_TX2  14
#define TCCINTNUM_RX2  15

/* define some constants */
#define BUFF_SZ    128          /*For IFFT input buffer */
#define BUFF_SZ_12  12
#define FCPU       150000000      /* CPU clock frequency              */

 /*Number of Constellation pts for QAM64*/
#define NUM_CONSTELATION_POINTS  64
#define CHANNELS              64   /*Number of IFFT points and Sub-Carriers*/
#define ARRAY_SIZE            64   /*Number of IFFT points and Sub-Carriers*/
#define ARRAY_SIZE_2          128  /*2x ->Number of IFFT points and Sub-Carriers*/
48   /*Size of out array to hold single OFDM sybmol*/
#define OUTPUT_ARRAY
 #define PTS                  64   /*Number of IFFT points and Sub-Carriers*/
#define SQRT_PTS              8    /*Sqrt of Number of IFFT points*/
#define PI                 3.14159265358979
#define PCH_SZ                4
#define CH_SZ                 16
/*********************************************************************/
```

Not all variable declarations will be show.

```
/* Create the buffers. We want to align the buffers to be cache friendly */
/* by aligning them on an L2 cache line boundary.                */


        #pragma DATA_ALIGN(outbuff,128);
        int outbuff[BUFF_SZ];

        /*Buffer for incoming data*/
        #pragma DATA_ALIGN( edmaInbuff,128);
        int edmaInbuff[BUFF_SZ]; /* buffer for EDMA supporting devices */


        /*Buffers holding output OFDM symbols*/
        #pragma DATA_ALIGN(outBuff1r,128);
```

```
#pragma DATA_ALIGN(outBuff11r,128);

int outBuff1r[OUTPUT_ARRAY];//OUTPUT_ARRAY = 48
int outBuff11r[OUTPUT_ARRAY];


#pragma DATA_ALIGN( edmaBuff,128);
#pragma DATA_ALIGN(edmaBuff9,128);
int  edmaBuff[BUFF_SZ_12]; //BUFF_SZ_12 = 12
int edmaBuff9[BUFF_SZ_12];


/************************************/
#pragma DATA_ALIGN(fftBuffer ,128);
#pragma DATA_ALIGN(fftBuffer9,128);


float  fftBuffer[ARRAY_SIZE*2];
float  fftBuffer9[ARRAY_SIZE*2];

/************************************/
#pragma DATA_ALIGN(intfftBuffer,128);
#pragma DATA_ALIGN(intfftBuffer9,128);

int   intfftBuffer[ARRAY_SIZE*2];//64 * 2
int   intfftBuffer9[ARRAY_SIZE*2];

/************************************/
float twiddle[PTS*2];
short index[SQRT_PTS];
int   channel[CH_SZ]; //CH_SZ=16
int   preChannel[PCH_SZ];//PCH_SZ=4
int   temp[10];
/************************************/
/* global variable*/
static volatile int RX_once = FALSE;
static volatile int count  = 0;
static volatile int count2  = 0;

static volatile float  max;
static volatile float  min;
/************************************/

/* DECLARE SOME CSL OBJECTS */
EDMA_Config cfgEdma;  /* EDMA configuration structure          */
```

Constellation Table Below

```
struct POINT constellation[NUM_CONSTELATION_POINTS] =
{
  /*Points for input binary sequence 0-31decimal
   *Are located on top half of constelation
   */

      /*0 to 7*/
      {-0x7000, 0x7000},
      {-0x5000, 0x7000},
      {-0x1000, 0x7000},
      {-0x3000, 0x7000},
      {0x3000, 0x7000},
      {0x5000, 0x7000},
      {0x1000, 0x7000},
      {0x7000, 0x7000},

      /*7 to 15*/
      {-0x7000, 0x5000},
      {-0x5000, 0x5000},
      {-0x1000, 0x5000},
      {-0x3000, 0x5000},
      {0x3000, 0x5000},
      {0x5000, 0x5000},
      {0x1000, 0x5000},
      {0x7000, 0x5000},

      /*16 to 23*/
      {-0x7000, 0x1000},
      {-0x5000, 0x1000},
      {-0x1000, 0x1000},
      {-0x3000, 0x1000},
      {0x3000, 0x1000},
      {0x5000, 0x1000},
      {0x1000, 0x1000},
      {0x7000, 0x1000},

      /*24 to 31*/
      {-0x7000, 0x3000},
      {-0x5000, 0x3000},
      {-0x1000, 0x3000},
      {-0x3000, 0x3000},
      {0x3000, 0x3000},
      {0x5000, 0x3000},
      {0x1000, 0x3000},
```

```
        {0x7000, 0x3000},

        /*32 to 39*/
        {-0x7000, -0x3000},
        {-0x5000, -0x3000},
        {-0x1000, -0x3000},
        {-0x3000, -0x3000},
        {0x3000, -0x3000},
        {0x5000, -0x3000},
        {0x1000, -0x3000},
        {0x7000, -0x3000},

        /*40 to 47*/
        {-0x7000, -0x5000},
        {-0x5000, -0x5000},
        {-0x1000, -0x5000},
        {-0x3000, -0x5000},
        {0x3000, -0x5000},
        {0x5000, -0x5000},
        {0x1000, -0x5000},
        {0x7000, -0x5000},

        /*48 to 55*/
        {-0x7000, -0x1000},
        {-0x5000, -0x1000},
        {-0x1000, -0x1000},
        {-0x3000, -0x1000},
        {0x3000, -0x1000},
        {0x5000, -0x1000},
        {0x1000, -0x1000},
        {0x7000, -0x1000},

        /*56 to 63*/
        {-0x7000, -0x7000},
        {-0x5000, -0x7000},
        {-0x1000, -0x7000},
        {-0x3000, -0x7000},
        {0x3000, -0x7000},
        {0x5000, -0x7000},
        {0x1000, -0x7000},
        {0x7000, -0x7000},
};

/*-MAIN----------------------------------------------------------------------*/
```

```
void main(){

 int v;



  for (v=0; v < BUFF_SZ ;v++) { /* Initialize the Outbuff */
    edmaInbuff[v]  =0x00000000;
  }//End- for

  CACHE_flush(CACHE_L2,edmaInbuff ,BUFF_SZ);


/*--INITIALIZE-----------------------------------------------------------------*/
    //Taking 32PTS FFT. Twiddle is 64 fftBuffer
    //Prepare Twiddle factors
    gen_w_r2(twiddle, PTS);

    //Apply bit reversal on twiddle factors
    bit_rev(twiddle, PTS>>1);

    bitrev_index(index, PTS);
/*-----------------------------------------------------------------------------*/
  /* Let's disable/clear related interrupts just in case they are pending */
  /* fram a previous run of the program.                                  */
  IRQ_reset(IRQ_EVT_EDMAINT);

  /*Disable EDMA interrupts*/
  EDMA_intDisable(TCCINTNUM_TX2);
  EDMA_intDisable(TCCINTNUM_RX2);

  /*Clear EDMA interrupts*/
  EDMA_intClear(TCCINTNUM_TX2);
  EDMA_intClear(TCCINTNUM_RX2);

  /* Although not required, let's clear all of the EDMA parameter RAM. */
  /* This makes it easier to view the RAM and see the changes as we    */
  /* configure it.                                                     */
   EDMA_clearPram(0x00000000);


  /* Now let's program up the EDMA channel with the configuration structure */
  EDMA_config(hEdmaCha2, &cfgEdma);
  EDMA_config(hEdmaCha14, &cfgEdmaTX_port1);
  EDMA_config(hEdmaCha15, &cfgEdmaRX_port1);
```

```
//EDMA_config(hEdmaCha12, &cfgEdmaTX_port0);
//EDMA_config(hEdmaCha13, &cfgEdmaRX_port0);

/* Let's also configure the reload parameter tables in the EDMA PRAM */
/* with the values in the configuration structures.          */

/*Port 1*//*PaRAM Handle with Configuration Handle*/
 EDMA_config(hEdmaTX_port1  , &cfgEdmaTX_port1  );
 EDMA_config(hEdmaTX_port1_2, &cfgEdmaTX_port1_2);
 EDMA_config(hEdmaTX_port1_3, &cfgEdmaTX_port1_3);
 EDMA_config(hEdmaTX_port1_4, &cfgEdmaTX_port1_4);
 EDMA_config(hEdmaTX_port1_5, &cfgEdmaTX_port1_5);
 EDMA_config(hEdmaTX_port1_6, &cfgEdmaTX_port1_6);
 EDMA_config(hEdmaTX_port1_7, &cfgEdmaTX_port1_7);
 EDMA_config(hEdmaTX_port1_8, &cfgEdmaTX_port1_8);
 EDMA_config(hEdmaTX_port1_9,  &cfgEdmaTX_port1_9);
 EDMA_config(hEdmaTX_port1_10, &cfgEdmaTX_port1_10);
 EDMA_config(hEdmaTX_port1_11, &cfgEdmaTX_port1_11);

