

國立交通大學

電信工程學系碩士班 碩士論文

具最佳硬體資源利用之 MIMO-OFDM 系統 之 FPGA 之實現

FPGA Realization of a MIMO-OFDM System with Optimized Hardware Resource Utilization

研究生：陳彥宇

Student: Yen-Yu Chen

指導教授：李大嵩 博士

Advisor: Dr. Ta-Sung Lee

中華民國九十五年六月

具最佳硬體資源利用之 MIMO-OFDM 系統
之 FPGA 之實現

FPGA Realization of a MIMO-OFDM System with
Optimized Hardware Resource Utilization

研 究 生：陳彥宇

Student: Yen-Yu Chen

指導教授：李大嵩 博士

Advisor: Dr. Ta-Sung Lee

國立交通大學

電信工程學系碩士班

碩士論文

A Thesis

Submitted to Department of Communication Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Communication Engineering

June 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年六月

具最佳硬體資源利用之 MIMO-OFDM 系統 之 FPGA 之實現

學生：陳彥宇

指導教授：李大嵩 博士

國立交通大學電信工程學系碩士班

摘要

正交分頻多工(OFDM)技術在新一代無線通訊系統佔有相當關鍵性的地位，它可提供高速數據傳輸，且適合操作在多重路徑所引起之頻率選擇性通道下；另一方面，多輸入多輸出(MIMO)技術可提升傳輸率及鏈路品質。因此，在新一代通訊系統中，MIMO-OFDM 將成為極具有潛力之關鍵技術之一。在本論文中，吾人將使用快速雛形發展平台 Aptix[®] MP3C，以及自行研發之平台，實現一 2×2 MIMO-OFDM 系統，其中基頻演算法部分將實現於平台之 FPGA 模組。在此系統中，吾人採用了兩種不同之時空演算法，分別為 STBC 及 VBLAST。其餘演算法包括通道估計器，相位追蹤器，迴旋碼解碼器等，也將完整的實現於系統中。此外，吾人更進一步提出一套有系統的量化演算法，能在浮點數轉定點數時有效的壓抑量化誤差(quantization error)，並且同時最佳化所需之硬體資源利用。

FPGA Realization of a MIMO-OFDM System with Optimized Hardware Resource Utilization


Student: Yen-Yu Chen

Advisor: Dr. Ta-Sung Lee

Department of Communication Engineering

National Chiao Tung University

Abstract



In recent years, orthogonal frequency division multiplexing (OFDM) becomes a key technology in the development of new wireless communication systems, enabling high data rate transmission, and is suitable for frequency selective channels caused by multipath propagation. On the other hand, multiple-input multiple-output (MIMO) technique has a great potential of delivering either a dramatic increase of throughput or improvement of link quality. Combined with the MIMO technique, OFDM systems become more suited to next generation wireless communications. In this thesis, we propose a total solution for building up a 2×2 MIMO-OFDM system on two FPGA-based platforms: a fast prototyping platform Aptix[®] MP3CF and a self-designed platform. There are two space-time algorithms adopted in our system, including Space-Time Block Coding (STBC) and Vertical Bell Labs Layered Space-Time (VBLAST). Furthermore, since fixed-point computation is adopted in our system due to the cost and complexity of floating-point hardware, we also propose a quantization algorithm which can not only minimize the hardware resource requirement but also constrain the quantization error within a specified limit when converting floating-point arithmetic to fixed-point arithmetic.

Acknowledgement

First, I am very grateful to my advisor, Dr. Ta-Sung Lee, for his enthusiastic guidance and great patience, especially the training of oral presentation and being earnest in our works. Then I would also thanks to Chung-Ta Ku, Po-Tien Lee, and Dr. Juinn-Horng Deng who spend lots of time for my consultation. Special thanks to Jeff Tsai for the technical support on the circuit design of self-designed platform. Heartfelt thanks are also offered to all members in the Communication Signal Processing and System Design (CSPSD) Lab for their constant encouragement and help.

Finally, I would like to express my deepest gratitude to my family for their endless love, especially my mom for her tender encouragement, and my dad as a constant reminder of health.



Contents

Chinese Abstract	I
English Abstract	II
Acknowledgement	III
Contents	IV
List of Figures	VIII
List of Tables	XI
Acronym Glossary	XII
1 Introduction	1
2 MIMO-OFDM Baseband Transceiver Architecture	4
2.1 Overview of MIMO-OFDM System	4
2.2 Transmitter Architecture	6
2.2.1 Convolutional Encoder	7
2.2.2 Interleaver / De-interleaver	8
2.2.3 Mapper / De-mapper	9
2.2.4 Preamble Channel and Frame Structure	9
2.2.5 Root Raised Cosine Filter	10
2.3 Receiver Architecture	11
2.3.1 Timing Synchronizer	12



2.3.2 Frequency Synchronizer	12
2.3.3 Channel Estimator.....	13
2.3.4 Phase Estimator.....	14
2.3.5 Viterbi Decoder.....	15
2.4 MIMO Techniques	18
2.4.1 Spatial Diversity Technique.....	18
2.4.2 Spatial Multiplexing Technique.....	20
2.5 Summary.....	22
3 MIMO-OFDM System Platforms	23
3.1 Fast Prototyping Platform.....	23
3.1.1 Aptix® System Explorer.....	24
3.1.2 FPGA Module	28
3.1.2.1 FPGA Overview.....	28
3.1.2.2 FPGA Design Flow	29
3.1.3 ‘C6701 DSP EVM	31
3.1.3.1 TMS320C6701 DSP Overview	32
3.1.3.2 DSP Design Flow	34
3.1.4 USB 2.0 Module	35
3.1.5 AD and DA Modules.....	36
3.1.6 Debugging Tools.....	37
3.2 Self-designed Platform	38
3.2.1 RF Module	39
3.2.2 AD and DA Modules.....	40
3.2.3 MAC/BB Platform.....	42
3.2.4 USB Interface	42

3.2.5 Debugging Tools	43
3.3 Summary	45
4 MIMO-OFDM System Realization	46
4.1 Design Flow	46
4.2 MATLAB Verification	47
4.2.1 Floating-Point Verification	48
4.2.2 Fixed-Point Verification	51
4.3 FPGA Realization	53
4.3.1 Design Principles	53
4.3.2 Circuit Design	55
4.3.2.1 Circuit Design of Transmitter	55
4.3.2.2 Circuit Design of Receiver	62
4.4 ModelSim simulation	71
4.5 Experimental Results	72
4.5.1 Fast Prototyping Platform	73
4.5.2 Self-designed Platform	74
4.6 Summary	76
5 Proposed Quantization Algorithm with Minimum Hardware Requirement	77
5.1 Introduction of Quantization	78
5.2 Previous work	80
5.3 Proposed Quantization Algorithm	81
5.3.1 Pre-quantization Works	81
5.3.2 Determine Hardware Resource Weightings	85
5.3.3 Determine Integer Lengths	87

5.3.4 Determine Fraction Lengths	88
5.3.4.1 Coarse Modification.....	89
5.3.4.2 Fine Modification.....	90
5.4 Simulation Results	91
5.5 Summary	95
6 Conclusion	96
Bibliography	98



List of Figures

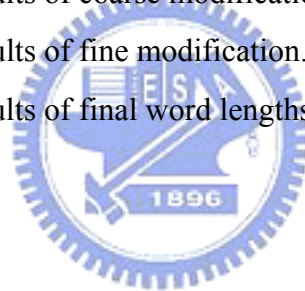
Figure 2.1: (a) Conventional multicarrier technique (b) Orthogonal multicarrier modulation technique.....	6
Figure 2.2: Transmitter architecture of MIMO-OFDM system.....	7
Figure 2.3: Convolutional encoder with code rate 1/3 and constraint length 5.....	7
Figure 2.4: Interleaver and de-interleaver schemes.....	8
Figure 2.5: QPSK, 16-QAM, and 64-QAM constellations.....	9
Figure 2.6: Training sequence and frame structure of IEEE 802.11a standard.....	10
Figure 2.7: Receiver architecture of MIMO-OFDM system.....	11
Figure 2.8: Trellis diagram part 1.....	17
Figure 2.9: Trellis diagram part 2.....	17
Figure 3.1: Development environment of fast prototyping system.....	24
Figure 3.2: Modules installed on Aptix MP3CF platform.....	24
Figure 3.3: Aptix [®] MP3CF platform.....	25
Figure 3.4: Explorer flow.....	26
Figure 3.5: FPGA design flow.....	30
Figure 3.6: 'C6701 DSP EVM.....	32
Figure 3.7: Architecture of 'C6701 DSP EVM.....	33
Figure 3.8: USB 2.0 module.....	35
Figure 3.9: USB 2.0 module and its neighborhood.....	35
Figure 3.10: AD and DA modules on fast prototyping platform.....	36
Figure 3.11: Development environment of self-designed platform.....	38
Figure 3.12: Main board of self-designed platform.....	39
Figure 3.13: RF module on self-designed platform.....	40
Figure 3.14: Measured carrier spectrum form RF module.....	40
Figure 3.15: AD/DA module on self-designed platform.....	41

Figure 3.16: Measured data waveform from AD/DA module	42
Figure 3.17: MAC/BB platform.....	42
Figure 3.18: USB module on self-designed platform.....	43
Figure 3.19: Spectrum analyzer block diagram	44
Figure 3.20: Vector signal analyzer block diagram	44
Figure 4.1: FPGA design flow	47
Figure 4.2: Impulse and frequency response of RRC filter with $\beta=0.22$	48
Figure 4.3: (a) Original waveform (b) RRC shaped waveform on transmitter (c) RRC shaped waveform on receiver	49
Figure 4.4: Eye diagram of RRC shaped waveform.....	49
Figure 4.5: Coarse timing synchronization output.....	50
Figure 4.6: Real and estimated channel frequency response	50
Figure 4.7: Floating-point system performance.....	51
Figure 4.8: Fixed-point system performance.....	52
Figure 4.9: Circuit design of transmitter.....	55
Figure 4.10: Circuit design of convolutional encoder	56
Figure 4.11: Circuit design of interleaver	57
Figure 4.12: Circuit design of mapper	58
Figure 4.13: Circuit design of STBC encoder with pilot zero tones adder.....	58
Figure 4.14: Circuit design of de-multiplexer with pilot zero tones adder.....	59
Figure 4.15: Circuit design of fast Fourier transform.....	60
Figure 4.16: Circuit design of oversampler and CP adder.....	60
Figure 4.17: Circuit design of first and second preamble stream generators	61
Figure 4.18: Circuit design of RRC filter	61
Figure 4.19: Circuit design of receiver side.....	62
Figure 4.20: Circuit design of timing synchronizer	63
Figure 4.21: Circuit design of oversample and CP remover.....	63
Figure 4.22: Circuit design of pilot zero tone mover.....	64
Figure 4.23: Circuit design of channel estimator.....	65
Figure 4.24: Circuit design of phase estimator	65

Figure 4.25: Circuit design of STBC decoder	66
Figure 4.26: Original real number calculating strategy in VBLAST.....	67
Figure 4.27: Modified real number calculating strategy in VBLAST	68
Figure 4.28: Circuit design of VBLAST detector.....	68
Figure 4.29: Circuit design of Viterbi decoder	69
Figure 4.30: Circuit design of branch metric generator.....	69
Figure 4.31: Circuit design of add, compare, and select block.....	70
Figure 4.32: STBC ModelSim simulation result	71
Figure 4.33: VBLAST ModelSim simulation result.....	71
Figure 4.34: Transmitted waveform of MIMO-OFDM system.....	72
Figure 4.35: Prototyping platform experimental result	73
Figure 4.36: Self-designed platform development environment	74
Figure 4.37: Self-designed platform experimental result: received spectrum and waveforms on PSA and VSA.....	75
Figure 4.38: Self-designed platform experimental result: timing synchronization waveform on LA.....	75
Figure 4.39: Self-designed platform experimental result: source data and detected data waveform on LA.....	76
Figure 5.1: Quantization example 1: truncation and rounding.....	78
Figure 5.2: Quantization example 2: saturation and wrapping.....	79
Figure 5.3: Data flow paths and distribution of quantization-related blocks and quantization-irrelevant blocks in MIMO-OFDM system.....	83
Figure 5.4: Influenced circuit blocks by q_{ifft} and q_{RRC}	86
Figure 5.5: Influenced circuit blocks by q_{RRC} , q_{rxRRC} , q_{fft} , q_{lpmb} , q_{ch} and q_{ph}	87
Figure 5.6: Detected constellation under floating-point case	91
Figure 5.7: Detected constellation under $EM_{\text{target}} = 1$	92
Figure 5.8: Detected constellation under $EM_{\text{target}} = 5$	92
Figure 5.9: Detected constellation under $EM_{\text{target}} = 10$	92
Figure 5.10: System performance under different EMs as q_{ifft} uses saturation.....	93
Figure 5.11: System performance under different EMs as q_{ifft} uses wrapping.....	94

List of Tables

Table 2.1: Specification of MIMO-OFDM baseband transceiver	6
Table 2.2: State transition table.....	16
Table 4.1: Word lengths under different EMs.....	52
Table 4.2: Synthesis and P&R information.....	72
Table 5.1: Quantizers and their settings in MIMO-OFDM system	85
Table 5.2: Experimental results of hardware resource weightings	87
Table 5.3: Experimental results of integer lengths.....	88
Table 5.4: Experimental results of coarse modification.....	90
Table 5.5: Experimental results of fine modification.....	91
Table 5.6: Experimental results of final word lengths and EMs	95



Acronym Glossary

4G	the fourth generation
ACS	add, compare, and select
AD	analog to digital converter
AMPS	advanced mobile phone services
ASIC	application specific integrated circuit
AWGN	additive white Gaussian noise
BER	bit error rate
BMG	branch metric generator
BPF	bandpass filter
BPSK	binary phase shift keying
CCS	code composer studio
CDMA	code division multiple access
CP	cyclic prefix
CPLD	complex programmable logic device
CPU	central processing unit
DA	digital to analog converter
DAB	digital audio brocasting
D-AMPS	digital AMPS
DBLAST	diagonal Bell laboratory layered space-time
DFT	discrete Fourier transform
DSP	digital signal processor
DVB-T	terrestrial digital video broadcasting
EDA	electronic desing automation
EDIF	electronic design interface format
EM	error metric

EVM	evaluation module
FDA	filter design and analysis
FFT	fast Fourier transform
FIFO	first in, first out
FIR	finite impulse response
FPCB	field programmable circuit board
FPGA	field programmable gate array
FPIC	field programmable interconnect component
FSM	finite state machine
GSM	global system for mobile communications
HDL	hardware description language
I/O	input/output
ICI	inter-carrier interference
IDE	integrated development environment
IEEE	institute of electrical and electronics engineers
IF	intermediate frequency
ISI	inter-symbol interference
JTAG	joint test action group
LA	logic analyzer
LUT	look up table
MF	match filter
MFLOPS	mega floating-point operations per second
MIMO	multiple-input multiple-output
MMSE	minimum mean square error
OBW	occupied bandwidth
OFDM	orthogonal frequency division multiplexing
OTP	one time programmable
PC	personal computer
PIC	parallel interference cancellation
PLD	programmable logic device
PLL	phase-locked loop

QAM	quadrature amplitude modulation
QoS	quality of service
QPSK	quaternary phase shift keying
RAM	random access memory
RBW	resolution bandwidth
ROM	read-only memory
RRC	root raised cosine
RTL	register transfer level
SBSRAM	synchronous burst SRAM
SIC	successive interference cancellation
SNR	signal to noise ratio
SoC	system on a chip
SRAM	static random access memory
STBC	space time block code
TBU	trace back unit
UART	universal asynchronous receiver/transmitter
USB	universal serial bus
VBLAST	vertical Bell laboratory layered space-time
VCO	voltage-controlled oscillator
VHDL	very high speed integrated circuit hardware description language
VLIW	very long instruction word
WCDMA	wideband CDMA
WLAN	wireless local area network
ZF	zero forcing

Chapter 1

Introduction

Communication technologies have been developed rapidly in recent decades of years. The first-generation (1G) radio systems transmit voice over radio by using analog communication techniques, such as Advanced Mobile Phone Services (AMPS), which were developed in the 1970s and 1980s. The 2G systems were built in the 1980s and 1990s, and featured the adoption of digital technology, such as Global System for Mobile Communications (GSM), Digital-AMPS (D-AMPS), and code division multiple access (CDMA); among them GSM is the most successful and widely used 2G system. 3G mobile technologies provide users with high-data-rate mobile access, which developed rapidly in the 1990s and is still developing today. The major radio air interface standard for 3G is wideband CDMA (WCDMA), whose transmission data rate can be up to 2 Mbps in good conditions. However, there are some limitations with 3G, such as the difficulty in extending to very high data rates due to excessive interference between services, and the difficulty in providing multi-rate services with different quality of service (QoS) due to the restrictions imposed on the core network by the air interface standard. Therefore, the future mobile communication system having the features of high-data-rate transmission and open network architecture, called 4G, is desired to meet the increasing demand for broadband wireless access. In fact, the combination of multiple-input multiple-output (MIMO) signal processing with orthogonal frequency division multiplexing (OFDM) has been regarded as a promising solution for enhancing the data rates of next-generation wireless communication systems [1].