/* Enable the interrupts and EDMA interrupts */
IRQ_enable(IRQ_EVT_EDMAINT);
EDMA_intEnable(TCCINTNUM_RX2);
EDMA_intEnable(TCCINTNUM_TX2);

/* Link to next transfer configuration Enable the EDMA channel */
EDMA_link(hEdmaCha14,hEdmaTX_port1);

/*Enable the EDMA Channels*/
 EDMA_enableChannel(hEdmaCha14);
 EDMA_enableChannel(hEdmaCha15);

/*To ensure first read is a new buffer? Do this to be sure about input data. */
CACHE_clean(CACHE_L2,edmaInbuff,BUFF_SZ);

/*Start serial port 1*/
  MCBSP_start(hMcbsp1,MCBSP_RCV_START | MCBSP_XMIT_START
              MCBSP_SRGR_START| MCBSP_SRGR_FRAMESYNC, 0);

} /*End of main*/


/*This is the software routine when called upon in the hardware interrupt routine*/
void swiRXFunc(int arg){

  int *inbuff;
  int x;
```

```
int v;

inbuff = edmaInbuff;

/* Now let's process the input buffer, by copy it to secondary buffers to avoid it being
   overwritten. Each buffer hold enough data for one OFDM symbol*/

 for (x=0; x<(BUFF_SZ_12); x++) {//BUFF_SZ_12 = 12
  edmaBuff[x]    = inbuff[x];
  edmaBuff1[x] = inbuff[x+12];
  edmaBuff2[x] = inbuff[x+24];
  edmaBuff3[x] = inbuff[x+36];
  edmaBuff4[x] = inbuff[x+48];
  edmaBuff5[x] = inbuff[x+60];
  edmaBuff6[x] = inbuff[x+72];
  edmaBuff7[x] = inbuff[x+84];
  edmaBuff8[x] = inbuff[x+96];
  edmaBuff9[x] = inbuff[x+108];
 }//End-for

/*Now do serial to parallel conversion      */
/* Then map onto QAM64 Constellation        */
/* by calling our 'stp' function            */
/*  -edmaBuffx contains input               */
/*  -fftBufferx contains output             */
stp(fftBuffer ,edmaBuff);
stp(fftBuffer1,edmaBuff1);
stp(fftBuffer2,edmaBuff2);
stp(fftBuffer3,edmaBuff3);
stp(fftBuffer4,edmaBuff4);
stp(fftBuffer5,edmaBuff5);
stp(fftBuffer6,edmaBuff6);
stp(fftBuffer7,edmaBuff7);
stp(fftBuffer8,edmaBuff8);
stp(fftBuffer9,edmaBuff9);

/*Need to 'bit reversal' all inputs first */
/*before using IFFT                       */

DSPF_sp_bitrev_cplx(fftBuffer, index, PTS);
DSPF_sp_bitrev_cplx(fftBuffer1, index, PTS);
DSPF_sp_bitrev_cplx(fftBuffer2, index, PTS);
DSPF_sp_bitrev_cplx(fftBuffer3, index, PTS);
DSPF_sp_bitrev_cplx(fftBuffer4, index, PTS);
DSPF_sp_bitrev_cplx(fftBuffer5, index, PTS);
DSPF_sp_bitrev_cplx(fftBuffer6, index, PTS);
```

```
DSPF_sp_bitrev_cplx(fftBuffer7, index, PTS);
DSPF_sp_bitrev_cplx(fftBuffer8, index, PTS);
DSPF_sp_bitrev_cplx(fftBuffer9, index, PTS);

/*Apply the IFFT                    */
/*Data and Twiddle fftBuffers should be float types*/
/*Array Size: fftBuffer 128, twiddle 128, PTS 64   */
DSPF_sp_icfftr2_dif(fftBuffer, twiddle, PTS);
DSPF_sp_icfftr2_dif(fftBuffer1, twiddle, PTS);
DSPF_sp_icfftr2_dif(fftBuffer2, twiddle, PTS);
DSPF_sp_icfftr2_dif(fftBuffer3, twiddle, PTS);
DSPF_sp_icfftr2_dif(fftBuffer4, twiddle, PTS);
DSPF_sp_icfftr2_dif(fftBuffer5, twiddle, PTS);
DSPF_sp_icfftr2_dif(fftBuffer6, twiddle, PTS);
DSPF_sp_icfftr2_dif(fftBuffer7, twiddle, PTS);
DSPF_sp_icfftr2_dif(fftBuffer8, twiddle, PTS);
DSPF_sp_icfftr2_dif(fftBuffer9, twiddle, PTS);


/* Normalize IFFT output manually*/
for(v=0; v< CHANNELS*2; v++) {

   fftBuffer[v]  = fftBuffer[v]/PTS;
   fftBuffer1[v] = fftBuffer1[v]/PTS;
   fftBuffer2[v] = fftBuffer2[v]/PTS;
   fftBuffer3[v] = fftBuffer3[v]/PTS;
   fftBuffer4[v] = fftBuffer4[v]/PTS;
   fftBuffer5[v] = fftBuffer5[v]/PTS;
   fftBuffer6[v] = fftBuffer6[v]/PTS;
   fftBuffer7[v] = fftBuffer7[v]/PTS;
   fftBuffer8[v] = fftBuffer8[v]/PTS;
   fftBuffer9[v] = fftBuffer9[v]/PTS;

}//End- for


   /* first zero are two max & min variables
    * then do a earch through all buffers to find
    * the max and min values out of IFFT func
    */

   max =0; min =0;

   maxMin(fftBuffer);
   maxMin(fftBuffer1);
   maxMin(fftBuffer2);
```

```
maxMin(fftBuffer3);
maxMin(fftBuffer4);
maxMin(fftBuffer5);
maxMin(fftBuffer6);
maxMin(fftBuffer7);
maxMin(fftBuffer8);
maxMin(fftBuffer9);

            /*With 'min' found, offset all values
             *to make the new 'min' value zero
             *(-min) is used b/c 'min' is negative
             */

for(v=0;v<ARRAY_SIZE_2;v++){
 fftBuffer[v]  = fftBuffer[v]  +( -min);
 fftBuffer1[v] = fftBuffer1[v] +( -min);
 fftBuffer2[v] = fftBuffer2[v] +( -min);
 fftBuffer3[v] = fftBuffer3[v] +( -min);
 fftBuffer4[v] = fftBuffer4[v] +( -min);
 fftBuffer5[v] = fftBuffer5[v] +( -min);
 fftBuffer6[v] = fftBuffer6[v] +( -min);
 fftBuffer7[v] = fftBuffer7[v] +( -min);
 fftBuffer8[v] = fftBuffer8[v] +( -min);
 fftBuffer9[v] = fftBuffer9[v] +( -min);
}//End-for

/*The new maximum value also accounts for scale*/
max = max +( -min);

/*Scale all values between 0 and 4000*/
for(v=0;v<ARRAY_SIZE_2;v++){
 fftBuffer[v]  = fftBuffer[v]  *( 4000/max);
 fftBuffer1[v] = fftBuffer1[v] *( 4000/max);
 fftBuffer2[v] = fftBuffer2[v] *( 4000/max);
 fftBuffer3[v] = fftBuffer3[v] *( 4000/max);
 fftBuffer4[v] = fftBuffer4[v] *( 4000/max);
 fftBuffer5[v] = fftBuffer5[v] *( 4000/max);
 fftBuffer6[v] = fftBuffer6[v] *( 4000/max);
 fftBuffer7[v] = fftBuffer7[v] *( 4000/max);
 fftBuffer8[v] = fftBuffer8[v] *( 4000/max);
 fftBuffer9[v] = fftBuffer9[v] *( 4000/max);
}//End-for

   /*convert to int
    *Casting floating pt to integer will result in
    *information loss
```

```
    */

    for (x=0; x<ARRAY_SIZE_2; x++) {
    intfftBuffer[x]   = fftBuffer[x];
    intfftBuffer1[x]  = fftBuffer1[x];
    intfftBuffer2[x]  = fftBuffer2[x];
    intfftBuffer3[x]  = fftBuffer3[x];
    intfftBuffer4[x]  = fftBuffer4[x];
    intfftBuffer5[x]  = fftBuffer5[x];
    intfftBuffer6[x]  = fftBuffer6[x];
    intfftBuffer7[x]  = fftBuffer7[x];
    intfftBuffer8[x]  = fftBuffer8[x];
    intfftBuffer9[x]  = fftBuffer9[x];
    }//End for

      /* Perform parallel to serial format
       * and pack data together
       *  -Input is 'intfftBuffer' with scaled IFFT output from 0 to 4000
       *  -Output is outBuffXr
       */

    pts(outBuff1r, intfftBuffer);
    pts(outBuff2r, intfftBuffer1);
    pts(outBuff3r, intfftBuffer2);
    pts(outBuff4r, intfftBuffer3);
    pts(outBuff5r, intfftBuffer4);
    pts(outBuff6r, intfftBuffer5);
    pts(outBuff7r, intfftBuffer6);
    pts(outBuff8r, intfftBuffer7);
    pts(outBuff9r, intfftBuffer8);
    pts(outBuff10r, intfftBuffer9);

    CACHE_flush(CACHE_L2,outBuff1r,OUTPUT_ARRAY);
    CACHE_flush(CACHE_L2,outBuff2r,OUTPUT_ARRAY);
    CACHE_flush(CACHE_L2,outBuff3r,OUTPUT_ARRAY);
    CACHE_flush(CACHE_L2,outBuff4r,OUTPUT_ARRAY);
    CACHE_flush(CACHE_L2,outBuff5r,OUTPUT_ARRAY);
    CACHE_flush(CACHE_L2,outBuff6r,OUTPUT_ARRAY);
    CACHE_flush(CACHE_L2,outBuff7r,OUTPUT_ARRAY);
    CACHE_flush(CACHE_L2,outBuff8r,OUTPUT_ARRAY);
    CACHE_flush(CACHE_L2,outBuff9r,OUTPUT_ARRAY);
    CACHE_flush(CACHE_L2,outBuff10r,OUTPUT_ARRAY);

/* Since we're done processing the input buffer, clean it from cache, */
/* this invalidates it from cache to ensure we read a fresh version    */
/* the next time.                                                      */
```

```
 /*We do not need fftBuffer or edmaInbuff                    */

  CACHE_clean(CACHE_L2,fftBuffer,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,fftBuffer1,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,fftBuffer2,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,fftBuffer3,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,fftBuffer4,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,fftBuffer5,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,fftBuffer6,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,fftBuffer7,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,fftBuffer8,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,fftBuffer9,ARRAY_SIZE_2);

  CACHE_clean(CACHE_L2,intfftBuffer,ARRAY_SIZE_2);//MOD
  CACHE_clean(CACHE_L2,intfftBuffer1,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,intfftBuffer2,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,intfftBuffer3,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,intfftBuffer4,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,intfftBuffer5,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,intfftBuffer6,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,intfftBuffer7,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,intfftBuffer8,ARRAY_SIZE_2);
  CACHE_clean(CACHE_L2,intfftBuffer9,ARRAY_SIZE_2);

  CACHE_clean(CACHE_L2,inbuff,BUFF_SZ);

 /*This should not be required but done:
  *With the processing finished
  *We can set the counter to zero so the next symbol transmitted
  *will be the first one. See the interrupt serivice routine for
  *details
  */
    count=0;

  /*Enable the service routine to link with other buffers for
   *Transmission
   */

    RX_once = TRUE;

}// End - swiProcessFunc

/*-------------------------------------------------------------------------*/
/*Hardware Interrupts are handled here.
/*-------------------------------------------------------------------------*/
```

```
void hwiEdmaIsr(int arg){


 /*-RX Port1-------------------------*/

 if (EDMA_intTest(TCCINTNUM_RX2)) { //BB

  if(!RX_once){
   EDMA_intClear(TCCINTNUM_RX2);

     /*Print out the first buffer values to
      *Code Composers Message Log
      * The user inspects if its correct b/c
      * the intial bits would be known
      */

     LOG_printf(&trace,"%d", edmaInbuff[0]);

     /*Data is ready to be proceeded*/
     SWI_post(&swiRX);

  }

  else{
    /*Just clear the Interrupt */
    EDMA_intClear(TCCINTNUM_RX2);

  }//End-else

  }//End-if


 /*-TX Port1-------------------------------------------------------------------*/
 /* Flush before writing to Port                                              */
 /* Clean to eliminate                                                        */
 /*-TX Port1-------------------------------------------------------------------*/

 if (EDMA_intTest(TCCINTNUM_TX2)) { //CC

    EDMA_intClear(TCCINTNUM_TX2);

    if(RX_once) {
```

```
if(count == 0){
  EDMA_link(hEdmaCha14,hEdmaTX_port1);
  count++;
  LOG_printf(&trace,"Outbuffer1i");
}//End if(outBuffer1Busy)

else if(count == 1){
  EDMA_link(hEdmaCha14,hEdmaTX_port1_2);
  LOG_printf(&trace,"Outbuffer2i");
  count++;
 }//End else

else if(count == 2){
  EDMA_link(hEdmaCha14,hEdmaTX_port1_3);
  LOG_printf(&trace,"Outbuffer3i");
  count++;
 }//End else

else if(count == 3){
  EDMA_link(hEdmaCha14,hEdmaTX_port1_4);
  LOG_printf(&trace,"Outbuffer4i");
  count++;
 }//End else

else if(count == 4){
  EDMA_link(hEdmaCha14,hEdmaTX_port1_5);
  LOG_printf(&trace,"Outbuffer5i");
  count++;
 }//End else

else if(count == 5){
  EDMA_link(hEdmaCha14,hEdmaTX_port1_6);
  LOG_printf(&trace,"Outbuffer6i");
  count++;
 }//End else

else if(count == 6){
  EDMA_link(hEdmaCha14,hEdmaTX_port1_7);
  LOG_printf(&trace,"Outbuffer7i");
  count++;
 }//End else

else if(count == 7){
  EDMA_link(hEdmaCha14,hEdmaTX_port1_8);
  LOG_printf(&trace,"Outbuffer8i");
  count++;
```

```
        }//End else

       else if(count == 8){
         EDMA_link(hEdmaCha14,hEdmaTX_port1_9);
         LOG_printf(&trace,"Outbuffer9i");
         count++;
        }//End else

       else if(count == 9){
         EDMA_link(hEdmaCha14,hEdmaTX_port1_10);
         LOG_printf(&trace,"Outbuffer10i");
         count++;
        }//End else

       else{
         EDMA_link(hEdmaCha14,hEdmaTX_port1_11);
         LOG_printf(&trace,"Outbuffer11i");
         count = 0;
        }//End else

     }//End - RX_once

    else {
                /*Buffers not ready to transmit OFDM data
                 *Here we just send out the first buffer
                 *
                 /
        EDMA_link(hEdmaCha14,hEdmaTX_port1);
    }//End-else

     }//End-CC


}//End of HWI routine

/*-------------------------------------------------------------------------*/

/*************************************************************************
*******\
* End of main.c
\*************************************************************************
*******/


/*************************************************************************
* This is the serial to parallel conversion function
```

```
* After the bits have been grouped in to their parallel array
* QAM64 Mapping will be applied
* Inputs:
* -'in' buffer is an 12 element array
* Outputs:
* -floating buffer size 128 elements ready for IFFT
*
*
**************************************************************************/

void stp(float *buffer, int *in){

  int i;
  int v;
  v=0;

  //Every
  for(i=0; i<4; i++) {


      /*Extract 6-bits in sequential manner through
       *the 'in' buffer array and store them in temp 'channel'
       *holders
       *
       */
      channel[0] = (in[v]  & 0xFC000000) >> 26;
      channel[1] = (in[v]  & 0x03F00000) >> 20;
      channel[2] = (in[v]  & 0x000FC000) >> 14;
      channel[3] = (in[v]  & 0x00003F00) >> 8;
      channel[4] = (in[v]  & 0x000000FC) >> 2;

    preChannel[0] = (in[v]  & 0x00000003) << 4;
    preChannel[1] = (in[v+1]& 0xF0000000) >> 28;
      channel[5] = (preChannel[0] | preChannel[1]);

      channel[6] = (in[v+1]& 0x0FC00000) >> 22;
      channel[7] = (in[v+1]& 0x003F0000) >> 16;
      channel[8] = (in[v+1]& 0x0000FC00) >> 10;
      channel[9] = (in[v+1]& 0x000003F0) >> 4;

    preChannel[2] = (in[v+1]& 0x0000000F) << 2;
    preChannel[3] = (in[v+2]& 0xC0000000) >> 30;
      channel[10] = (preChannel[2] | preChannel[3]);

      channel[11] = (in[v+2]& 0x3F000000) >> 24;
      channel[12] = (in[v+2]& 0x00FC0000) >> 18;
```