OFDM has become a popular technique for transmission of signals over wireless channels, and its most well known advantage is the capability of converting a frequency-selective channel into a parallel collection of frequency flat sub-channels, which makes the receiver simpler. Therefore, OFDM has been adopted in several wireless standards such as digital audio broadcasting (DAB), terrestrial digital video broadcasting (DVB-T), the IEEE 802.11a/g wireless local area network (WLAN) standard, and the IEEE 802.16-2005 standard. These show its potential of being a candidate for future-generation (4G) mobile wireless systems.

MIMO techniques are also popular recently; it can basically be categorized into two groups. The first one aims to improve the power efficiency and transmission reliability by maximizing spatial diversity; one popular example is the space-time block codes (STBC) [2]. The second type uses a layered approach to increase capacity; one popular example of such a system is the vertical-Bell Laboratories layered space-time (VBLAST) architecture [3] [4], in where independent data signals are transmitted over antennas to increase the data rate.

The goal of this thesis is to realize a 2x2 MIMO-OFDM system on FPGA-based platforms, where we intend to verify the above-mentioned space-time algorithms on both fast prototyping platform and self-designed platform. The complete functional blocks in both the transmitter and receiver are provided, and the associated algorithms applied in each functional block are also presented. After giving an overview of system architecture, we propose a total solution to build up FPGA-based platforms for realizing the MIMO-OFDM system, including MATLAB verification, and FPGA realization. The developed system contains a baseband transmitter, a digital-analog converter, an analog-digital converter, and a baseband receiver.

Furthermore, owing to the cost and complexity of floating-point hardware, the proposed MIMO-OFDM system on FPGA is limited to fixed-point arithmetic. Therefore the floating-point to fixed-point conversion becomes an inevitable procedure. To determine word lengths of all input, intermediate, and output signals, we propose a quantization algorithm which can minimize the hardware resources while constraining the quantization error within a specific limit. Moreover, the concept of hardware resource weighting is introduced, and some communication characteristics are also

taken into account.

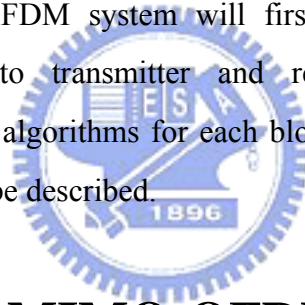
The organization of this thesis is as follows. Chapter 2 describes the proposed MIMO-OFDM transceiver architecture and its corresponding schemes. In Chapter 3, the development environments of the proposed fast prototyping platform and self-designed platform are introduced. In Chapter 4, the overall system realization is presented, and the performance evaluation is also included. Later, a systematical quantization algorithm is provided in Chapter 5. Finally, we make our concluding remarks in Chapter 6.



Chapter 2

MIMO-OFDM Baseband Transceiver Architecture

This chapter focuses on the MIMO-OFDM baseband transceiver architecture. An overview of the MIMO-OFDM system will first be given. Then we divide the developed architecture into transmitter and receiver, and provide functional descriptions and associated algorithms for each block. Finally, the MIMO techniques adopted on the system will be described.



2.1 Overview of MIMO-OFDM System

OFDM has long been regarded as an efficient approach to combat the adverse effects of multipath spread, and is the main solution to many wireless systems. It converts a frequency-selective channel into a parallel collection of frequency flat subchannels, which makes the receiver simpler. The time domain waveforms of the subcarriers are orthogonal, yet the signal spectrum corresponding to the different subcarriers overlap in frequency domain. Therefore, the available bandwidth is used very efficiently, especially compared with those systems having intercarrier guard bands, as shown in Figure 2.1 [5]. In order to eliminate inter-symbol interference (ISI) almost completely, a guard time is introduced for each OFDM symbol. Moreover, to eliminate inter-carrier interference (ICI), the OFDM symbol is further cyclically extended in the guard time, resulting in the cyclic prefix (CP). Otherwise, multipath

remains an advantage for an OFDM system since the frequency selectivity caused by multipaths can improve the rank distribution of the channel matrices across those subcarriers, thereby increasing system capacity. We summarize the advantages of OFDM as follows [1]:

- High spectral efficiency
- Simple implementation by FFT
- Robustness against narrowband interference
- High flexibility in terms of link adaptation for having many subcarriers
- Suitability for high-data-rate transmission over a multipath fading channel

MIMO systems where multiple antennas are used at both the transmitter and receiver have been also acknowledged as one of the most promising techniques to achieve dramatic improvement in physical-layer performance [6], [7]. Moreover, the use of multiple antennas enables space-division multiple access (SDMA), which allows intracell bandwidth reuse by multiplexing spatially separable users [8], [9]. Channel variation in the spatial domain also provides an inherent degree of freedom for adaptive transmission. To sum up, after OFDM is combined with MIMO techniques, MIMO-OFDM can be a potential candidate for the next generation wireless communication systems.

In our system, we refer to the IEEE 802.11a standard [10], and further extend the SISO-OFDM system to MIMO-OFDM system with two transmitted antennas and two received antennas. Six OFDM symbols modulated by 64-tap IFFT are attached after ten short preambles and two long preambles, where the detailed structure of preamble will be discussed in Section 2.2.4. Furthermore, twelve zero tones are inserted on predefined subcarriers into every OFDM symbol so as to diminish the interference caused by adjacent signals, and four pilot tones are also inserted in a symmetric way for the sake of tracking the phase drift at the receiver. Data tones are transmitted in the remaining forty-eight subcarriers. Specification of MIMO-OFDM baseband transceiver is shown in Table 2.1.

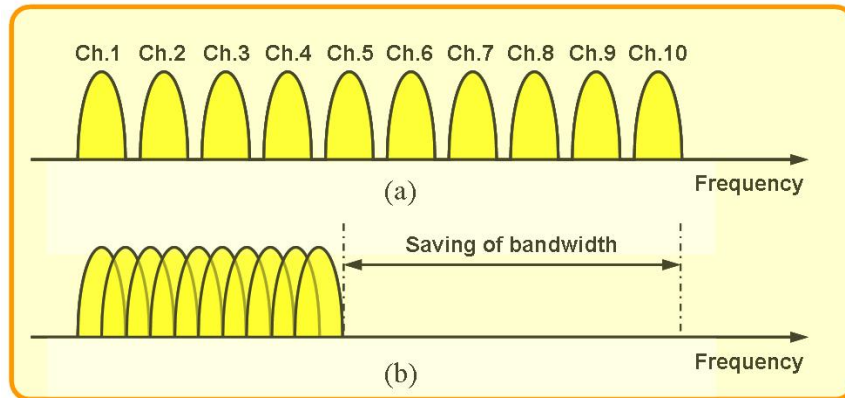


Figure 2.1: (a) Conventional multicarrier technique
(b) Orthogonal multicarrier modulation technique

Table 2.1: Specification of MIMO-OFDM baseband transceiver

Number of Transmit Antennas	2
Number of Receive Antennas	2
Number of Long Preambles / Packet	2
Number of Short Preambles / Packet	10
Number of OFDM Symbols / Packet	6
Number of Data Tones / Symbol	48
Number of Zero Tones / Symbol	12
Number of Pilot Tones / Symbol	4
FFT Size	64 {-32:31}
Long Preamble Size	64+16 (CP)
Short Preamble Size	16
Locations of Data Tones	{-26:-22, -20:-8, -6:-1,1:6, 8:20, 22:26}
Locations of Zero Tones	{-32:-27, 0, 27:31}
Locations of Pilot Tones	{-21, -7, 7, 21}

2.2 Transmitter Architecture

The baseband MIMO-OFDM transmitter architecture is shown in Figure 2.2 [11]. The source data is first fed into the channel encoder, e.g., using the convolution code for error correction at the receiver. The encoded output is then interleaved by distributing the same coded bits into different positions in the packet so that the transmitted information is better resistant to the channel distortion. A MIMO system is

typically designed to meet two different, yet opposite, targets: either to achieve high spectral efficiency, e.g., the VBLAST scheme suggested by Foschini et al. [3] [4], or to improve the transmission reliability against channel fading, e.g., the space-time block codes (STBC) [2] first discovered by Alamouti for two transmit antennas. The preamble channels, coded by the rule of STBC, will be attached in front of the data channel modulated by IFFT. Finally, all traffic data composed of the preamble part and data channels are sent to individual DA modules to convert the baseband signals onto the desired frequency band.

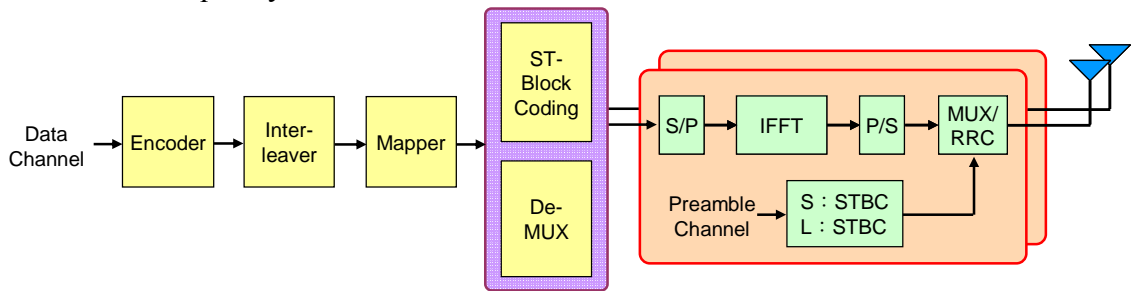


Figure 2.2: Transmitter architecture of MIMO-OFDM system

2.2.1 Convolutional Encoder

A convolutional encoder typically will generate two or three output bits for each input bit. The output bits are dependent on the current input bit, as well as the state of the encoder. The state of the encoder is represented by several bits which precede the current bit. Figure 2.3 shows a convolutional encoder adopted in our system with code rate equal to $1/3$ and constraint length equal to 5. Convolutional coding adds redundant bits in such a way that the decoder can, within limits, detect errors and correct them.

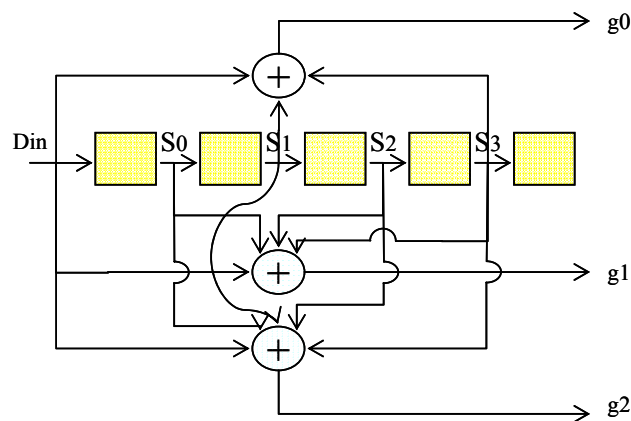


Figure 2.3: Convolutional encoder with code rate $1/3$ and constraint length 5

2.2.2 Interleaver / De-interleaver

In real life, bit errors often occur in bursts due to the fact that linear-fading dips affect several consecutive bits. Unfortunately, the convolutional encoder is most effective in detecting and correcting single random errors and is not effective when errors occur in bursts. Interleaving is the reordering of data coming out of a convolutional encoder prior to transmission so that consecutive bits of data are distributed over a larger sequence of data to reduce the effect of burst errors. At the receiver, the reverse permutation is performed before decoding. A commonly used interleaving scheme is the block interleaving, where the input bits are written in a matrix column by column and read out row by row.

Referring to institute of electrical and electronics engineers IEEE 802.11a standard [10], we use a block interleaver as shown in Figure 2.4. In the standard, the interleaving depth is suggested being the length of an OFDM symbol. Each coded data symbol after convolutional encoder contains 96 bits. Therefore the interleaving depth we adopt is 96, as illustrated in the figure. The interleaver satisfies the following expression

$$j = 6 \times (v \bmod 16) + \lfloor v/16 \rfloor \quad (2.1)$$

where v is the index of input coded data, and $v = 0, 1, \dots, 95$; j is the index of output interleaved data; $\lfloor m \rfloor$ is the greatest integer smaller than m .

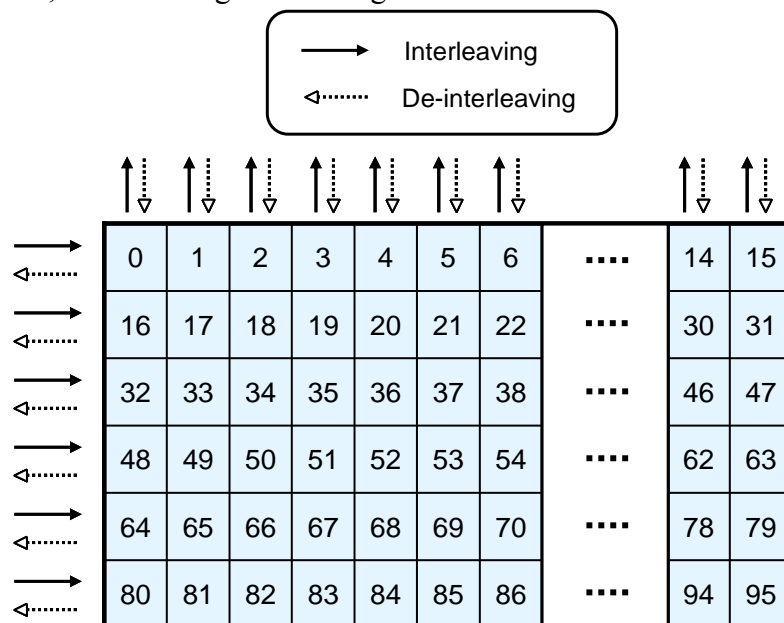


Figure 2.4: Interleaver and de-interleaver schemes

2.2.3 Mapper / De-mapper

Quadrature amplitude modulation (QAM) is the most popular type of modulation using in the OFDM system. The rectangular constellations are especially easy to implement as they can be split into independent in-phase and quadrature parts. A mapper is used to map a small group of bits into a symbol according to the rectangular constellation adopted. Figure 2.5 shows the rectangular constellations of Quadrature Phase Shift Keying (QPSK), 16-QAM, and 64-QAM. The higher modulation order the mapper adopts, the more information a symbol can carry, yet higher modulation order always suffers from interference more severely. In our system, we only adopt QPSK as our modulation scheme.