```
 channel[13] = (in[v+2]& 0x0003F000) >> 12;
 channel[14] = (in[v+2]& 0x00000FC0) >> 6;
 channel[15] = (in[v+2]& 0x0000003F) >> 0;

/*Advance the index*/
v = v + 3;

/*Using the grouped bits in 'channel', look up
 *the constellation table and map them to symbol values.
 *The symbol values are stored in the output 'buffer'
 *array
 */

buffer[i*32]   = constellation[ channel[0] ].I;
buffer[i*32+1] = constellation[ channel[0] ].Q;

buffer[i*32+2] = constellation[ channel[1] ].I;
buffer[i*32+3] = constellation[ channel[1] ].Q;

buffer[i*32+4] = constellation[ channel[2] ].I;
buffer[i*32+5] = constellation[ channel[2] ].Q;

buffer[i*32+6] = constellation[ channel[3] ].I;
buffer[i*32+7] = constellation[ channel[3] ].Q;

buffer[i*32+8]  = constellation[ channel[4] ].I;
buffer[i*32+9]  = constellation[ channel[4] ].Q;

buffer[i*32+10] = constellation[ channel[5] ].I;
buffer[i*32+11] = constellation[ channel[5] ].Q;

buffer[i*32+12] = constellation[ channel[6] ].I;
buffer[i*32+13] = constellation[ channel[6] ].Q;

buffer[i*32+14] = constellation[ channel[7] ].I;
buffer[i*32+15] = constellation[ channel[7] ].Q;

buffer[i*32+16] = constellation[ channel[8] ].I;
buffer[i*32+17] = constellation[ channel[8] ].Q;

buffer[i*32+18] = constellation[ channel[9] ].I;
buffer[i*32+19] = constellation[ channel[9] ].Q;

buffer[i*32+20] = constellation[ channel[10] ].I;
buffer[i*32+21] = constellation[ channel[10] ].Q;
```

```
    buffer[i*32+22] = constellation[ channel[11] ].I;
    buffer[i*32+23] = constellation[ channel[11] ].Q;

    buffer[i*32+24] = constellation[ channel[12] ].I;
    buffer[i*32+25] = constellation[ channel[12] ].Q;

    buffer[i*32+26] = constellation[ channel[13] ].I;
    buffer[i*32+27] = constellation[ channel[13] ].Q;

    buffer[i*32+28] = constellation[ channel[14] ].I;
    buffer[i*32+29] = constellation[ channel[14] ].Q;

    buffer[i*32+30] = constellation[ channel[15] ].I;
    buffer[i*32+31] = constellation[ channel[15] ].Q;

  }//End-for

}//End stp

/*PTS packs the output data from IFFT
 *into output buffers for serial transmission.
 *
 *
 */
void pts(int *oBuffer, int *in){

  int i, v;

  /*We do not have to do this but reduces any chances of errors*/
  /*Zero all bits except for the 12 least significant bits    */

  for(i=0; i<(ARRAY_SIZE_2); i++) {
      in[i] = (in[i] & 0x00000FFF);
    }//End -for

  //Initalize counter v
  v=0;
  /*Pack the bits into the output buffer*/
  for(i=0; i<(ARRAY_SIZE_2/8); i++) {
      temp[0]  = (in[i*8])   << 20; //0xFFF00000
      temp[1]  = (in[i*8+1]) << 8;  //0x000FFF00
      temp[2]  = (in[i*8+2]) >> 4;  //0x000000FF

      temp[3]  = (in[i*8+2]) << 28; //0xF0000000
      temp[4]  = (in[i*8+3]) << 16; //0x0FFF0000
      temp[5]  = (in[i*8+4]) << 4;  //0x0000FFF0
```

```
        temp[6]  = (in[i*8+5]) >> 8;  //0x0000000F

        temp[7]  = (in[i*8+5]) << 24; //0xFF000000
        temp[8]  = (in[i*8+6]) << 12; //0x00FFF000
        temp[9]  = (in[i*8+7])     ; //0x00000FFF

        oBuffer[v*3]   = (temp[0] | temp[1] | temp[2]);
        oBuffer[v*3+1] = (temp[3] | temp[4] | temp[5] | temp[6]);
        oBuffer[v*3+2] = (temp[7] | temp[8] | temp[9]);
        v++; //Increment Counter//0,1,2,3....15

    }//End -for

}//End pts


/*Function: maxMin
 *Performs search to find max and min values for a
 *given output buffer
 */
void maxMin(float *in) {

  int v;

      for (v=0; v<ARRAY_SIZE_2; v++)//ARRAY_SIZE_2 = 128
      {
        if (in[v] > max)
                max=in[v];

        if (in[v] < min)
           min=in[v];

                }//End-for
}
```

Appendix B

B1: DSP Peripheral Configuration

This section provides a tabulated configuration summary of serial port McBSP1 and the EDMA channels to service this port in the DSP/BIOS GUI.

Table B-1. McBSP1 [TX] Configuration

Field	Symbol	Setting	Comments
Operating Mode			
Digital Loop Back	DLB	Disabled	• Enabled when doing a internal loop back test on the same port
Transmit Mode			
Interrupt Mode	XINTM	XRDY	• Interrupt type generated when transmitter ready • Used to Acknowledge EDMA when to move element/word into transmit register
Bit Synch. Error	XSYNCERR	Clear	• For error event purposes when transmitter data is overwritten before transmission
Clock Mode	CLKXM	Output/McBSP-master	• Ouput Master clock
Clock Polarity	CLKXP	Rising Edge	• Proper polarity
Frame Synch Mode	FSXM	Sample rate generator	• Clock and frame synch derived from internal timing

Table B-1 Continued. McBSP1 [TX] Configuration

Field	Symbol	Setting	Comments
Frame Synch. Polarity	FSXP	Active high	• High when synchronization pulse active
DX Enabler		Enable	• Introduces additional delay for data transmission • DX stands for the transmission pin
Unexpected Frame Pulse	XFIG	Pulse Ignored	• Not applicable because the port is the master
Data Delay	XDATDLY	1bit	• For maximum frame frequency
Transmit Frame\Element			
Phase(s)	XPHASE	Single phase	• All 32bit elements
Words per phase		126	• Transmit 2 PN sequences • Ideally 127 would provide 25.175Mps, but one word must be dropped to continuous operation between input and output.
Element length	XWDLEN(1)	32bits	• To minimize CPU interrupt frequency
Sample Rate Generator			
Sample Rate Gen Clock Mode	CLKSM	Internal Synch	• Use external CLKS pin to for external clock reference
Transmit Frame Synch mode	FSGM	Frame Synch Signal	• Synchronization will be generated based on the setting for frame period listed below
Frame Period		4064	• Two full PN sequences before generating frame synch
Frame Width		1	• Synch. Pulse is 1 clock period
Clock Freq Divider		3	• Equivalent to 25MHz clock from CPU • Max frequency is CPU/2 • Max frequency is 75MHz

Table B-2. McBSP1 [RX] Configuration

Description	Symbol	Setting	Comments
Operating Mode			
Digital Loopback Mode	DLB	Disabled	• None
Receiver Mode			
Interrupt Mode	RINTM	End of Sub frame	• Interrupt signals EDMA when ready for read
Bit Synch. Error	RSYNCERR	Clear	• Error detection purposes
Clock Mode	CLKRM	Input	• Acting as slave; timing off master[Altera]
Clock Polarity	CLKRP	Falling Edge	• None
Frame Synch Mode	FSRM	External Source	• Frame from Altera
Frame Synch Polarity	FSRP	Active High	• Frame detected when value high
Data Justification	RJUST	Right/zero-fill	• Proper justification • Least significant bit to right
Unexpected Frame Pulse	RFIG	Pulse Ignored	• There should be no unexpected pulse from Altera, if so they should be ignored
Data Delay	RDATDLY	1bit	• For max frame frequency
Receive Frame\Element			
Phase(s)	RPHASE	Single phase	• All 32bit elements
Words per phase		127	• Equivalent to 2 full PN sequences
Element length	RWDLEN	32bits	• To minimize CPU interrupt frequency

Table B-3. EDMA Configuration for Channels 14[TX]/15[RX]

Description	Symbol	Setting	Comments
Operation Mode			
Frame Synch	FS	None	• Not needed
Element Size	ESIZE	32bits	• Same as McBSP received per word
Priority Levels	PRI	High	• Priority set high importance for CPU
Two Dimensional Source Transfer	2DS	Disable	• Not Needed
Source Address Update Mode	SUM	None[RX] Incremental[TX]	• Incremental when writing or reading to buffer. • Fix when reading/writing to serial port register
Source		McBSP DRR[RX] L2 Cache OuputBuffer [TX]	• Data source EDMA transports from
Two Dimensional Destination Transfer	2DD	Disable	• Not Needed
Destination Address update mode	DUM	Incremental [RX] None [TX]	• Incremental b/c of buffer
Destination		L2 Cache InputBuffer [RX] McBSP DXR[TX]	• Data destination EDMA transport to
Link Event	LINK	Enable	• Allows for loading new PaRAM configuration

Table B-3 Continued. EDMA Configuration for Channels 14[TX]/15[RX]

Description	Symbol	Setting	Comments
Index			
Frame Index	FIX	0x0000	• Single frame • No indexing needed
Element Index	EIX	0x0004	• Index for 32bit element size from destination or source
Transfer Count			
Transfer Counter Format		Numeric	
Frame Counter	FC	0x0000	• Not needed
Element Counter	EC	0x007F[RX] 0x007E[TX]	• 127 Elements [RX] • 126 Elements [TX]
Element Count Reload	ECRLD	0x0000	• Not needed
Transfer Complete			
Transfer Complete Interrupt	TCINT	Enable	• Enable interrupt when transfer completed
Transfer Complete Code	TCC	15 [RX] 14 [TX]	• Code posted when transfer for every 127Elements moved by EDMA

Appendix C

C1: Verilog Source Code for CPLD

In this section, just the main Verilog code for the 'serial-to-parallel' and 'LFSR PN generator' modules synthesized onto the CPLD are included.

```
/*****************************************************/
 *THIS IS THE LFSR BLOCK
 *Output: data – serial data taken from LFSR
           synch – synchronization pulse to mark beginning
                   of 'data' sequence
/*****************************************************/
 always @(posedge clk or posedge reset_clean)
  begin

/* If reset is true, then:
 *       -Reload the LFSR with 12'b000000000001
 *       -Ensure synch pulse is low
 *Else shift data in LFSR
 */

   if(reset_clean)
    begin
     regan <= 1;
     synch <= 0;

    end

   else
    begin
     //Only one of the two lines below are active.

     //For 12th order LFSR use below
     regan[n:1] <= {regan[n-1:8],regan[n]^regan[7],regan[6:5],regan[n]^regan[4],regan[n]^regan[3],regan[2:1],regan[n]};

     //For 13th order LFSR use below
     //regan[n:1] <= {regan[n-1:5],regan[n]^regan[4],regan[n]^regan[3],regan[2], regan[n]^regan[1],regan[n]};
```

```
      /*Synch Pulse is generated when the LFSR matches with the selected binary sequence below */
       if(regan == 12'b000000000010) // use 'if(regan == 13'b0000000000010)' for 13th order LSFR
         synch <=1;
       else
         synch <=0;
   end

  end

// 'data' is the assigned serial data output from 'x1'
assign data = regan[1];

/*****************************************************/
/* This is the serial-to-parallel conversion block
 * countDD is 5-bit counter which counts from 1 to 24
 */
/*****************************************************/

  always @(posedge clk or posedge reset_clean)
   begin
     if(reset_clean)
      dac <=0;

     else
      begin
      if(countDD == 5'b00000) dac   <=in_data_synch; //This state should never occur
      if(countDD == 5'b00001)
         begin  //Begin shifting external data into shift register
            dac[24] <=in_data_synch;
            dac[1:23] <=0;
         end

      //sequential incoming data shifted into shift register.
      if(countDD > 1)
         begin
          dac[24] <=in_data_synch;
          dac[1:23] <= dac[2:24]
         end
     end
   end


/*****************************************************/
/*COUNTER FOR STP                    */
/* This generates the 'countDD' for the serial-to-parallel converter
/* It counts from 1 to 24 and resets to 1 when an synchronization pulse
 * is received and aligned from the external DSP
/*****************************************************/

  always @(posedge clk or posedge reset_clean)
   begin
     if(reset_clean)
      begin
       countDD <= 1;
      end
```

```
      else if( synch_in_synch)
       begin
        countDD <= 1;
       end

      else if(countDD > 23)  //MOD
       begin
        countDD <=1;
       end

      else
       begin
          countDD <= countDD + 1;
       end

  end

/*****************************************************/
  always @(posedge clk)
   begin
    if(countDD == 1)
       count1 <= 1;
    else
       count1 <= 0;
   end

/*****************************************************/
// This block dumps the data out from the serial-to-parallel shift
// register to the DACs
// Input: dac[0:23]
// Output: dac_out[0:23]
/*****************************************************/
   always @(posedge clk or posedge reset_clean)
    begin
     if(reset_clean)
       dac_out <= 0;
     else if(countDD==1) //OK
       dac_out <= dac;
     else
       dac_out <= dac_out;
    end

/*****************************************************/
//Generate the DAC CLOCK reference clock , which is divided down
//by 24 from clk = 25.175MHz
// Output: clk_dac = 1.049MHz
/*****************************************************/
   always @(posedge clk)

//COUNTEE is a register that counts from 0 to 23
//and resets when an incoming 'synch_in_delay' pulse arrives
//to synchronize the clock with the incoming/outgoing data to DAC

    begin
     if(synch_in_delay || (COUNTEE > 22) ) //MOD
      COUNTEE =0;
```

```
      else
       COUNTEE = COUNTEE + 1;

//the output clock is clk_dac and is low when COUNTEE is 0 to 11
// and high when COUNTEE is 12 to 23

      if(COUNTEE < 12)//MOD
        begin
        clk_dac = 0;
        end

      else
        begin
        clk_dac = 1;
        end
    end

/*****************************************************/
// This block is used to realign the incoming serial data
//  and synch pulse (To be used by the scope). Both signals are
// arriving from the DSP
// Inputs:  in_data, synch_in;  ( Both from DSP )
// Outputs: in_data_synch, synch_in_synch
/*****************************************************/
    always @(posedge clk or posedge reset_clean)
     begin
      if(reset_clean)
       begin
        in_data_synch  <= 0;
        synch_in_synch <= 0;
       end
       else

       //Capture incoming signals using flip-flop
       begin
        in_data_synch <= in_data;
        synch_in_synch_temp<= synch_in;
        synch_in_synch    <= synch_in_synch_temp;
       end
     end
```

D1: Simulation Matlab Code

In this section, the simulation code written in Matlab language is provided for 64QAM modulation scheme. The code for 16QAM and 256QAM follow the same methodology, but differ with the mapping format because of the constellation size.

SCRIPT: main.m
Purpose: Run this script to execute the simulation

```
%THINGS TO SET

nSym = 200; %Number of OFDM Symbols
dcQ = 4; %DACQ output voltage
dcI = 4;  %DACI output voltage
delaySample = 22;  %Vary this to simulate I/Q phase delay
DCoffset =  0.0; %Offset in the received samples
DCoffsetb = 0.0; %redundant

upSample = 5;
Fa=9; %number of samples per symbol during up conversion
    %delaySample needs to be set. ideal its Fa/2 rounded up.

%DECLARE VARIABLES USED IN SCRIPT

rData = zeros(64,nSym);
cData = zeros(64,nSym);
ifft_outR = zeros(64,nSym);
ifft_outI = zeros(64,nSym);
ifft_outR4 = zeros(64,nSym);
ifft_outI4 = zeros(64,nSym);
ifft_outR_s = zeros(64*nSym,1);
ifft_outI_s = zeros(64*nSym,1);
ifft_outR2 = zeros(64,nSym);
ifft_outI2 = zeros(64,nSym);
max0 = 0; max1 =0; max2=0;
min0 = 0; min1 =0; min2=0;
final_demod_real  =zeros(64,nSym);
final_demod_ideal =zeros(64,nSym);
final_points_real  =zeros(64,nSym);
```

```
final_points_ideal =zeros(64,nSym);
igain = 0;
qgain = 0;
compare = zeros(64,nSym);
fft_inRI = zeros(64,nSym);
fft_data = zeros(64,1);
sixtyFour = [1:64];
symCount  = zeros(64);

serial_ifft_outR = zeros(64*nSym,1);
serial_ifft_outI = zeros(64*nSym,1);
new_dataQ = zeros((64*nSym*(Fa+1)),1); %This will hold new
new_dataI = zeros((64*nSym*(Fa+1)),1); %data after upsampling
%-----------------------------------------------------------------------