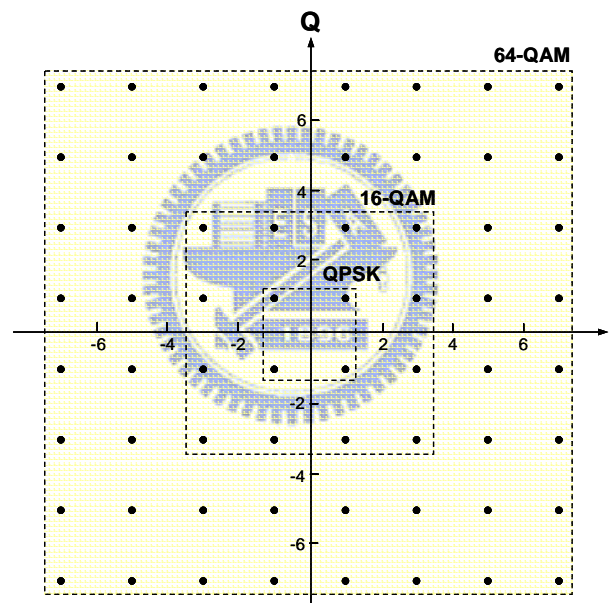


Figure 2.5: QPSK, 16-QAM, and 64-QAM constellations

2.2.4 Preamble Channel and Frame Structure

Referring IEEE 802.11a standard [10], we attach the training sequence, also called preamble, in front of every packet. At the receiver, preambles can be utilized to do a number of tasks, such as timing synchronization, frequency synchronization, and channel estimation. The format of preamble channel and frame structure is shown in Figure 2.6 [10]. Preambles can further be separated into short preamble and long

preamble, and both short and long preamble are modulated by BPSK and encoded by STBC scheme. Short preamble, as implied by the name, has a shorter length compared with long preamble. Each short preamble symbol contains 16 bits with time-span $0.8 \mu\text{s}$, and ten symbols form a complete short preamble with a total time-span of $8 \mu\text{s}$. The following parts are two long preamble symbols, and each one is protected by a guard interval filled with its cyclic extension, which have a total time-span of $8 \mu\text{s}$. After preamble channel, data symbols with cyclic extension follow. There are four pilot tones embedded symmetrically in every data symbol. Note that the first non-preamble symbol is designed for signaling in the standard, such as code rate and modulation order.

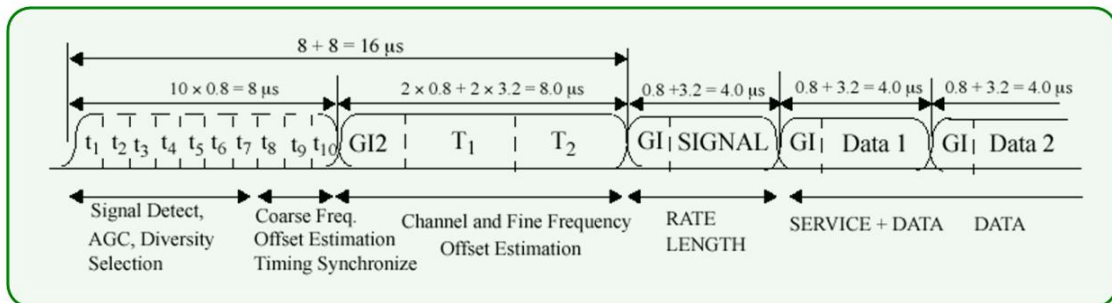


Figure 2.6: Training sequence and frame structure of IEEE 802.11a standard

2.2.5 Root Raised Cosine Filter

Root raised cosine (RRC) filter is commonly used in digital communication systems to limit ISI. The ideal root raised cosine filter, frequency response consists of unity gain at low frequencies, the square root of raised cosine function in the middle, and total attenuation at high frequencies. The width of the middle frequencies is defined by the roll off factor constant β ($0 < \beta < 1$). The root raised cosine filter is generally used in series pairs, so that the total filtering effect is that of a raised cosine filter. The advantage is that if the transmit side filter is stimulated by an impulse, then the receive side filter is forced to filter an input pulse shape that is identical to its own impulse response, thereby setting up a matched filter and maximizing signal to noise ratio (SNR) while at the same time minimizing ISI.

Mathematically, the frequency response $F_{rrc}(\omega)$ may be written as

$$F_{rrc}(\omega) = \begin{cases} 1 & \text{For } \omega \leq \omega_c(1 - \beta) \\ 0 & \text{For } \omega \geq \omega_c(1 + \beta) \\ \sqrt{\frac{1 + \cos\left(\frac{\pi(\omega - \omega_c(1 - \beta))}{2\beta\omega_c}\right)}{2}} & \text{For } \omega_c(1 - \beta) < \omega < \omega_c(1 + \beta) \end{cases} \quad (2.2)$$

where ω_c is half the data rate.

2.3 Receiver Architecture

The baseband function diagram of the proposed MIMO-OFDM receiver is shown in Figure 2.9 [11]. The received signal is first down-converted to the baseband. After passing through RRC, data streams are processed by FFT so as to demodulate the OFDM symbol. A space-time detector is used for separating the multi-antenna signals. Since the transmitted signals can be space-time block encoded or spatially multiplexed, the corresponding decoding scheme such as space-time block detector or VBLAST detector has to be performed. The detected symbol streams are then de-interleaved, followed by a Viterbi decoder to recover the source bits. To acquire the channel information, long preamble is used to do frequency domain channel estimation. We will also include pilot subcarriers inserted in data channel to estimate the phase shift in a symbol to further improve the performance.

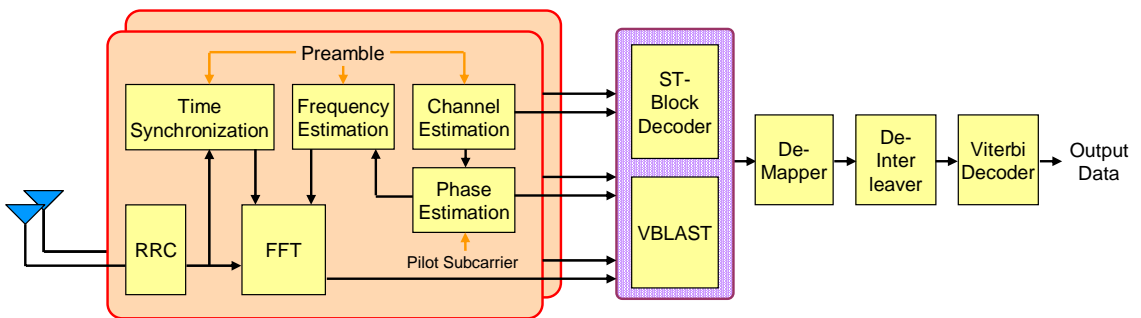


Figure 2.7: Receiver architecture of MIMO-OFDM system

2.3.1 Timing Synchronizer

Before an OFDM receiver can demodulate the subcarriers, it has to find out where the symbol boundaries are and what the optimal timing instants are to minimize the effects of ICI and ISI. Moreover, timing synchronization can be divided into coarse timing synchronization and fine timing synchronization [11] - [13].

The task of the coarse timing synchronization is to identify the preamble in order to detect a packet arrival. Here we discuss the coarse timing synchronization algorithm adopted in our system. First, short preamble is chosen to do our coarse timing synchronization, and a matched filter (MF) which can match a short preamble symbol is designed. After passing the received signal through MF, we can obtain ten peak values where two adjacent peaks are at the interval of 16. To acquire a more accurate frame position, data after MF is further passed to a finite impulse response (FIR) filter so that we can obtain a succession of increasing peaks and finally choose the time instant a deterministic delay away from the maximum value as the frame start.

The fine timing synchronization in an OFDM system decides where to place the start of the FFT window within the OFDM symbol. Although an OFDM system exhibits a guard interval, making it somewhat robust against timing offsets, non-optimal symbol timing will cause more ISI and ICI in delay spread environments. This will result in performance degradation. To eliminate timing offset induced by different path delays, fine timing synchronization will be performed after coarse timing synchronization.

2.3.2 Frequency Synchronizer

The purpose of frequency synchronization is to correct the frequency offset, which is caused by the difference of oscillator frequencies at the transmitter and the receiver. Frequency offset may result in the loss of the orthogonality between subcarriers and degrade the system performance. Therefore, we try to estimate the frequency offset and compensate the received signals.

Assuming that the absolute value of the frequency offset does not exceed $\frac{1}{2DT_d}$,

where D is the delay between the identical samples of the two symbols; T_d denotes the sampling period, then the estimated frequency offset $\Delta\hat{f}$ can be shown by

$$\Delta\hat{f} = \frac{\hat{\phi}}{2\pi DT_d} \quad (2.3)$$

where $\hat{\phi}$ denotes the estimated phase shift through two adjacent symbols, which can be computed by an arc tangent of the summation of conjugate multiplications between two identical samples of the two repeated symbols. To do the above task, the preamble channel becomes the most proper candidate.

The 802.11a standard specifies a maximum oscillator error of 20 ppm, therefore the total maximum error is 40 ppm. Supposing that the carrier frequency is 5.3 GHz, the maximum possible frequency error is about 212 kHz. Owing to the inherent structures of short preamble and long preamble, the maximum unambiguous estimated frequency offset is 625 kHz for short preamble and 156.25 kHz for long preamble. Therefore, both short preamble and long preamble are required to estimate frequency offset so as to cover the probable frequency offset specified by the standard.

2.3.3 Channel Estimator

The channel can be estimated using the known training symbols within the preamble. In our system, owing to the same symbol structure as data symbols, long preamble becomes the best candidate for performing this job. Moreover, since preamble channel is BPSK modulated and two long preambles are identical, the space-time block encoded signal model can be denoted as

$$\begin{bmatrix} t & -t \\ t & t \end{bmatrix} \quad (2.4)$$

where t denotes the time domain long preamble sequence.

Supposing that T^k denotes the long preamble chips (in frequency domain); H_{pq} is the channel frequency response from the p th transmit antenna to the q th receive antenna; Z denotes the received signal after passing FFT, then the noise free post-FFT received signal at the k th subcarrier can be shown as follows

$$\begin{aligned}
\begin{bmatrix} Z_1^k(n) & Z_1^k(n+1) \\ Z_2^k(n) & Z_2^k(n+1) \end{bmatrix} &= \begin{bmatrix} H_{11}^k & H_{21}^k \\ H_{12}^k & H_{22}^k \end{bmatrix} \cdot \begin{bmatrix} T^k & -T^k \\ T^k & T^k \end{bmatrix} \\
&= \begin{bmatrix} T^k H_{11}^k + T^k H_{21}^k & -T^k H_{11}^k + T^k H_{21}^k \\ T^k H_{12}^k + T^k H_{22}^k & -T^k H_{12}^k + T^k H_{22}^k \end{bmatrix}
\end{aligned} \tag{2.5}$$

where the superscript k denotes the k th subcarrier; suffix 1 or 2 denotes received antenna 1 or 2, and n or $(n+1)$ represents the n th or $(n+1)$ th symbol.

Based on the structure of the received signal shown above, the estimated channel frequency response \hat{H}_{pq}^k can be obtained simply by the following equations

$$\begin{aligned}
\hat{H}_{11}^k &= \frac{1}{2T^k} (Z_1^k(n) - Z_1^k(n+1)) = H_{11}^k \\
\hat{H}_{21}^k &= \frac{1}{2T^k} (Z_1^k(n) + Z_1^k(n+1)) = H_{21}^k \\
\hat{H}_{12}^k &= \frac{1}{2T^k} (Z_2^k(n) - Z_2^k(n+1)) = H_{12}^k \\
\hat{H}_{22}^k &= \frac{1}{2T^k} (Z_2^k(n) + Z_2^k(n+1)) = H_{22}^k
\end{aligned} \tag{2.6}$$

2.3.4 Phase Estimator

The processing of the preamble takes care of the initial synchronization of the MIMO-OFDM receiver. It is, however, likely that the frequency offset will vary during the reception of the packet, making solely initial frequency synchronization insufficient. Furthermore, the system will experience phase noise invoked by the combination of the RF oscillator and the phase-locked loop (PLL). It is, therefore, necessary to estimate and correct the rotation of the received constellation points by using pilots which are embedded in data symbols.

Recalling that there are four pilot tones P^k , $k \in \pm 7, \pm 21$, in every data symbol, and pilot tones in successive symbols are encoded in STBC scheme. First we get the post-FFT received signal without phase noise at the k th pilot subcarriers ($k = \pm 7, \pm 21$) of n th and $(n+1)$ th symbols on antenna 1 and antenna 2 by

$$\begin{aligned}
\begin{bmatrix} Z_1^k(n) & Z_1^k(n+1) \\ Z_2^k(n) & Z_2^k(n+1) \end{bmatrix} &= \begin{bmatrix} H_{11}^k & H_{21}^k \\ H_{12}^k & H_{22}^k \end{bmatrix} \cdot \begin{bmatrix} P_1^k & -(P_2^k)^* \\ P_2^k & (P_1^k)^* \end{bmatrix} \\
&= \begin{bmatrix} P_1^k H_{11}^k + P_2^k H_{21}^k & -(P_2^k)^* H_{11}^k + (P_1^k)^* H_{21}^k \\ P_1^k H_{12}^k + P_2^k H_{22}^k & -(P_2^k)^* H_{12}^k + (P_1^k)^* H_{22}^k \end{bmatrix}
\end{aligned} \tag{2.7}$$

Then, adding the effects of phase noise ϕ by multiplying a $e^{j\phi(n)}$ term, the original equation will lead to the following form

$$\begin{bmatrix} \left\{ P_1^k H_{11}^k + P_2^k H_{21}^k \right\} e^{j\phi_1(n)} & \left\{ -(P_2^k)^* H_{11}^k + (P_1^k)^* H_{21}^k \right\} e^{j\phi_1(n+1)} \\ \left\{ P_1^k H_{12}^k + P_2^k H_{22}^k \right\} e^{j\phi_2(n)} & \left\{ -(P_2^k)^* H_{12}^k + (P_1^k)^* H_{22}^k \right\} e^{j\phi_2(n+1)} \end{bmatrix} \tag{2.8}$$

Therefore, the estimated phase shift $\hat{\phi}(n)$ can be obtained by

$$\begin{aligned}
\hat{\phi}_1(n) &= \angle \left\{ \sum_{k=\pm 7, \pm 21} \left\{ P_1^k \hat{H}_{11}^k + P_2^k \hat{H}_{21}^k \right\}^* \cdot Z_1^k(n) \right\} \\
\hat{\phi}_1(n+1) &= \angle \left\{ \sum_{k=\pm 7, \pm 21} \left\{ -(P_2^k)^* \hat{H}_{11}^k + (P_1^k)^* \hat{H}_{21}^k \right\}^* \cdot Z_1^k(n) \right\} \\
\hat{\phi}_2(n) &= \angle \left\{ \sum_{k=\pm 7, \pm 21} \left\{ P_1^k \hat{H}_{12}^k + P_2^k \hat{H}_{22}^k \right\}^* \cdot Z_1^k(n) \right\} \\
\hat{\phi}_2(n+1) &= \angle \left\{ \sum_{k=\pm 7, \pm 21} \left\{ -(P_2^k)^* \hat{H}_{12}^k + (P_1^k)^* \hat{H}_{22}^k \right\}^* \cdot Z_1^k(n) \right\}
\end{aligned} \tag{2.9}$$

Instead of doing correlation between adjacent samples and averaging all the symbols, the scheme used by the phase estimator only averages the phase residue among four pilot tones in each symbol.

2.3.5 Viterbi Decoder

Decoding of convolutional codes is most often performed by the Viterbi decoder, which is an efficient way to obtain the optimal maximum likelihood estimate of the encoded sequence. Viterbi decoder can be further divided into hard-decision and soft-decision decoding, where hard-decision is adopted in our system. According to the design of the convolutional encoder in transmitter, we can derive the state transition table in Table 2.2 and then further illustrate the trellis diagram as shown in Figure 2.8

and Figure 2.9.

The Viterbi algorithm is a recursive sequential minimization algorithm that can be used to find the least expensive way to route symbols from one edge of a state diagram to another. To do this, the algorithm uses a cost analysis mechanism to calculate the distance between the received symbol and the symbol associated to that edge.