%Begin here
cd ..\lfsr25
lfsr_17; % Generate the PN data and store in 'rData', which is the output
cd ..\gain_64_40sym_newScaling

loadTable %Script to load constellation table

for ia = 1:nSym

   %Convert the PN random data to symbols using table loaded
   for ib = 1:64
      cData(ib,ia) = tbl( rData(ib,ia),1 );
      symCount( rData(ib,ia) ) = symCount( rData(ib,ia) ) + 1;
   end

   %Perform OFDM modulation on QAM symbols by applying IFFT
      ifft_outR( (1:64), ia) = real(ifft( cData( (1:64),ia) ,64));
      ifft_outI( (1:64), ia) = imag(ifft( cData( (1:64),ia) ,64));

%Find the max and min values in IFFT outputs
      max1 = max( ifft_outR( (1:64), ia));
      max2 = max( ifft_outI( (1:64), ia));
      min1 = min( ifft_outR( (1:64), ia));
      min2 = min( ifft_outI( (1:64), ia));

      if max1>max0
        max0=max1;
      end
      if max2>max0
        max0=max2;
      end
      if min1<min0
        min0=min1;
      end
      if min2<min0;
        min0=min2;
      end
end %End of Modulation Loop
%These are the resultant max and min values in the search
```

```
max0;
min0;

%Capture the ideal constellation points for bit error, QAM symbol error, and SIR computation
final_points_ideal = cData;
final_demod_ideal = rData-1;


%max3 is the new max value after the offset is applied
max3 = max0 - min0;

%Here IFFT output values scaled from 0 to 4095
ifft_outR2 = round((ifft_outR - min0).*(4095/max3));
ifft_outI2 = round((ifft_outI - min0).*(4095/max3));
ifft_outR3 = (ifft_outR2.*(dcI/4095))-(dcI/2)+DCoffsetb; %now its from -2.x to +2.x VOLTS
ifft_outI3 = (ifft_outI2.*(dcQ/4095))-(dcQ/2)+DCoffsetb;

%Data is formatted in a matrix format and is converted to serial data for all
%OFDM symbols together
for aa=1:nSym
    serial_ifft_outR((aa-1)*64+1:(aa)*64,1) = ifft_outR3(1:64,aa);
    serial_ifft_outI((aa-1)*64+1:(aa)*64,1) = ifft_outI3(1:64,aa);

end




%Increase number of data samples per OFDM sample by a factor of 'Fa'

for i=1:(nSym*64)
  for j=1:Fa
    new_dataQ( (i-1)*Fa+j)=serial_ifft_outI(i,1);
    new_dataI( (i-1)*Fa+j)=serial_ifft_outR(i,1);
  end
end

%filter the samples
filter_lpf2;

%Need to collect the filter output at the proper location due to convolution factor
resynch_dataQ = yq(delay:(64*nSym*Fa)+delay);
resynch_dataI = yi(delay:(64*nSym*Fa)+delay);

%Use a 3rd order spline to perform line interpolation to increase resolution
len = length(resynch_dataQ);
xa = transpose(1:(len));
xx = transpose( linspace(1,len,len*upSample) );.
resynch_dataQ2 = spline(xa,resynch_dataQ,xx);
resynch_dataI2 = spline(xa,resynch_dataI,xx);


%Perform Decimation on the over-sampled signal
for ja = 1:(64*nSym)
  %HERE WE SELECT TO BYPASS LPF OR NOT
```

```
   %To use the filtered output—use the two lines below
    %resampled_dataQ(ja,1)=resynch_dataQ2( ((Fa*upSample)-delaySample)+((ja-1)*(Fa*upSample)), 1 );
    %resampled_dataI(ja,1)=resynch_dataI2( ((Fa*upSample)-22)+((ja-1)*(Fa*upSample)), 1 );

   %To use the outputs from the DAC—use the two lines below
    resampled_dataQ(ja,1)=new_dataQ( (Fa-6)+((ja-1)*Fa), 1 );
    resampled_dataI(ja,1)=new_dataI( (Fa-6)+((ja-1)*Fa), 1 );
end


%This section is activated for using an ADC/quantization in the receiver.
%cd ..\quantization
  %Select only one of the two lines below
  %quantize_script_noOffset %GOTO THIS SCRIPT TO set -4 to 4 V or -2 to 2 Volt ADC input
  %quantize_script_LPF_noOffset %This takes samples the filter output
%cd ..\gain_64_40sym_newScaling\

%Demodulation is performed from this point
%First undo the scaling of the signal

in_q=((resampled_dataQ+((dcQ/2))+DCoffset).*(max3/4).*(4095/4095))+min0*(dcQ/4);
in_i=((resampled_dataI+((dcQ/2))+DCoffset).*(max3/4).*(4095/4095))+min0*(dcI/4);

   %Need to convert the serial data of all OFDM symbols, into a group format
   % where the OFDM samples are grouped together a 64xNumberOfSymbols format
   for yb =1:numSym
     wave_in_q(1:64,yb)      = in_q((1+(64*(yb-1))):(64*yb));
     wave_in_inphase(1:64,yb) = in_i((1+(64*(yb-1))):(64*yb));
   end

%Demodulate the OFDM time waveform samples
ber_fft
  %Save demodulated QAM symbols and its equivalent decimal value in its unmapped binary format
  %These values are used for SIR-bit, and MSE constellation. OUTPUTS:
  final_demod_real = final_demod;
  points_in = const_points;

%plot the actual data constellation points
  %yy_q =((resampled_dataQ+((dcQ/2))+DCoffset).*(max3/4).*(4095/4095))+min0*(dcQ/4);
  %yy_inphase =((resampled_dataI+((dcI/2))+DCoffset).*(max3/4).*(4095/4095))+min0*(dcI/4);
  plot_const_serial


%Demodulate the theoretical QAM symbols and Binary data in decimal values
  in_q =serial_ifft_outI3;
  in_i =serial_ifft_outR3;
  ber_fft
   final_demod_ideal = final_demod;
   points_theory = const_points;

%Compare Actual and theory
%Checking for QAM symbol errors
  comparer;
   out_err;

%To find the MSE constellation with/without first FFT output bin
```

```
find_mse;
  %OUTPUT
  mean_ERR_DC  = meanErr

find_mse_noDC;
  %OUTPUT
  mean_ERR_noDC = meanErr

%To find the SIR –bit with/without first FFT output bin
find_sir
  %OUTPUT
  sig_noise_bit
  sig_noise_bit_DC


 %These are unscaled IFFT output values and ADC sampled values
 %They are used to find the MSE waveform
 wave_in_q ;
 wave_in_inphase;
 wave_theory_q     = ifft_outI;
 wave_theory_inphase = ifft_outR;
  %Waveform MSE script
  wave_mse;
  %OUTPUT
  meanWaveErr
```

SCRIPT: lfsr_17.m

Purpose: This the scrip for for 64QAM only; used to generate the PN data

```
numSym = 200; %Must manually set to match main
n=17;  %LFSR ORDER
m=16; %16 bits per clock cycles
run = numSym*64*6; %Total Number of clock cycles = num bits
bData = zeros(1,(numSym*64*6)); %Binary data generated
rData = zeros(64,numSym); %Decimal values of the 6-bit groups
reg = zeros(1,n); %Shift register
reg(n-1) = 1; %Preload shift register

%Generate the binary data in this loop
for i=1:run
  %This is the 17th order polynomial equation
  reg = [ reg(2:(n-3)) xor(reg(1),reg(n-2)) reg(n-1) reg(n) reg(1)];

 reg(1);
 bData(1,i)=reg(1);
end

cnt=1;%External loop counter

%Group the generated binary data into 6-bit groups and convert to decimal value
for ka=1:6:(64*numSym*6)
    dData(cnt,1)=bin2dec( [num2str(bData(ka))  num2str(bData(ka+1))
        num2str(bData(ka+2))     num2str(bData(ka+3)) num2str(bData(ka+4))
        num2str(bData(ka+5)) ] );
  cnt=cnt+1;
```

```
  ka;
end

%Convert matrix format of the data
for kb=1:numSym
  for kc=1:64
    rData(kc,kb) = dData( ((kb-1)*64)+kc,1);
  end
end

%need to add one for values to range from 1-64, instead of 0-63
%This simplifies the table look up later in Matlab
rData = rData + 1;
```

SCRIPT: loadTable.m
Purpose: Loads the table that stores the 64QAM constellation

```
%these variable are used to help define coordinates on constellation
a = 4096;
b = 12288;
c = 20480;
d = 28672;

%Here are the table values for 64QAM with Greycode mapping
tbl = [
complex(-d,d);
complex(-c,d);
complex(-a,d);
complex(-b,d);
complex( b,d);
complex( c,d);
complex( a,d);
complex( d,d);

complex(-d,c);
complex(-c,c);
complex(-a,c);
complex(-b,c);
complex( b,c);
complex( c,c);
complex( a,c);
complex( d,c);

complex(-d,a);
complex(-c,a);
complex(-a,a);
complex(-b,a);
complex( b,a);
complex( c,a);
complex( a,a);
complex( d,a);

complex(-d,b);
complex(-c,b);
```

```
complex(-a,b);
complex(-b,b);
complex( b,b);
complex( c,b);
complex( a,b);
complex( d,b);

complex(-d,-b);
complex(-c,-b);
complex(-a,-b);
complex(-b,-b);
complex( b,-b);
complex( c,-b);
complex( a,-b);
complex( d,-b);

complex(-d,-c);
complex(-c,-c);
complex(-a,-c);
complex(-b,-c);
complex( b,-c);
complex( c,-c);
complex( a,-c);
complex( d,-c);

complex(-d,-a);
complex(-c,-a);
complex(-a,-a);
complex(-b,-a);
complex( b,-a);
complex( c,-a);
complex( a,-a);
complex( d,-a);

complex(-d,-d);
complex(-c,-d);
complex(-a,-d);
complex(-b,-d);
complex( b,-d);
complex( c,-d);
complex( a,-d);
complex( d,-d);];
```

SCRIPT: filter_lpf2.m
Purpose: Low pass filtering

```
%Inputs to filter
  %-new_dataQ; .serial data
  %-new_dataI;
%Outputs
  %-delay .integer
  %-yq .serial data
  %-yi
```

```
%Use remezord to find filter order and coefficents
 [n1,fo1,mo1,w1] = remezord( [0.35 2.4] , [1 0], [0.01 0.01], Fa );
bb = remez(n1,fo1,mo1,w1);
ord = n1;
[h,w] = freqz(bb,1);
%Plot frequency response of the filter
figure,stem(w/pi,20*LOG10(abs(h)));

%Fillter the I/Q components from the DA
yq = filter(bb,1,new_dataQ);
yi = filter(bb,1,new_dataI);

%filter delay factor used later to find first OFDM sample
delay=ceil(ord/2);%add one if filter is even
```

SCRIPT: Quatization_script_LPF.m
Purpose: Send data from LPF to ADC for quantization

```
%Required inputs
 %-serial_ifft_outR
 %-serial_ifft_outR
%Outputs
 %-resampled_dataI
 %-resampled_dataQ

  input_inphase = resampled_dataI2;%this is in serial format
  input_q = resampled_dataQ2;
  quantization2;  %Quantization script
  resampled_dataI =inphase3;
  resampled_dataQ  = q3;
```

SCRIPT: ber_fft.m
Purpose: Demodulates the ADC data samples to QAM symbols

```
%Inputs
 %-numSym
 %-in_q
 %-in_i
    numSym = nSym;

real_unscale4= in_q;
imag_unscale4= in_i;

final_demod = zeros(64,numSym);

for k=1:numSym

cmplx = complex(inphase2((1+(64*(k-1))):(64*k)),q2( (1+(64*(k-1)) ):(64*k) ) );
%Take the FFT
fft_data = fft(cmplx,64);

const_points(1:64,k) = (fft_data);
demod_real = zeros(length(fft_data),1);
demod_imag = zeros(length(fft_data),1);
```

```
%De-map the FFT outputs
for i=1:(length(fft_data))

   var = real(fft_data(i));

      if(var < x(1))
        demod_real(i) = -d;

   elseif(var > x(1)) && (var < x(2))
        demod_real(i) = -c;

   elseif(var > x(2)) && (var < x(3))
        demod_real(i) = -b;

   elseif(var > x(3)) && (var < x(4))
        demod_real(i) = -a;

   elseif(var > x(4)) && (var < x(5))
        demod_real(i) =  a;

   elseif(var > x(5)) && (var < x(6))
        demod_real(i) =  b;

   elseif(var > x(6)) && (var < x(7))
        demod_real(i) =  c;

   else
        demod_real(i) =  d;

   end
end




for i=1:(length(fft_data))

   var = imag(fft_data(i));

   if(var < x(1))
        demod_imag(i) = -d;

   elseif(var > x(1)) && (var < x(2))
        demod_imag(i) = -c;

   elseif(var > x(2)) && (var < x(3))
        demod_imag(i) = -b;

   elseif(var > x(3)) && (var < x(4))
        demod_imag(i) = -a;

   elseif(var > x(4)) && (var < x(5))
```

```
      demod_imag(i) = a;

  elseif(var > x(5)) && (var < x(6))
      demod_imag(i) = b;

  elseif(var > x(6)) && (var < x(7))
      demod_imag(i) = c;

  else
      demod_imag(i) = d;

  end
end

vars = complex(demod_real,demod_imag);

for i=1:(length(fft_data))

  for j=1:64
    if(vars(i)==tbl(j))
      final_demod(i,k) = j-1;
      break;
    end
  end
end

final_demod; %Output: This variable holds all the demodulated QAM symbols

end %end of big for loop
```

SCRIPT: plot_const_serial.m
Purpose: Takes the FFT of the received ADC samples and plots them on constellation

```
figure
%Inputs
 % -yy_q
 %-yy_inphase
 %  numSym;

q2 = yy_q;
inphase2 = yy_inphase;


%Take the FFT
cmplx = complex(inphase2(1:64),q2(1:64));
fft_data1 = fft(cmplx,64);
fft_data  = (fft_data1); %extra line
fft_data3(1:64,1) = fft_data;

%Plot the first OFDM symbol
plot(real(fft_data), imag(fft_data), 'rx');

%Add grid and dimensions to the figure
grid on
```

```
a = 4096;
ax = 9*a;
axis([-ax ax -ax ax]);

hold on
y2 =[-7*a,-5*a,-3*a,-a,a,3*a,5*a,7*a];

for i=1:8
 x2 =[y2(i) y2(i) y2(i) y2(i) y2(i) y2(i) y2(i) y2(i)];
 plot(x2,y2,'go')
end


%Plot all sequential OFDM symbols
for k=2:numSym
cmplx = complex(inphase2((1+(64*(k-1))):(64*k)),q2( (1+(64*(k-1)) ):(64*k) ) );

fft_data1 = fft(cmplx,64);
fft_data        =(fft_data1); %extra line
fft_data3(1:64,k)  =fft_data;

fft_data_actual = fft_data;
plot(real(fft_data_actual), imag(fft_data_actual), 'r+')

end
```

SCRIPT: plot_const_serial.m

Purpose: Takes the FFT of the received ADC samples and plots them on constellation

```
%Inputs:
 %   *final_demod_real
 %   *final_demod_ideal

numSym;

  compare = zeros(64,numSym);
  for g=1:numSym
   for n=1:64
    if final_demod_real(n,g) == final_demod_ideal(n,g)
      compare(n,g) = 0;
    else
      compare(n,g) = 1;
    end
   end
  end
  compare;
  sum(compare);
  out_err = sum(sum(compare)) %Output: This should be zero for no errors
```

SCRIPT: find_mse_noDC.m

Purpose: Find the mean squared error with the demodulated QAM symbols on the constellation, excluding first FFT output bins. Script is similar to 'find_mse.m', which includes all FFT bins.