The distance between the received symbol s and the symbol associated to that edge in the state diagram is often referred to as the branch metric. If $BM [i, j](s)$, is the metric of the branch from state i to state j , the problem is finding the path for which the metric, i.e. the sum of the branch metrics of the path edges, is at a minimum. The Viterbi algorithm solves this problem by applying the following recursive equation for each state transition

$$PM [j](t) = \min (PM [i](t-1) + BM [i, j](s)) \quad (2.10)$$

where $PM [j](t)$ is the path metric associated to the (minimum cost path leading to) state j at time t . At the end of the decoding, it is possible to reconstruct the maximum likelihood sequence through a trace back starting from the possible decoder states.

Normally for decoders using non-punctured codes, the trace back depth equals five-times constraint length, which is sufficient to decode the correct output in the presence of noise. In our system, constraint length equals 5; therefore an appropriate trace back depth is 25.

Table 2.2: State transition table

din	s0	s1	s2	s3	state	g2	g1	g0	s0'	s1'	s2'	s3'	state'	din	s0	s1	s2	s3	state	g2	g1	g0	s0'	s1'	s2'	s3'	state'		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	8
0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	8	
0	0	0	1	0	2	1	1	0	0	0	0	1	1	1	0	0	1	0	2	0	0	1	1	0	0	0	1	9	
0	0	0	1	1	3	0	0	1	0	0	0	1	1	1	0	0	1	1	3	1	1	0	1	0	0	0	1	9	
0	0	1	0	0	4	1	0	1	0	0	1	0	2	1	0	1	0	0	4	0	1	0	1	0	1	0	10		
0	0	1	0	1	5	0	1	0	0	0	1	0	2	1	0	1	0	1	5	1	0	1	1	0	1	0	10		
0	0	1	1	0	6	0	1	1	0	0	1	1	3	1	0	1	1	0	6	1	0	0	1	0	1	0	11		
0	0	1	1	1	7	1	0	0	0	0	1	1	3	1	0	1	1	1	7	0	1	1	1	1	0	1	11		
0	1	0	0	0	8	1	1	0	0	1	0	0	4	1	1	0	0	0	8	0	0	1	1	1	0	0	12		
0	1	0	0	1	9	0	0	1	0	1	0	0	4	1	1	0	0	1	9	1	1	0	1	1	0	0	12		
0	1	0	1	0	10	0	0	0	0	1	0	1	5	1	1	0	1	0	10	1	1	1	1	1	0	1	13		
0	1	0	1	1	11	1	1	1	0	1	0	1	5	1	1	0	1	1	11	0	0	0	1	1	0	1	13		
0	1	1	0	0	12	0	1	1	0	1	1	0	6	1	1	1	0	0	12	1	0	0	1	1	1	0	14		
0	1	1	0	1	13	1	0	0	0	1	1	0	6	1	1	1	0	1	13	0	1	1	1	1	1	0	14		
0	1	1	1	0	14	1	0	1	0	1	1	1	7	1	1	1	0	0	14	0	1	0	1	1	1	1	15		
0	1	1	1	1	15	0	1	0	0	1	1	1	7	1	1	1	1	1	15	1	0	1	1	1	1	1	15		

2.4 MIMO Techniques

The MIMO techniques for wireless communication improve the signal quality of the receiver on one side of the link by simple processing across two antennas on the opposite side. These schemes could be very attractive in wireless communication applications where the performance of the system is limited by multipath fading. MIMO techniques can basically be split into two groups: spatial diversity technique [14] and spatial multiplexing technique [7] [15] [16]. Spatial diversity technique increases the performance of the communication system by coding over the different transmitter branches, whereas spatial multiplexing technique achieves a higher throughput by transmitting independent data streams on the different transmit branches simultaneously and at the same carrier frequency. In the following sections, we will explain the MIMO techniques adopted in our system in detail.

2.4.1 Spatial Diversity Technique

In wireless communication systems, diversity techniques are widely used to reduce the effects of multipath fading and improve the reliability of transmission without increasing the transmitted power or sacrificing the bandwidth. Diversity techniques are classified into time, frequency, and space diversity. Space diversity, also called antenna diversity, can be further classified into two categories, transmit diversity and receive diversity. Among various transmit diversity schemes, STBC is the most popular scheme with the feature of open loop (i.e., no feedback signaling is required) as channel information is not required at the transmitter. Therefore we will focus on the scheme of STBC in this section.

The space-time block coding scheme was first discovered by Alamouti [2] for two transmit antennas. Symbols transmitted from those antennas are encoded in both space and time in a simple manner to ensure that transmissions from both the antennas are orthogonal to each other. This would allow the receiver to decode the transmitted information with a slight increment in the computational complexity. In the following discussion, we will give an overview of Alamouti's scheme.

Considering about the adopted 2x2 MIMO-OFDM system, the input symbols to the space-time block encoder are divided into groups of two symbols. At a given symbol period, the encoder takes a block of two modulated symbols X_1^k and X_2^k in each encoding operation and maps them to the transmit antennas according to a code matrix given by

$$\begin{bmatrix} X_1^k & -(X_2^k)^* \\ X_2^k & (X_1^k)^* \end{bmatrix} \quad (2.11)$$

The encoded outputs are transmitted in two consecutive transmission periods from two transmit antennas. Let H_{pq}^k be the channel frequency response from the p th transmitted antenna to the q th received antenna on subcarrier k , then the noise free post-FFT received signal, Z_1^k and Z_2^k , can be expressed as

$$\begin{aligned} \begin{bmatrix} Z_1^k(n) & Z_1^k(n+1) \\ Z_2^k(n) & Z_2^k(n+1) \end{bmatrix} &= \begin{bmatrix} H_{11}^k & H_{21}^k \\ H_{12}^k & H_{22}^k \end{bmatrix} \cdot \begin{bmatrix} X_1^k & -(X_2^k)^* \\ X_2^k & (X_1^k)^* \end{bmatrix} \\ &= \begin{bmatrix} X_1^k H_{11}^k + X_2^k H_{21}^k & -(X_2^k)^* H_{11}^k + (X_1^k)^* H_{21}^k \\ X_1^k H_{12}^k + X_2^k H_{22}^k & -(X_2^k)^* H_{12}^k + (X_1^k)^* H_{22}^k \end{bmatrix} \end{aligned} \quad (2.12)$$

Here we advanced take phase noise effects into consideration, and then the original equation will lead to

$$\begin{bmatrix} \left\{ X_1^k H_{11}^k + X_2^k H_{21}^k \right\} e^{j\phi_1(n)} & \left\{ -(X_2^k)^* H_{11}^k + (X_1^k)^* H_{21}^k \right\} e^{j\phi_1(n+1)} \\ \left\{ X_1^k H_{12}^k + X_2^k H_{22}^k \right\} e^{j\phi_2(n)} & \left\{ -(X_2^k)^* H_{12}^k + (X_1^k)^* H_{22}^k \right\} e^{j\phi_2(n+1)} \end{bmatrix} \quad (2.13)$$

Since we have obtained the estimated channel \hat{H} and the estimated phase $\hat{\phi}$ before this stage, we can easily calculate the detected signals \hat{D}_1^k and \hat{D}_2^k by Eq. 2.14

$$\begin{aligned} \hat{D}_1^k &= (\hat{H}_{11}^k)^* \cdot Z_1^k(n) \cdot e^{-j\hat{\phi}_1(n)} + \hat{H}_{21}^k \cdot (Z_1^k(n+1))^* \cdot e^{j\hat{\phi}_1(n+1)} + \\ &\quad (\hat{H}_{12}^k)^* \cdot Z_2^k(n) \cdot e^{-j\hat{\phi}_2(n)} + \hat{H}_{22}^k \cdot (Z_2^k(n+1))^* \cdot e^{j\hat{\phi}_2(n+1)} \\ \hat{D}_2^k &= (\hat{H}_{21}^k)^* \cdot Z_1^k(n) \cdot e^{-j\hat{\phi}_1(n)} - \hat{H}_{11}^k \cdot (Z_1^k(n+1))^* \cdot e^{j\hat{\phi}_1(n+1)} + \\ &\quad (\hat{H}_{22}^k)^* \cdot Z_2^k(n) \cdot e^{-j\hat{\phi}_2(n)} - \hat{H}_{12}^k \cdot (Z_2^k(n+1))^* \cdot e^{j\hat{\phi}_2(n+1)} \end{aligned} \quad (2.14)$$

Moreover, assuming that the estimated channel \hat{H} and the estimated phase $\hat{\phi}$ are accurate, that is, $\hat{H} = H$ and $\hat{\phi} = \phi$, the results of Eq. 2.14 will turn as follows

$$\begin{aligned}\hat{D}_1^k &= \left(|H_{11}^k|^2 + |H_{21}^k|^2 + |H_{12}^k|^2 + |H_{22}^k|^2 \right) \cdot X_1^k \\ \hat{D}_2^k &= \left(|H_{11}^k|^2 + |H_{21}^k|^2 + |H_{12}^k|^2 + |H_{22}^k|^2 \right) \cdot X_2^k\end{aligned}\quad (2.15)$$

where $\left(|H_{11}^k|^2 + |H_{21}^k|^2 + |H_{12}^k|^2 + |H_{22}^k|^2 \right)$ is a diversity gain.

2.4.2 Spatial Multiplexing Technique

Spatial multiplexing technique multiplexes multiple spatial channels to send as many independent data as possible over different antennas for a specific error rate. There are four spatial multiplexing schemes: diagonal BLAST (DBLAST), horizontal BLAST, VBLAST, and turbo BLAST [17]. Of them, VBLAST is the most promising for its implementation simplicity, which is adopted in our system. Hereafter the equation derivations are held under the hypothesis of VBLAST scheme being used.

In transmitter side, the encoding process is simply a multiplex operation followed by independent substreams. No inter-substream coding, or coding of any kind, is required. The transmitted signal in frequency domain is given by

$$\begin{bmatrix} X_1^k \\ X_2^k \end{bmatrix}\quad (2.16)$$

In the receiver side, the signals after effects of MIMO channel and procedure of FFT, denoted as Z_1^k and Z_2^k , can be expressed as follows:

$$\begin{bmatrix} Z_1^k(n) \\ Z_2^k(n) \end{bmatrix} = \begin{bmatrix} H_{11}^k & H_{21}^k \\ H_{12}^k & H_{22}^k \end{bmatrix} \begin{bmatrix} X_1^k \\ X_2^k \end{bmatrix} = \begin{bmatrix} H_{11}^k X_1^k + H_{21}^k X_2^k \\ H_{12}^k X_1^k + H_{22}^k X_2^k \end{bmatrix}\quad (2.17)$$

After adding phase noise, the equation becomes

$$\begin{bmatrix} Z_1^k(n) \\ Z_2^k(n) \end{bmatrix} = \begin{bmatrix} \{H_{11}^k X_1^k + H_{21}^k X_2^k\} e^{j\hat{\phi}_1(n)} \\ \{H_{12}^k X_1^k + H_{22}^k X_2^k\} e^{j\hat{\phi}_2(n)} \end{bmatrix}\quad (2.18)$$

Furthermore, the estimated channel matrix $\hat{\mathbf{H}}$ can be obtained in channel estimation stage, where $\hat{\mathbf{H}}$ plays an important roll in the VBLAST decoding.

$$\hat{\mathbf{H}}^k = \begin{bmatrix} \hat{H}_{11}^k & \hat{H}_{21}^k \\ \hat{H}_{12}^k & \hat{H}_{22}^k \end{bmatrix} \quad (2.19)$$

VBLAST decoding can be separated into two steps. The first step is **interference nulling**, and the second step is **interference cancellation**. Nulling is done by linearly weighting the received signals by \mathbf{W}^k so as to satisfy some performance-related criterion, such as minimum mean-square error (MMSE) or zero-forcing (ZF). The detected signals after nulling, denoted as \hat{D}_1^k and \hat{D}_2^k , can be shown as follows

$$\begin{bmatrix} \hat{D}_1^k(n) \\ \hat{D}_2^k(n) \end{bmatrix} = \mathbf{W}^k \cdot \begin{bmatrix} Z_1^k(n) \cdot e^{-j\hat{\phi}_1(n)} \\ Z_2^k(n) \cdot e^{-j\hat{\phi}_2(n)} \end{bmatrix} \quad (2.20)$$

where \mathbf{W}^k can be calculated by the following equation when ZF criterion is adopted

$$\mathbf{W}^k = (\hat{\mathbf{H}}^k)^+ = \hat{\mathbf{H}}^k ((\hat{\mathbf{H}}^k)^H \hat{\mathbf{H}}^k)^{-1} \quad (2.21)$$

Otherwise, \mathbf{W}^k is given by Eq. 2.21 when MMSE criterion is adopted

$$\mathbf{W}^k = \left\{ \hat{\mathbf{H}}^k (\hat{\mathbf{H}}^k)^H + \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix} \right\}^{-1} \hat{\mathbf{H}}^k \quad (2.22)$$

where σ^2 is the noise power. In our proposed system, only ZF is adopted.

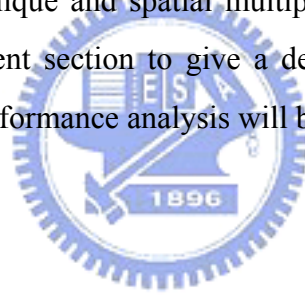
After nulling, interference from already-detected components \hat{D}_1^k and \hat{D}_2^k is subtracted out from the received signal, resulting in a modified received vector where less interferences are present, which is called interference cancellation. The most general interference cancellation skills are successive interference cancellation (SIC) and parallel interference cancellation (PIC). In our 2x2 MIMO system, SIC is adopted. Assuming that both \hat{H} and $\hat{\phi}$ are estimated correctly, then the detected signals after interference cancellation, denote as \hat{D}_1^k and \hat{D}_2^k , can be shown to be

$$\begin{aligned} \hat{D}_1^k(n) &= \begin{bmatrix} (\hat{H}_{11}^k)^* & (\hat{H}_{12}^k)^* \end{bmatrix} \begin{bmatrix} Z_1^k(n)e^{-j\hat{\phi}_1(n)} - \hat{H}_{21}^k \hat{D}_2^k(n) \\ Z_1^k(n)e^{-j\hat{\phi}_2(n)} - \hat{H}_{22}^k \hat{D}_2^k(n) \end{bmatrix} \\ &= \left(|H_{11}^k|^2 + |H_{12}^k|^2 \right) \cdot X_1^k \\ \hat{D}_2^k(n) &= \begin{bmatrix} (\hat{H}_{21}^k)^* & (\hat{H}_{22}^k)^* \end{bmatrix} \begin{bmatrix} Z_1^k(n)e^{-j\hat{\phi}_1(n)} - \hat{H}_{11}^k \hat{D}_1^k(n) \\ Z_1^k(n)e^{-j\hat{\phi}_2(n)} - \hat{H}_{12}^k \hat{D}_1^k(n) \end{bmatrix} \\ &= \left(|H_{21}^k|^2 + |H_{22}^k|^2 \right) \cdot X_2^k \end{aligned} \quad (2.23)$$

Therefore after we divide the scaling $\left(|H_{11}^k|^2 + |H_{12}^k|^2\right)$ and $\left(|H_{21}^k|^2 + |H_{22}^k|^2\right)$, the original signal X_1^k and X_2^k can then be recovered.

2.5 Summary

In this chapter, we first introduce the MIMO-OFDM system, and propose our system architecture including the transmitter and receiver. We also give the description of all functional blocks in the order of data passing through a system. At the transmitter, convolutional encoder, interleaver, mapper, adding preamble channel, and frame structure are gone through. At the receiver, synchronization is first mentioned, which consists of coarse timing, fine timing, frequency, and phase synchronizations. Then, channel estimation, de-mapper, de-interleaver, and Viterbi decoder are described in the rest part of the receiver. Finally, we highlight the space-time coding techniques, also called space diversity technique and spatial multiplexing technique, implemented on the system as an independent section to give a detailed introduction. More detailed experimental results and performance analysis will be given in Chapter 4.