```
%Inputs
points_in;
points_theory;

%Remove the first FFT output bin for 'find_mse_noDC.m'
%Do not do this if you want to find mse with all FFT output bins
for sa=1:numSym
   points_in2(1:63,sa) = points_in(2:64,sa);
   points_theory2(1:63,sa) = points_theory(2:64,sa);
end

total1 = abs(points_in2 - points_theory2);
total2 =  sum(sum(total1.^2));

totalc = ((4096*2)^2)*(numSym*63);

meanErr = (total2./totalc).*100;
meanErr; %Output
```

SCRIPT: find_sir.m
Purpose: To find the SIR, SIR-bit

```
%Inputs
 %-points_in.
 %-points_theory
 %-numSym
  M=64;
  numBits = 6; %number of bits per symbol

%Copy the inputs, but dropping the first FFT output bin
for w = 1:numSym
   points_theory2(1:63,w) = points_theory(2:64,w);
   points_in2(1:63,w)     = points_in(2:64,w);
end

%This is the total number of QAM symbols
pts = (numSym*63); %excludes first FFT output
pts3 = (numSym*64);

%This is the actual number of points transmitted
power_avg_bit_noDC = (sum(sum((abs(points_theory2)).^2))) /(pts*numBits);
power_avg_sym      = (sum(sum((abs(points_theory2)).^2)));
power_avg_sym_DC   = (sum(sum((abs(points_theory )).^2)));
power_avg_bit_DC   = (sum(sum((abs(points_theory )).^2))) /(pts3*numBits)

 points_diff_noDC = (abs(points_in2 - points_theory2)).^2;
 points_diff_DC   = (abs(points_in - points_theory)).^2;
 noise_avg        = (sum(sum(points_diff_noDC)));
 noise_avg_DC     = (sum(sum(points_diff_DC)));
```

```
sig_noise_sym_DC = 10*LOG10(power_avg_sym_DC/noise_avg_DC); %SIR
sig_noise_bit = 10*LOG10(power_avg_bit_noDC/noise_avg); %SIR-bit without first FFT bin
sig_noise_bit_DC= 10*LOG10(power_avg_bit_DC/noise_avg_DC); %SIR-bit
```

SCRIPT: wave_mse.m

Purpose: To find the mean squared of the sampled OFDM symbol time waveform

```
%find MSE between two waveforms
 %Inputs
wave_in_q;
wave_in_inphase;
wave_theory_q;
wave_theory_inphase;

cmplx_wave_actual = complex(wave_in_inphase,wave_in_q);
cmplx_wave_theory = complex(wave_theory_inphase,wave_theory_q);

total21 = abs(cmplx_wave_actual - cmplx_wave_theory);
total22 =  sum(sum(total21.^2)); %will sum each column

totalaa = abs(cmplx_wave_theory);
totalbb = sum(sum(totalaa.^2));

meanWaveErr = (total22./totalbb).*100;
meanWaveErr; %Output
```

References

[1] Ackenhusen, John G.: "**Real Time Signal Processing**" New Jersey: Prentice Hall 1999.

[2] Chassing, Rulph: "**DSP Applications Using C and the TMS320C6x DSK.**" New York: John Wiley & Sons, Inc. 2002.

[3] Debaillie, B., et al.: **"Impact of front-end filters on bit error rate performances in WLAN-OFDM transceivers"** IEEE Radio and Wireless Conference (RAWCON 2001), August 2001, pp. 193 – 196

[4] Eberle, W., et al.: **"OFDM-WLAN receiver performance improvement using digital compensation techniques" IEEE** Radio and Wireless Conference, Aug. 2002, pp. 111 – 114.

[5] Fort, A.; Eberle, W.: **"Synchronization and AGC Proposal for IEEE 802.11a Burst OFDM Systems"** IEEE GLOBECOM, vol. 3, Dec.2003 pp. 1335 – 1338.

[6] Haykin, Simon S.: "**Communication Systems.**" New York: Wiley, 2001.

[7] Hazy, Laszlo: Masters Thesis: **"Initial Channel Estimation and Frame Synchronization in OFDM Systems for Frequency Selective Channels."** Ottawa: Carleton University. 1996.

[8] Irons, F. H., et al.: **"The Noise Power Ratio – Theory and ADC Testing."** IEEE Transactions on Instrumentation and Measurements, Volume 49, Issue 3, June 2000.

[9] Knight, John: **" 97.478 ASIC Design Course Notes."** Ottawa: Carleton University. January 2001.

[10] Liu, Chia-Liang: **"Impacts of I/Q imbalance on QPSK-OFDM-QAM detection"** IEEE Transactions on Consumer Electronics , Volume: 44 , Issue: 3 , Aug. 1998 Pages:984 – 989.

[11] Moschitta, A.; Petri, D.: **"Wideband communication system sensitivity to overloading quantization noise [ADC characterization]"** IEEE Transactions on Instrumentation and Measurement, vol. 52, August 2003, p.1302 – 1307.

[12] Nee R. V.; Prasad R.: **"OFDM for wireless multimedia communications."** Boston: Artech House, 2000.

[13] Popov, O.V.: **"Dynamic DC offset impact on the 802.11a receiver performance"** 1st IEEE International Conference on Circuits and Systems for Communications (ICCSC), June 2002, p. 250 – 253

[14] Proakis, J.G.: **"Digital Communications",** McGraw-Hill 2001.

[15] Proakis, J.G.: **"Digital signal processing : principles, algorithms, and applications"**, McGraw-Hill 1996.

[16] Rudberg, M.K.: **"ADC offset identification and correction in DMT modems."** IEEE International Symposium on Circuits and Systems (ISCAS), vol. 4, May 2000. p. 677 – 680.

[17] Schuchert, A.; et. al.: **"A novel IQ imbalance compensation scheme for the reception of OFDM signals"** IEEE Transactions on Consumer Electronics, vol. 47, Issue: 3 , Aug. 2001, p. 313 – 318

[18] Shousheng H.; Torkelson, M.: **"Effective SNR estimation in OFDM system simulation"** Globecom 98. vol. 2, November 1998, p. 945 – 950.

[19] Texas Insturments. "**Application Report TMS320C6000 Peripherals Reference Guide.**" Literature Number: SPRU190D. February 2001.

[20] Texas Instruments: "**Application Report Understanding Data Converters.**" Texas Instruments. Literature Number: SLAA013, 1995

[21] Texas Instruments. "**How to Begin Development Today with the TMS320C6711.**" Literature Number: SPRA522. March 1999.

[22] Texas Instruments. "**THS5661A 12-BIT, 125 MSPS, CommsDAC DIGITAL-TO-ANALOG CONVERTER.**" Literature Number: SLAS247B. September 2002.

[23] Texas Instruments. "**THS56X1EVM for the THS5641A/51A/61A/71A 8-,10-,12-, and 14-Bit CommsDAC Digital-to-Analog Converters.**" Literature Number: SLAU032B. February 2001.

[24] Texas Instruments. "**TMS320C6000 Chip Support Library API User's Guide.**" Literature Number: SPRU401D. April 2002.

[25] Texas Instruments. "**TMS320C6000 CPU and Instruction Set Reference Guide.**" Literature Number: SPRU189F. October 2000.

[26] Texas Instruments. "**TMS320C6000 DMA Example Applications.**" Literature Number: SPRA529A. April 2002.

[27] Texas Insturments. "**TMS320C6000 McBSP: Data Packing.**" Literature Number: SPRA551. May 1999.

[28] Texas Insturments. "**TMS320C6000 McBSP: Initialization**" Literature Number: SPRA488. November 1998.

[29] Texas Insturments. "**TMS320C6000 McBSP Initialization.**" Literature Number: SPRA488B. April 2002.

[30] Texas Instruments. "**TMS320C6000 Programmer's Guide.**" Literature Number: SPRU198G. August 2002.

[31] Texas Instruments. "**TMS320C6711, TMS320C6711B, TMS320C6711C, TMS320C6711D FLOATING-POINT DIGITAL SIGNAL PROCESSORS.**" Literature Number: SPRS088L. May 2004.

[32] Texas Instruments. "**TMS320C67x DSP Library programmer's Reference Guide.**" Literature Number: SPRU657. February 2003.

[33] Texas Instruments. "**TMS320C6x Peripheral Support Library Programmer's.**" Literature Number: SPRU273B. July 1998.

[34] Texas Instruments. "**TMS320 Cross-Platform Daughtercard Specification Revision 1.0.**" SPRA711. November 2000.

[35] Texas Insturments. "**Using the TMS320C6000 McBSP as a High Speed Communication Port.**" Literature Number: SPRA455A. August 2001.

[36] Tubbax, J.; et al.: "**Compensation of IQ imbalance in OFDM systems**" IEEE International Conference on Communications (ICC), vol. 5, 2003

[37] Wang J.; Yang C.: **"The influence of analog device on OFDM system"** International Conference on Communication Technology Proceedings (ICCT), vol. 2, April 2003 p. 1060 – 1062.

[38] Witrisal K.: **"Impact of DC-offsets and Carrier Feed-through on Correlation-based Frequency Synchronization for OFDM,"** Proc. 6th international OFDM-Workshop, Hamburg (Germany), Sept. 2001, pp. 15-1–15-5.

[39] Wright, A.R and Naylor, P.A.: **"I/Q mismatch compensation in zero-IF OFDM receivers with application to DAB"**. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '03), vol. 2, April 2003. p. II - 329-32.

[40] Zhou Y; et al: **"An effective scheme to improve performance of OFDM-based system"** International Conference on Communication Technology Proceedings (ICCT), Volume: 2, April 2003, p. 1190 – 1193.