Chapter 3

MIMO-OFDM System Platforms

In Chapter 3, we will introduce the development environment, including fast prototyping platform and self-designed platform. The fast prototyping platform is chosen to be our initial verification platform of MIMO-OFDM baseband algorithms, since the debugging interface is much more convenient for designers and the system is much simpler than another platform. On the other hand, the self-designed platform is used to perform the final verification of whole MIMO-OFDM system including baseband and RF parts, where transmitter and receiver are implemented on two separated boards with their own RF modules each. The self-designed platform is much closer to a real wireless communication system and therefore can take all phenomena and effects of the wireless system into account. In the following sections, hardware modules, software design flows, and the corresponding debugging tools of these two platforms are detailed explained.

3.1 Fast Prototyping Platform

Figure 3.1 shows the development environment of the fast prototyping platform, including Aptix[®] System Explorer with several specific modules, a high speed work station, a digital to analog converter (DA), an analog to digital converter (AD), a logic analyzer (LA), an oscilloscope, and some PCs. A close-up shot of Aptix MP3C is given in Figure 3.2 where the modules installed on Aptix MP3C are highlighted, including three Xilinx FPGA modules, one DSP module, and one USB module.



Figure 3.1: Development environment of fast prototyping system

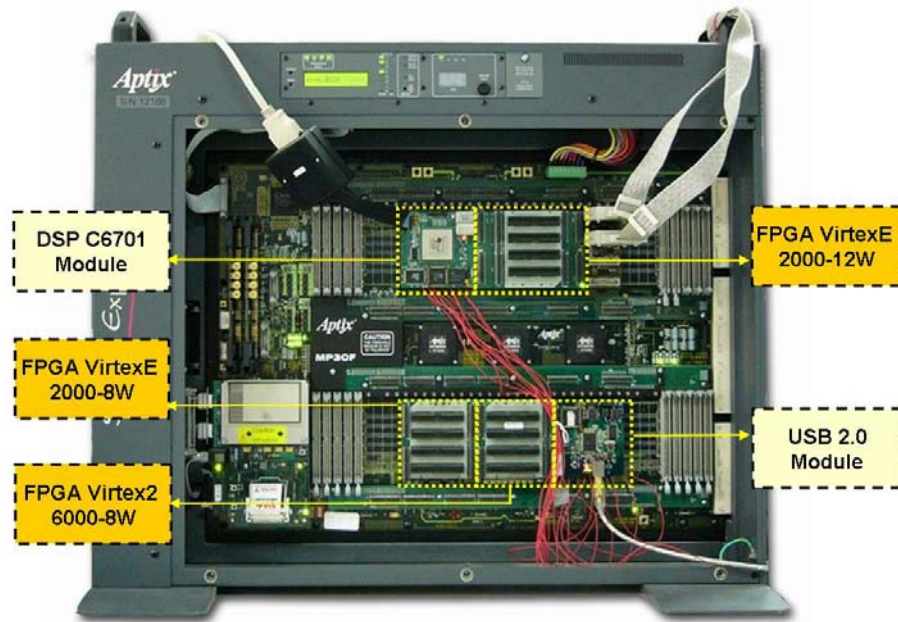


Figure 3.2: Modules installed on Aptix MP3CF platform

3.1.1 Aptix® System Explorer

Under the trend of System on Chip (SoC) and the concept of time-to-market, Aptix® corporation has developed a series of fast prototyping system named MPx, which provides a total solution of real-time verification and integration for industry and high-performance functional simulation for application specific integrated circuit

(ASIC) designer so as to achieve the goal of time-to-market. In our laboratory, we choose Aptix[®] System Explorer MP3CF as our fast prototyping system.

The Aptix[®] MP3CF System Explorer[™] contains two parts, hardware platform called MP3CF FPCB and software called Explorer, on which we will give more introduction in later sections.

3.1.1.1 Hardware: MP3CF Platform

Aptix[®] MP3CF Platform consists of several functional units, such as the onboard micro-controller, the clock generator, some re-programmable inter-connect chips called field programmable interconnect components (FPIC), the main motherboard called field programmable circuit board (FPCB), and some flexible input/output (I/O) buses [18] as illustrated in Figure 3.3.

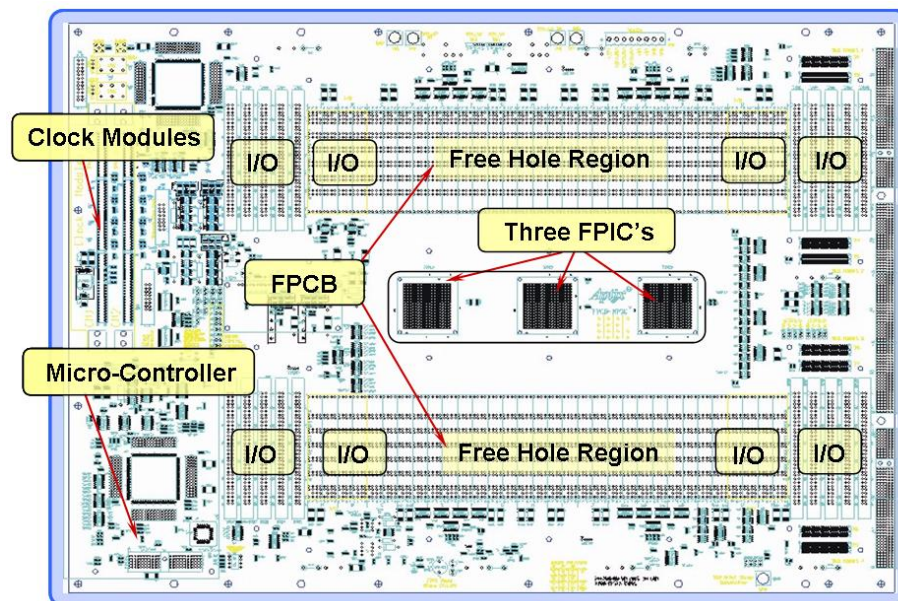


Figure 3.3: Aptix[®] MP3CF platform

Micro-controller mainly takes charge of the operation of the whole platform, such as the control of booting sequence and storing or loading the design of circuit through flash memory; clock generator provides system clock, and supports eight different clock sources from outside; FPIC is responsible for the inter-connect of all modules; FPCB is the place where modules can be installed; I/O bus is the bridge between Aptix[®] platform and devices outside.

Aptix[®] MP3CF is powerful and capable of easy expansion and high integration. It not only supports modules produced by Xilinx Corporation and Altera Corporation, but also those fitting the definition of freehole pins. By the right definition, we can install modules developed by other companies on Aptix[®] MP3CF through an adapter. For example, we developed a DSP C6701 EVM by using the core chip TMS320C6701 DSP of Texas Instruments (TI) and also a CYPRESS USB 2.0 module by using the core chip of CYPRESS CY7C68013, both of them being not the products from Xilinx or Altera. Therefore, by the usage of the adapter, we can integrate different modules on Aptix[®] MP3CF and make the system more flexible and powerful.

3.1.1.2 Software: Explorer

The software (called *Explorer*) provides an easy-to-use, consistent user interface which displays commands through a series of pull-down menus. The main design flow is described as follows and illustrated in Figure 3.4.

(1) Import Design into Explorer

Explorer requires to be informed about the netlist files that we are using in the design including top-level netlist, component netlist, and pinmap file. Top-level netlist is an electronic design interchange format (EDIF) file containing connectivity information between the different components that will be mounted on the MP3CF FPCB. Component netlists are EDIF files containing major design information in each component. All EDIF files can be generated by electronic design automation (EDA) tools that can support synthesis, such as Synplify Pro we adopt.

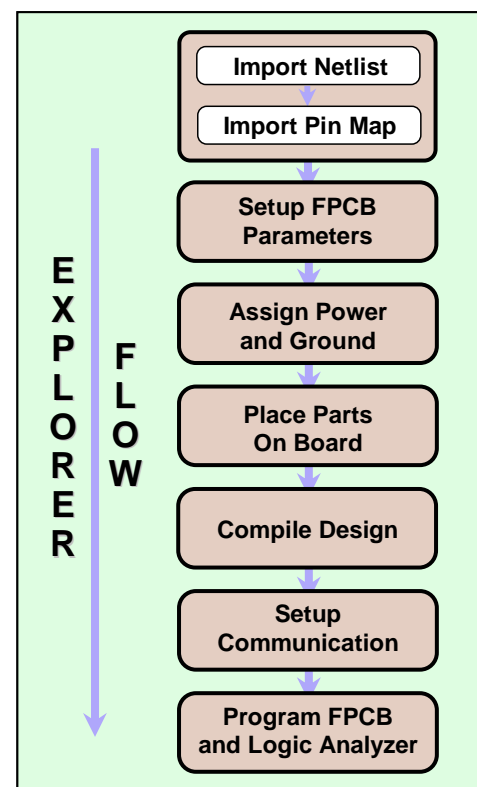


Figure 3.4: Explorer flow

Finally, we have to identify the pinmap file used in the design to assign packages, pins, and other information to those parts.

(2) Setup FPCB Parameters

Explorer can support several different FPCBs. We require specifying which FPCB we are using to develop.

(3) Assign Power and Ground

Some physical parameters of the design need to be set up, such as power and ground nets.

(4) Place Parts on Board

We need to place our design components in their correct positions on the coordinate system. There will be a board view window helping us move a component onto the right place of FPCB by dragging the component to the desired place with a mouse.

(5) Compile Design

The compilation process first maps the FPCB and then maps the existing I/O, clock, bus and FPGA nets to MP3CF hardware. Using the result of FPCB mapping, compilation continues with FPGA place-and-route which will run for all FPGAs in the design. Once the FPGA place and route has been completed successfully, compilation conducts the FPCB routing. The FPCB router routes the FPICs with all nets in the design mapped to the FPGAs. In general, place-and-route is the most time-consuming process of all.

(6) Setup Communication

In this process, we need to do some configurations about communication to program the board and devices. For hardware (FPCB board), we need to specify communication method, address for the method, and whether the flash is to be programmed or not when downloading. For debug (LA), we need to identify communication method, address for the method, and which probing pod of the LA is to be connected with.

(7) Program FPCB and LA

Finally, we can download our design onto FPCB and probing information to the LA, and start to verify our system design.

3.1.2 FPGA Module

In our fast-prototyping system, we use several FPGA modules mounted on Aptix[®] MP3CF platform to implement our communication system. In the following sections, we will give an overview of our FPGA modules. Then we will show the design flow of FPGA.

3.1.2.1 FPGA Overview

The demand for more complex programmable hardware is constantly growing to meet the formidable industry requirement. The major categories of programmable hardware are programmable logic device (PLD) and FPGA. A PLD consists of micro-cells and a central inter-connection logic. Typical PLD applications are “glue logic” for connecting other ASICs. On the other hand, FPGAs consist of even more complex logic block on one chip. Typical applications are central control units (CPU) and DSPs up to very complex SoC design. Therefore, we adopt some FPGA modules to realize our communication system. Generally, FPGA can be categorized into three types by its structure:

1. **Look-up-table (LUT):** Xilinx, Altera, AT&T
2. **Multiplexer:** Actel, Quicklogic
3. **Transistor array:** Cross point

If we focus on its programming architecture, there are two major types:

1. **SRAM:** Xilinx, Altera, AT&T, Atmel
2. **Anti-fuse:** Actel, Cypress, Quicklogic

Static random access memory (SRAM) type has a merit of being able to program repeatedly while Anti-fuse type has the feature of one time programmable (OTP).

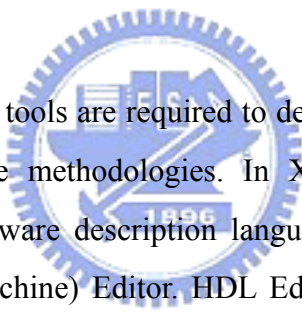
Anti-fuse type can offer security for design but cannot be modified further.

Compared to ASIC, FPGA has lower performance apparently, especially on power consumption and maximum supportable speed. However, as the technique of semiconductor industry grows, FPGA becomes more and more competitive to ASIC. Actually, FPGA has more integration ability and flexibility than ASIC, and undoubtedly, is the best candidate component for a fast-prototyping system.

3.1.2.2 FPGA Design Flow

In our design, we choose Xilinx ISE 7.1 and Synplify Pro 8.2 as the development tool for the first half of the design flow. The second half is done on a workstation with Explorer. Figure 3.5 is the main FPGA design flow and later we will give more information about the flow.

(1) Design Entry



In general, EDA tools are required to develop register transfer level (RTL) codes by appropriate methodologies. In Xilinx ISE 7.1, it supports three methods: HDL (hardware description language) Editor, Schematic Flow, and FSM (finite state machine) Editor. HDL Editor allows us to edit source files directly like VHDL (very high speed integrated circuit hardware description language) [19]-[22] and Verilog [23], which are the most common HDLs in use today. Schematic Flow is another choice to create our source files by drawing the scheme with underlying HDL macros. FSM Editor allows us to edit by timing state diagram, which is suitable for realization controller, such as memory access controller.

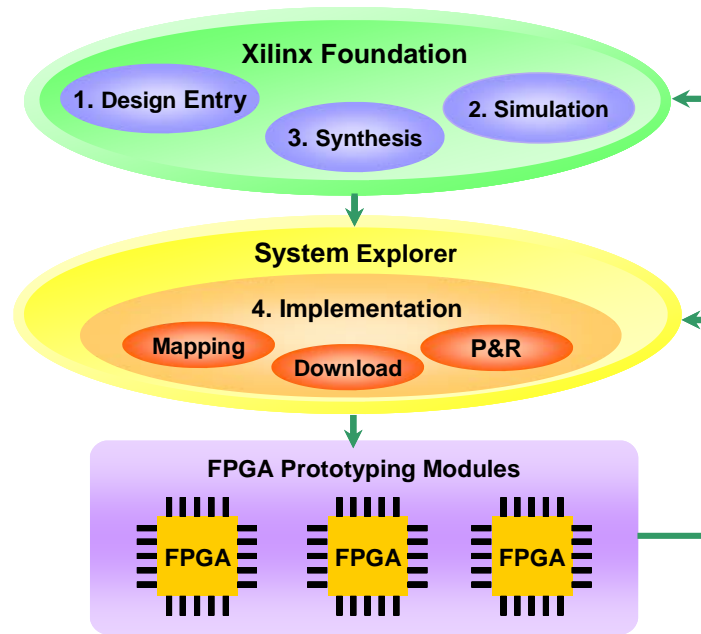


Figure 3.5: FPGA design flow

(2) Synthesis

After completing editing RTL source files, we need to translate them into gate level called netlist files, which only contains information of logic gates and inter-connections. We choose to use Synplify Pro 8.2 for synthesis.

(3) Simulation

Design verification is an important aspect of each project design. Before implementing our circuit in the target device, it is a good idea to simulate and verify the circuit. The most common verifications are functional simulation and timing simulation.

A. Functional Simulation

Functional simulation can be done after the schematic has been entered or a HDL file has been created and synthesized. Functional simulation gives information about the logic operation of the circuit, but it does not provide any information about timing delays.

B. Timing Simulation

The timing simulation will give us detailed information about the time it takes for a signal to pass from one gate to the other (gate delay) and

gives information on the circuit's worst-case conditions. The total delay of a complete circuit will depend on the number of gates the signal sees and on the way the gates have been placed in the FPGA.

One of the most popular simulation tools is ModelSim, which is completely integrated into Xilinx ISE 7.1, and can perform functional simulation and timing simulation very well. Thus, we choose ModelSim SE 5.5e as the simulation tool in our design flow.

(4) Implementation

The implementation is typically done after the design has been verified by functional simulation. The implementation tools will translate the netlist (schematic, HDL), place and route the design in the target device and generate a bitstream that can be downloaded into the device.

(5) Download to Aptix[®] Explorer MP3CF

After the process of implementation, we can download our design into hardware platform. To verify that signals are really working properly in circuit, we can use the LA to debug. Once the result does not match what we expect, we need to come back to modify our design and go through the whole design flow again. That is to say, iterative tests are required until we obtain the results we want.

3.1.3 'C6701 DSP EVM

Digital signal processors, such as TMS320 family of processors, are used in a wide range of applications, from communications and controls to image and speech processing. They are found in cellular phones, fax/modems, disk drivers, radio, and so on. Texas Instrument recently introduced the TM320C6x processor, based on the very-long-instruction-word (VLIW) architecture. This newer architecture supports features that facilitate the development of efficient high-level language compilers. The TMS320C67x DSPs are the floating-point DSP family in the TMS320C6000E DSP platform. We choose TMS320C6701 as our core chip on DSP EVM to implement our

MIMO-OFDM system. Later, we will give an overview of the core chip on DSP EVM. Then we will introduce the architecture of EVM. Finally, we will show the design flow about DSP.

3.1.3.1 TMS320C6701 DSP Overview

'C6701 DSP EVM shown in Figure 3.6 is developed to integrate with other modules on Aptix® platform so that we can come to the realization of MIMO-OFDM system. The EVM is applicable for Aptix® MPx series platform; it uses TMS320C6701 DSP as its core chip. The system clock is 132 MHz, and can be upgraded up to 167 MHz. Owing to having eight functional units in CPU, the DSP can perform 1056 mega floating-point operations per second (MFLOPS).



Figure 3.6: 'C6701 DSP EVM

The architecture of 'C6701 DSP EVM is shown in Figure 3.7, including TMS320C6701 DSP, flash memory, SBSRAM, universal asynchronous receiver/transmitter (UART), joint test action group (JTAG), and other interface circuits like transceiver and complex programmable logic device (CPLD). Later, we will give more information to what have not been mentioned.

(1) Flash Memory:

It is a nonvolatile read-only memory that is electronically erasable and programmable, and it has a capacity of 128 Kbytes. When completing our

development, we can program the design into the flash memory. On the other hand, when we reset the DSP, it will automatically load the design from flash memory into internal program memory.

(2) SBSRAM:

SBSRAM works on the frequency of 132 MHz and has a capacity of 512 Kbytes. There are two working modes determine what it is used for, called Map 0 and Map 1. When Map 0 mode is set, it plays the role of program memory. When Map 1 mode is set, it is taken as general memory.

(3) JTAG and UART:

Both of them are interfaces of data transmission. JTAG is an interface compliant with IEEE 1149.1 standard interface, and it also connects with Innovate Integration Code Hammer PCI interface on PC to load the program from the software, Code Composer Studio (CCS). We can even stop the program and catch the values in memory through JTAG while debugging; UART is the other choice to connect with PC through RS-232 port.

(4) Other Interface Circuit:

CPLD offers four control signals to handle the connection with FPGA or other modules.

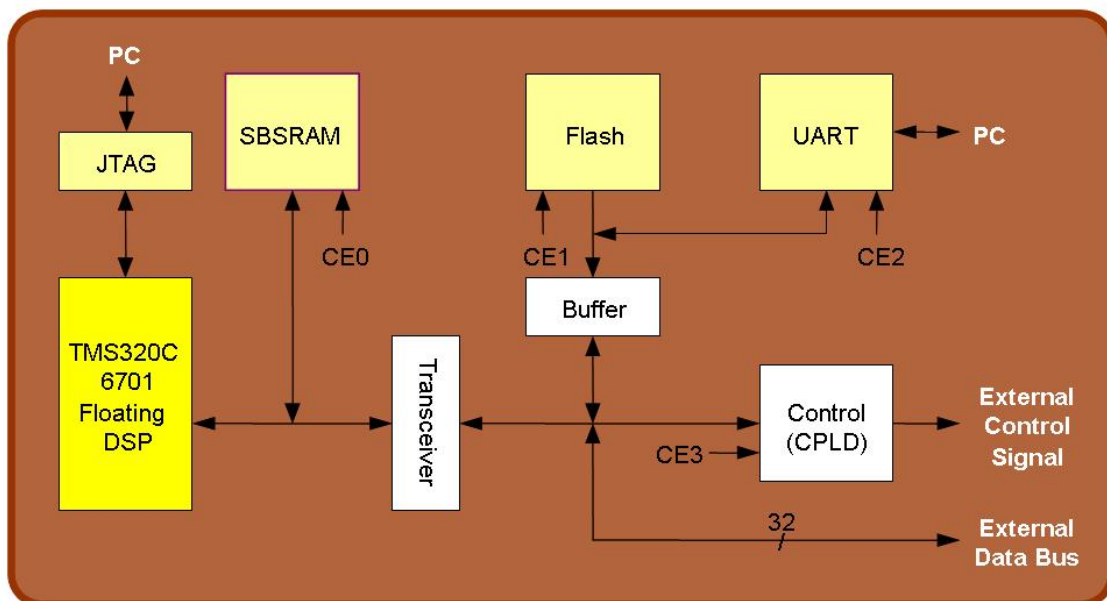


Figure 3.7: Architecture of 'C6701 DSP EVM

3.1.3.2 DSP Design Flow

The Code Composer Studio (CCS) [24] provides an integrated development environment (IDE) to incorporate the software tools. CCS includes tools for code generation, such as a C compiler, an assembler, and a linker. It also has graphical capabilities and supports real-time debugging, which enables us to develop our design efficiently. The DSP design flow with CCS can be separated into the following parts.

(1) Create Project:

First of all, we need to create a project, and add the necessary files for building the project. The most important files are source files, which can either be C source files (.c) or assembly source file (.asm). Then we also require Linker Command File (.cmd) and a run-time support library file (.lib). Last, we may require some header files (.h) to be included.

(2) Code Generation and Options:

Various options are associated with code generation tools, such as C compiler and linker. We can set up Compiler Option and Linker Option to do further configuration if we require, or we can just use the default setting in most cases.

(3) Building and Running the project:

After finishing code generation, we can build and run the project. In this process, it compiles and assembles all C files using *c16x* and assembles the assembly files using *asm6x*. The resulting object files are then linked with run-time library support file using *lnk6x*. This generates an executable file that can be loaded into 'C6701 processor and run. Then, we can load the program after a build.

(4) Monitoring the Watch Window:

Before monitoring the watch window, we need to verify that the processor is still running. After that, monitoring watch window allows us to change the value of a parameter or to monitor a variable we desire. Through monitoring, we can do debugging and regressive test until it works as we expect.

(5) Correcting Program Errors:

Once an error occurs, the error message will be listed and being a link directly to the line where the error occurs. After making the appropriate correction, we have to build, load, and run the program to verify our results.

3.1.4 USB 2.0 Module

USB 2.0 Module uses CYPRESS CY7C68013 [25] as its core chip as shown in Figure 3.8, which includes a 24 MHz 8051 and a 4 Kbytes FIFO. The maximum data rate can be up to 480 Mbps. The FIFO provides the interface between USB 2.0 module and C6701 EVM. Figure 3.9 shows a diagram of the USB 2.0 module and its neighborhood. Through USB 2.0 module, we can transfer data, which comes from PC and will come to USB FIFO first, to DSP EVM. Also, USB 2.0 module can transmit data coming from DSP EVM to PC. We can connect PC with a web camera to generate video stream as our data source.



Figure 3.8: USB 2.0 module

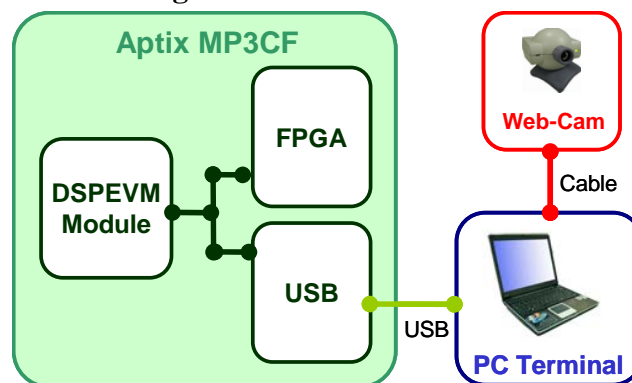


Figure 3.9: USB 2.0 module and its neighborhood

3.1.5 AD and DA Modules

In our MIMO-OFDM system, we use the dedicated AD and DA modules to do the conversion between digital and analog signals as illustrated in Figure 3.10. The major components of each module include eight AD/DA chips, clock source, four databuses, and eight I/O ports, and are described as follows.

1. **AD/DA Chips:** DAS825E and ADC900u are used as core chip respectively.
2. **Clock Source:** It can be setup by the combination of JP1, JP2, and JP10 jumpers.
3. **Databus:** Through the configuration of virtual pins in Aptix[®] Explorer, databus can receive and sent signals from and to FPGA modules by specific cables.

In addition, the output of DA contains eight resistors numbered from $R219$ to $R226$. When DA is connected to AD, we need to use $0.1\ \Omega$ resistors. But if we attempt to connect with the instrument that has $50\ \Omega$ input resistant, we must change resistors to $50\ \Omega$ to avoid the impedance mismatch problem, which will make signals decay.

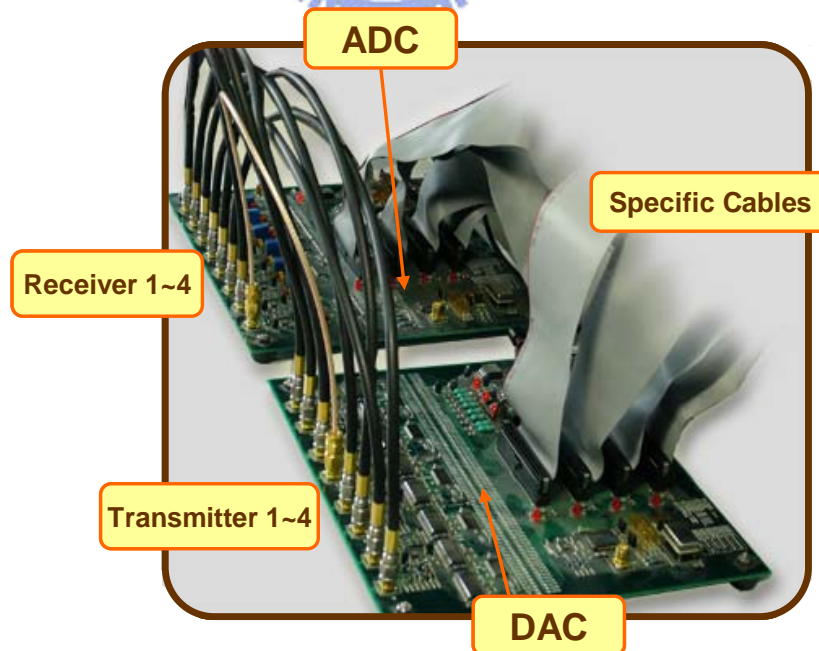
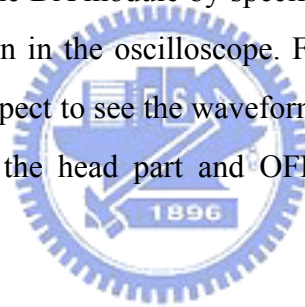


Figure 3.10: AD and DA modules on fast prototyping platform

3.1.6 Debugging Tools

As an old saying goes, “What is a workman without his tools.” In our fast prototyping system, we do have some useful tools for debugging as follows.

1. **Logic Analyzer:** We use Agilent 16702B LA to perform the major task of debugging. There are two modules installed on it. One is 16522A Pattern Generator Module, and the other is 16711A Measurement Module. The former is mainly used for generating desired signals, such as the reset signal or some selection signals for model selection; the latter is used for probing signals in FPGA on Aptix[®] MP3CF platform by connecting specific pods to the slots on Aptix[®].
2. **Oscilloscope:** It is usually used when transmitted signals are prepared by FPGA and sent to the DA module by specific cables. Therefore, we can verify the waveform shown in the oscilloscope. For OFDM signals following IEEE 802.11a, we may expect to see the waveform containing preambles in the form of square wave in the head part and OFDM symbols follow behind those preambles.



3.2 Self-designed Platform

In order to approach a real wireless communication system, the multi-synchronous and high-speed bus FPGA design, combined with our module-based RF, AD/DA, and MAC/BB hardware system, becomes the best solution. Our laboratory has finished and successfully tested RF, AD/DA and MAC/BB boards. The development environment is shown in Figure 3.11, and the close-up shot of main board is shown in Figure 3.12, where four Xilinx Virtex II 6000 FPGAs are mounted in MAC/BB board, and each MAC/BB board is able to connect with at most two AD/DA and two RF modules.

In order to avoid the interference between high speed digital bus, those layouts and interconnections of different modules shall be handled very carefully. Our measurements show that directly connected modules did achieve feasible solution which reduces the risk of facing interconnection problems.

Further analysis and evaluation during development are given in the following sections.

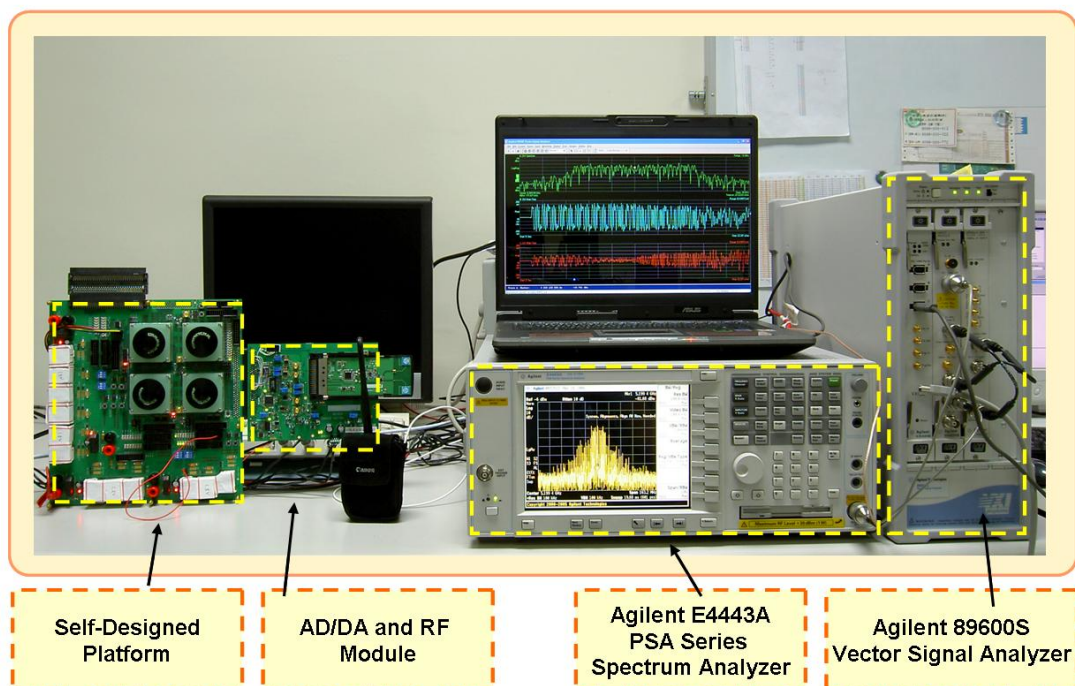


Figure 3.11: Development environment of self-designed platform

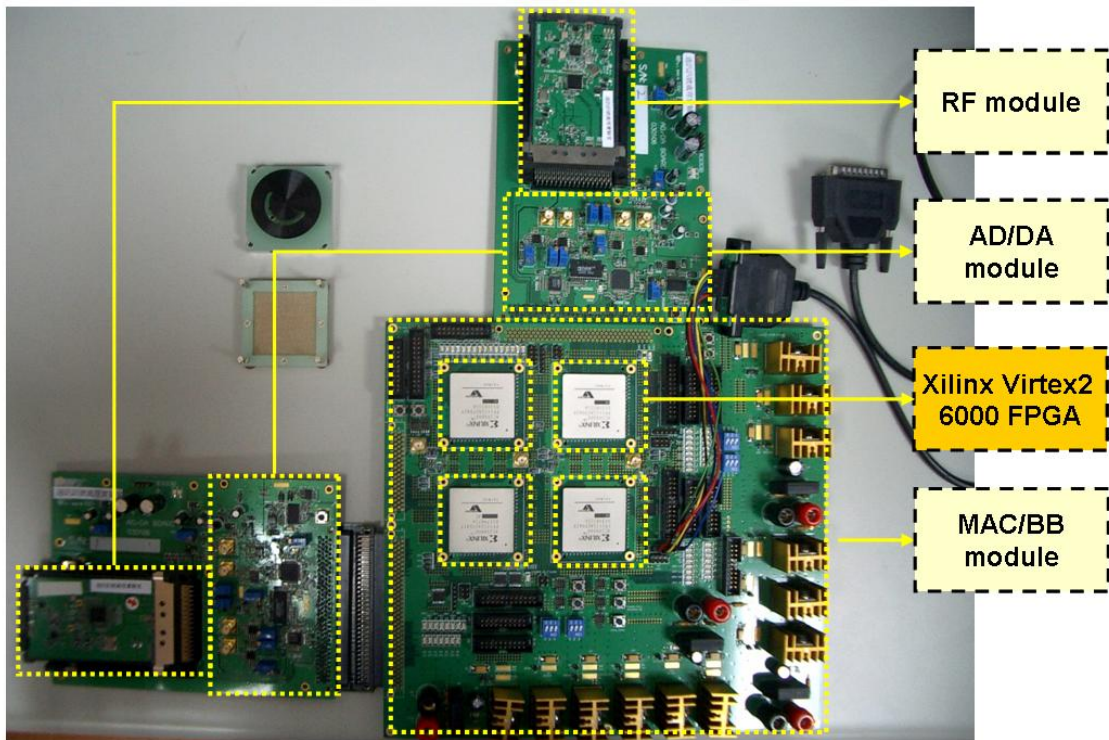


Figure 3.12: Main board of self-designed platform

3.2.1 RF Module

The RF module, as shown in Figure 3.13, consists of MAX2828, which is specifically designed for single-band IEEE 802.11a applications covering world-band frequencies of 4.9 GHz to 5.875 GHz. MAX2828 includes all circuitry required to implement the RF transceiver function, providing a fully integrated receive path, transmit path, voltage-controlled oscillator (VCO), frequency synthesizer, and baseband control interface. Only the RF switches, RF bandpass filters (BPF), RF baluns, and a small number of passive components are required to form the complete RF front-end solution. Because the balance of I/Q signals will impact on the waveform of RF output, the RLC components had been fine tuned. Besides, we also tested the frequency accuracy and power level of transmitted carriers in our interested band from 5.15 GHz to 5.875 GHz. One of those measurements is shown in Figure 3.14; the power level shall be further improved with fine tuning of matching circuits. We used 3-wires (Clock, Data and Latch) to control the RF module from PC currently, and then the control mechanism will be integrated into MAC/BB after verification.



Figure 3.13: RF module on self-designed platform

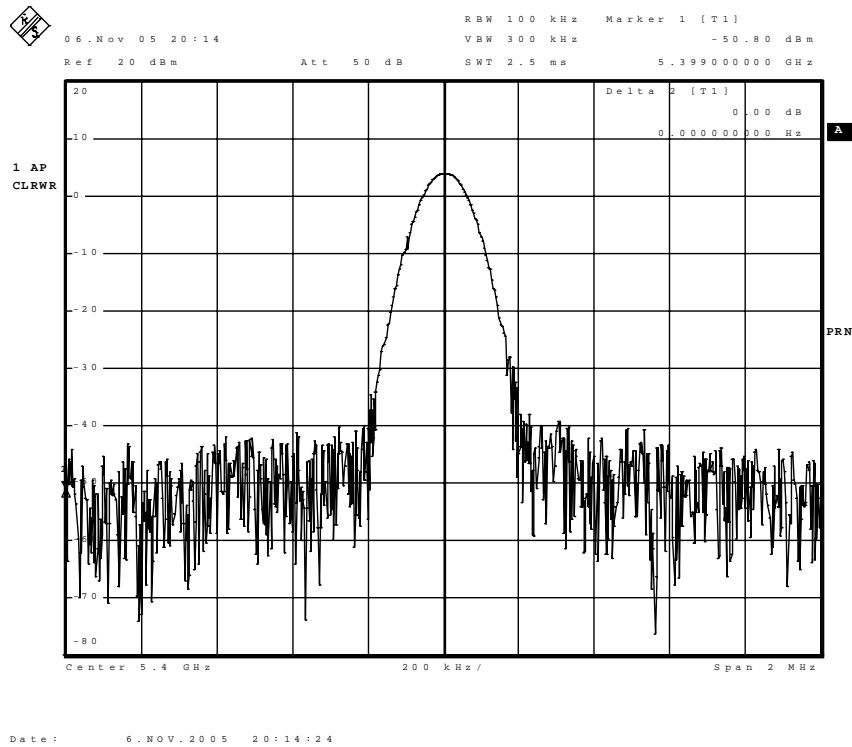


Figure 3.14: Measured carrier spectrum form RF module

3.2.2 AD and DA Modules

The AD/DA module, as shown in Figure 3.15, consists of ADS2807 and DAC2900. ADS2807 is an analog to digital converter which provides a high

bandwidth track-and-hold and gives excellent spurious performance up to and beyond the Nyquist rate. The measured timing diagram is shown in Figure 3.16, which indicates the valid data during the high clock period. In addition, it is recommended that data hold time is 3.5 ns for saving data from bus to SRAM, which had been verified on our AD/DA boards too. DAC2900 is a digital to analog converter which offers exceptional dynamic performance, and enables to generate very-high output frequencies suitable for “Direct IF” applications. It has been optimized for communications applications in which separate I and Q data are processed while maintaining tight offset matching.

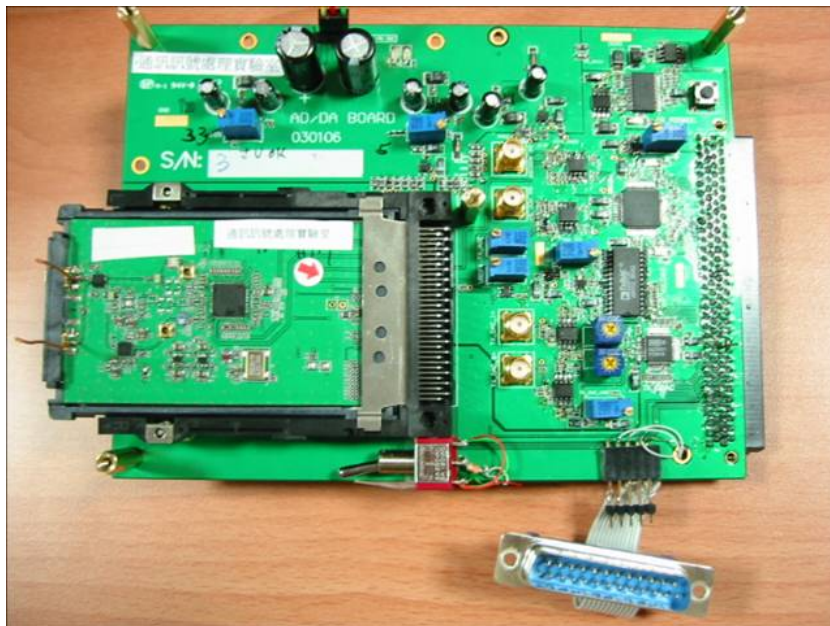


Figure 3.15: AD/DA module on self-designed platform

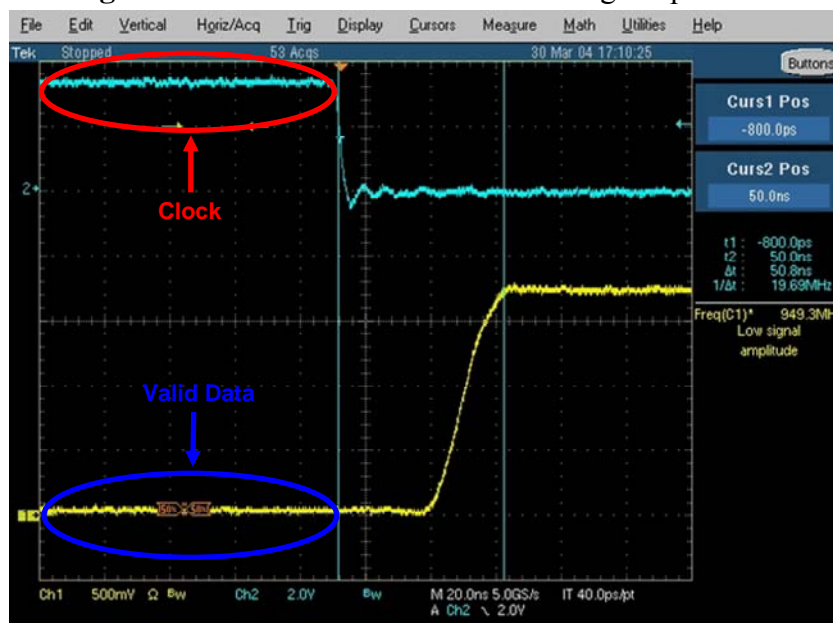


Figure 3.16: Measured data waveform from AD/DA module

3.2.3 MAC/BB Platform

The MAC/BB is an FPGA-based module which is composed of four Xilinx Virtex-II 6000 modules, as shown in Figure 3.17. It outperforms conventional DSP processors on a board-for-board comparison, resulting in significant improvements in processing speed, size, weight, power, and costs. Combining a wide variety of flexible features and a large range of densities up to 6 million system gates, the Virtex-II 6000 enhances programmable logic design capabilities and is a powerful alternative to mask-programmed gates arrays. With its advantages of very fast data rate, it can achieve full duplex and real time operating for wireless communication. The VHDL codes had been used to drive LEDs by differential clock rate from oscillator to verify its functionality.

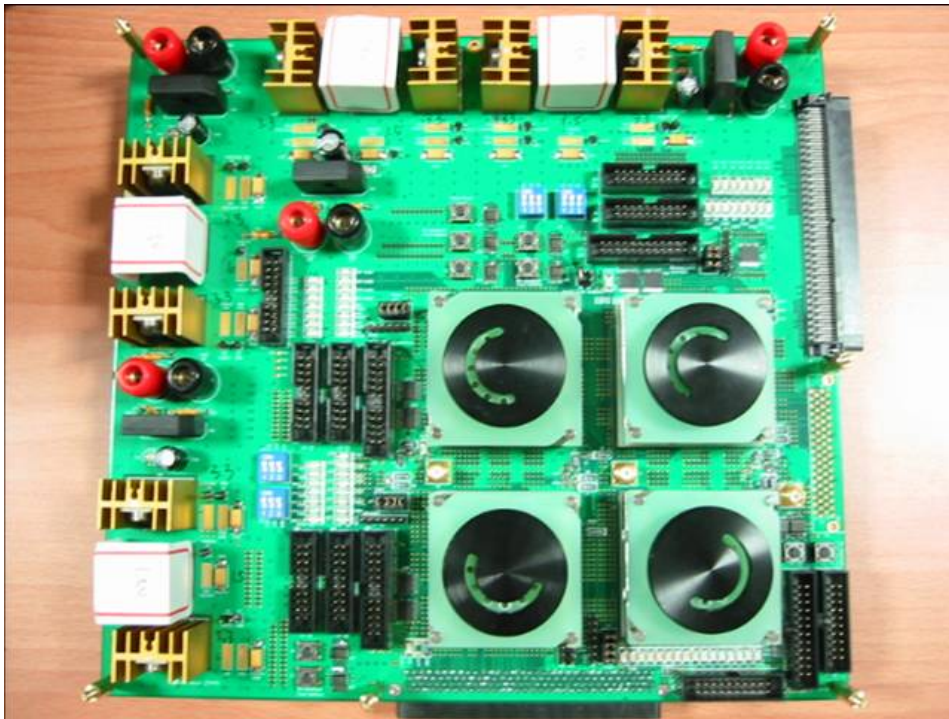


Figure 3.17: MAC/BB platform

3.2.4 USB Interface

In order to have a convenient input for the audio/video signal in the future, USB interface was designed into the platform, which is shown in Figure 3.18. It will comply

with the USB specification revision 1.1, and be upgraded to USB 2.0 if necessary. The compatibility test is conducted with compliance software run at PC equipped with PCI to UTMI compliant interface card during test stage. This will make sure we can easily connect our platform with any signal source with USB port. The built-in USB interface codes for FPGA was defined and implemented.

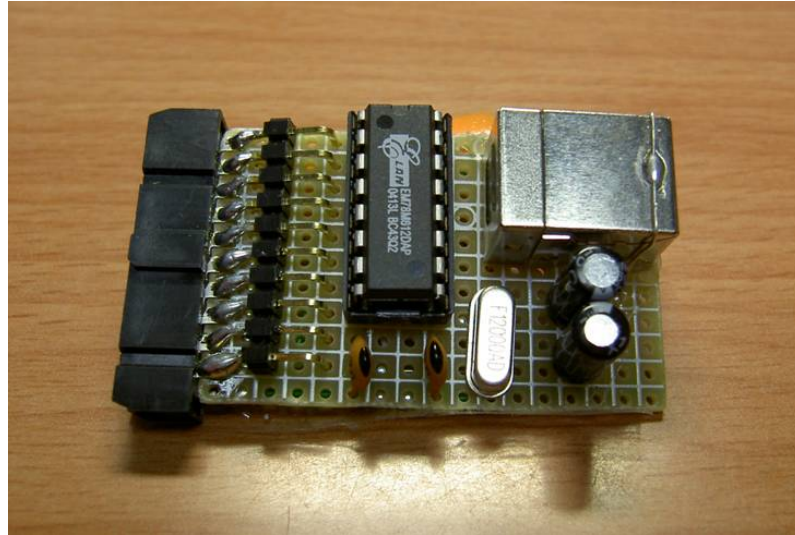


Figure 3.18: USB module on self-designed platform

3.2.5 Debugging Tools

Besides the logic analyzer and oscilloscope mentioned before, two additional instruments, spectrum analyzer and vector signal analyzer, are adopted to capture and analyze RF signals.

1. **Spectrum Analyzer:** Agilent PSA Series Spectrum Analyzer E4443A is chosen. It offers high-performance spectrum analysis up to 6.7 GHz and beyond with swept-tuned measurements with digital Resolution-BandWidths (RBW) filters. In our debugging flow, E4443A capture the transmitted 5.2GHz signals, down convert them to 70MHz intermediate frequency (IF), and then fed out to vector signal analyzer to perform advanced analysis. Its block diagram is shown in Figure 3.19.
2. **Vector Signal Analyzer:** Instead of swept-tuned measurements, vector signal analyzer 89600S performs fast Fourier transform (FFT) measurements with

digital FFT filters, which can measure all signal characteristics (i.e. phase) and avoid very long sweeps times required for narrow RBW. Figure 3.20 shows the block diagram of vector signal analyzer, notice that it is PC-based and therefore machines only capture the RF signal accurately and feeds to PC, where final analysis are performed on PC.

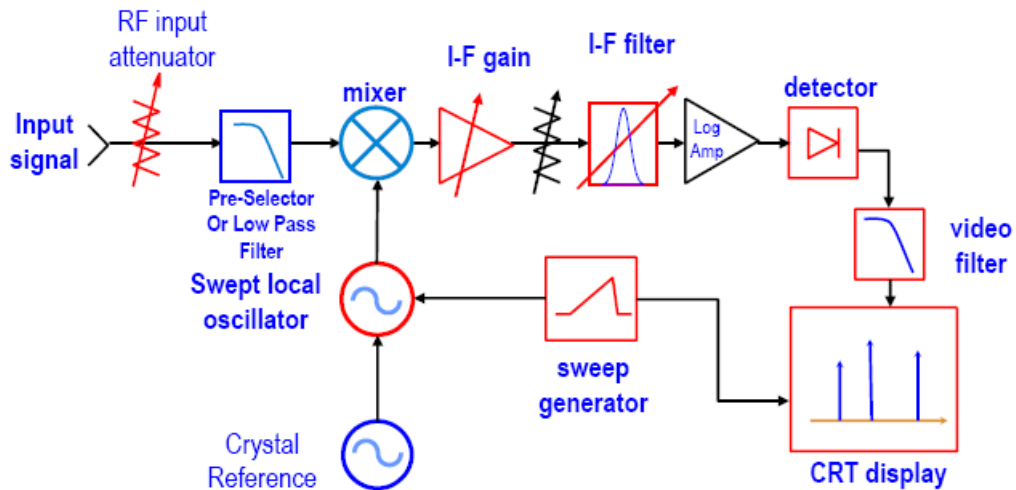


Figure 3.19: Spectrum analyzer block diagram

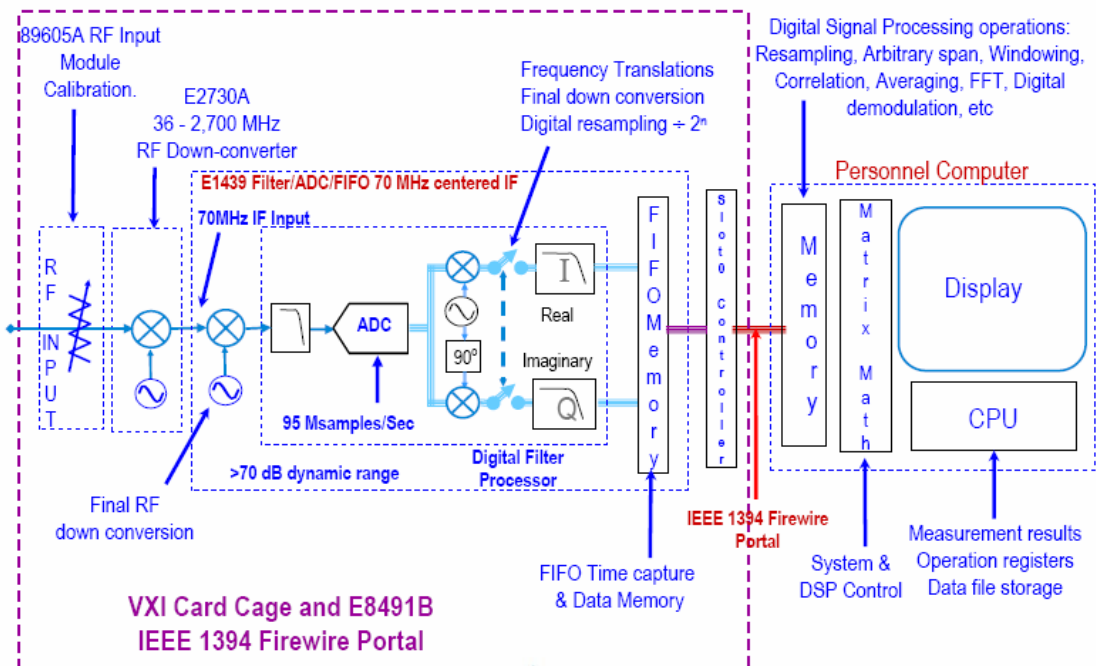


Figure 3.20: Vector signal analyzer block diagram

3.3 Summary

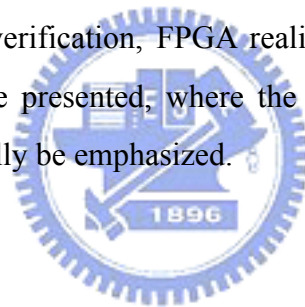
In this chapter, we introduce two adopted platforms, e.g., fast prototyping platform and self-designed platform. These two platforms both are equipped with FPGA, USB, and AD/DA modules; moreover the self-designed platform provides RF modules by which realistic wireless channel characteristics can be generated. Finally, corresponding debugging tools are mentioned; in particular the logic analyzer and oscilloscope are used to measure baseband signals, and spectrum analyzer and vector signal analyzer are used to capture and analyze RF signals.



Chapter 4

MIMO-OFDM System Realization

The 5 GHz MIMO-OFDM system is implemented on the FPGA-based hardware introduced in Chapter 3. It demonstrates both diversity and multiplexing schemes that use MIMO technique in conjunction with OFDM. In this chapter, a complete design flow including MATLAB verification, FPGA realization, ModelSim simulation, and experimental results will be presented, where the principles and concepts of circuit design on FPGA will specially be emphasized.



4.1 Design Flow

Digital Signal Processing (DSP) design has traditionally been divided into two types of activities — systems/algorithm development and hardware/software implementation. The majority of DSP system designers and algorithm developers use the MATLAB language for prototyping their DSP algorithm. Hardware designers take the specifications created by the DSP engineers and create a physical implementation of the DSP design by creating a register transfer level (RTL) model in a hardware description language (HDL) such as VHDL and Verilog. Our MIMO-OFDM system can be regarded as a DSP system, and Figure 4.1 shows the design flow we adopt.

First, we have to program a floating-point MATLAB code in order to not only verify the algorithms mentioned in Chapter 2 but to evaluate the system performance. Then, the floating-point MATLAB code is required to be manually converted into a fixed-point MATLAB code. Subsequently, RTL model is established, where we

choose VHDL as our hardware description language and Xilinx ISE 6.1 as our development tool. Next, this RTL implementation is simulated by ModelSim SE 5.5e and synthesized onto a netlist of gates using Synplify Pro 8.2. Finally, the netlist of gates is placed and routed onto Xilinx FPGAs using Xilinx ISE 6.1. The detailed design flow will be discussed in the following sections

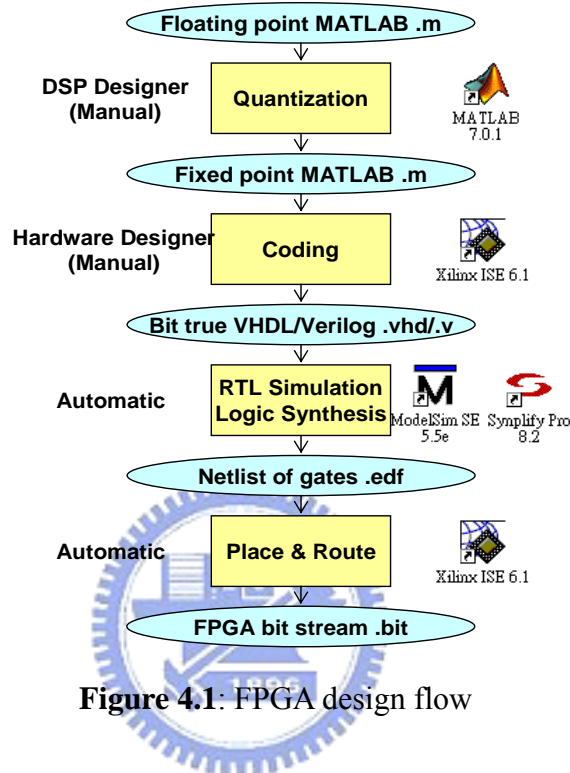


Figure 4.1: FPGA design flow

4.2 MATLAB Verification

As developing a communication system, MATLAB is one of the best candidates for us to model and simulate the system by means of its powerful matrix computation ability and well-defined communication functions. In addition, its 2D or 3D graphic interface also makes designers easily illustrate the system performance with the effects of simulated channel and quantization error and so on. In this section, both floating-point and fixed-point verifications will be mentioned. In the floating-point verification stage, we attempt to verify the accuracy of communication algorithms and sketch out the performance of the system, which is considered as the basis in comparison with fixed-point cases. On the other hand, in the fixed-point verification stage, we need to establish a fixed-point system model by using a quantization algorithm first, and then perform advanced fixed-point analysis.

4.2.1 Floating-Point Verification

In this section, the function blocks and adopted algorithms mentioned in Chapter 2 will be verified first, and then the whole system will be constructed and the system performance will be expressed.

1. RRC:

In our system, a 25-tap root raised cosine filter with roll off factor $\beta=0.22$ is designed, and its impulse response and frequency response is shown in Figure 4.2. It can be clearly observed in the frequency response that signals with frequency higher than approximately 12 MHz are filtered (the sample rate equals 40 MHz), that is, the waveform in time domain will become much smoother, and therefore can effectively combat the aliasing in AD/DA conversion and the ISI problem. Figure 4.3 shows the waveforms before and after RRC pulse shaping. Waveform in part (a) is a series of BPSK modulated signals. After the pulse shaping in transmitter side RRC, the smoother waveform will look like part (b). Next the waveform passing through RRC in the receiver side is shown in part (c). Finally, the eye diagram after RRC shaping is illustrated in Figure 4.4.

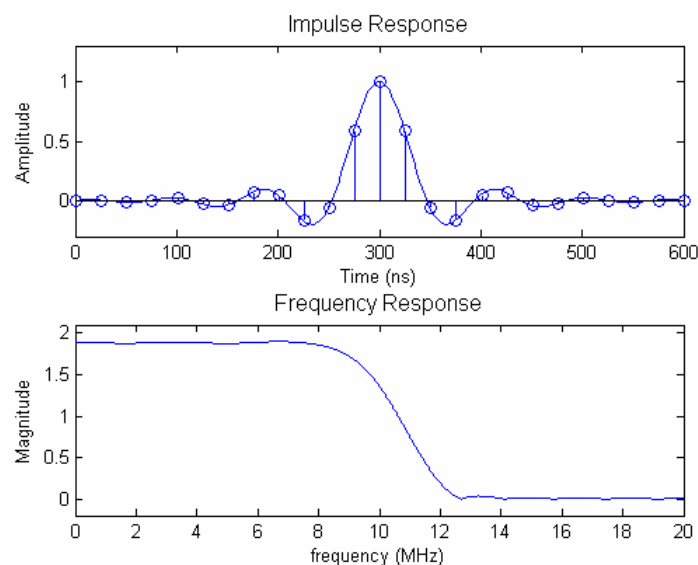


Figure 4.2: Impulse and frequency response of RRC filter with $\beta=0.22$

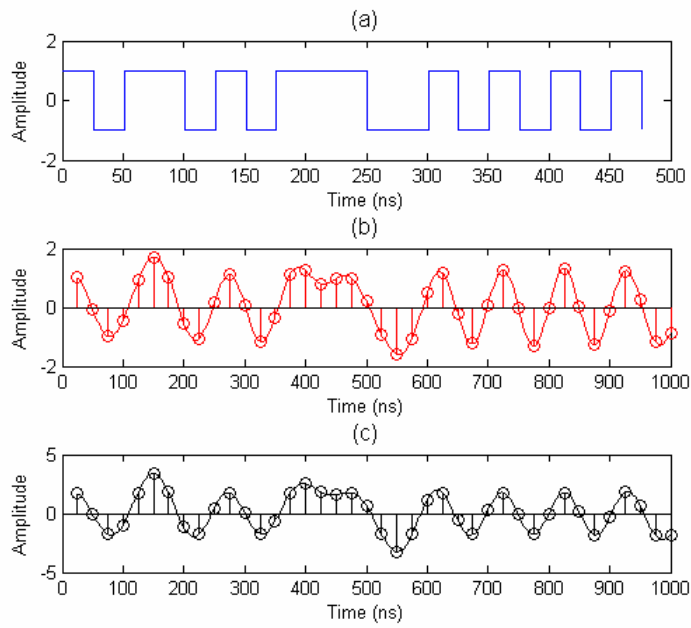


Figure 4.3: (a) Original waveform (b) RRC shaped waveform on transmitter
(c) RRC shaped waveform on receiver

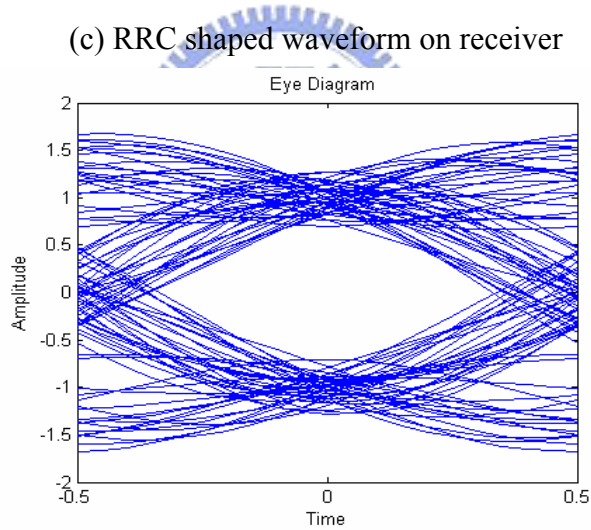


Figure 4.4: Eye diagram of RRC shaped waveform

2. Timing synchronization:

As we have mentioned in Section 2.3.1, a data-aided timing synchronization algorithm is adopted in our system. Here we pass our transmitted signal through a Rayleigh fading, multipath channel, and then process the post-RRC received signal by timing synchronization block. The output waveform is shown in Figure 4.5. It can be observed that there is a main hill, whose summit is our reference frame start position. Because of multipath

effect, there are also many smaller hills hidden inside the main hill; however they will all be neglected because our algorithm will only choose the most reliable, e.g., the most highest peak, as our reference start position.

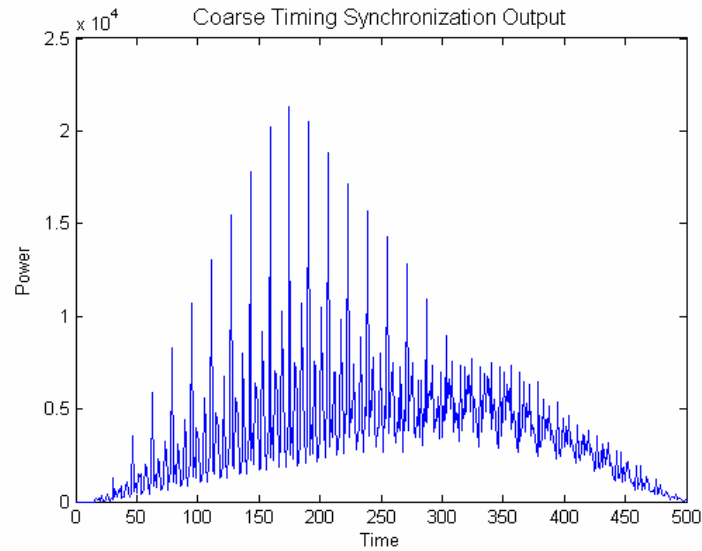


Figure 4.5: Coarse timing synchronization output

3. Channel estimation:

The channel estimation result is shown in Figure 4.6, where the above one is the real channel frequency response, and another one is the estimated channel frequency response. We can see these two curves are almost the same except the amplitude of the below one is one-time bigger than another, which is caused by the space-time structure in long preamble.

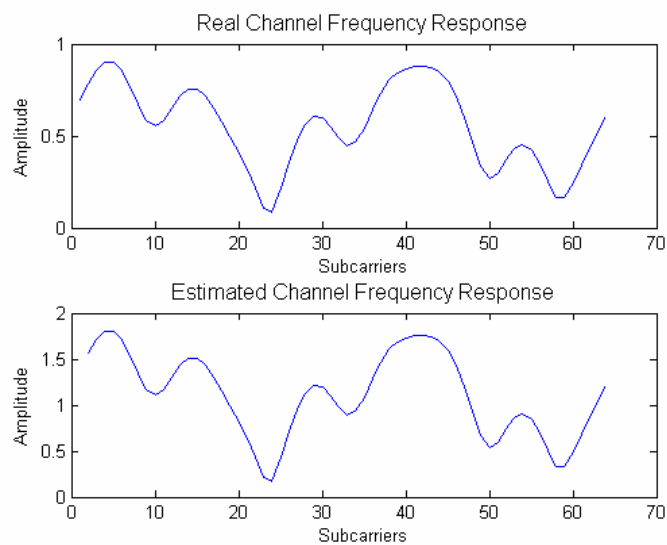


Figure 4.6: Real and estimated channel frequency response

4. System performance:

The floating-point BER to SNR system performance is shown in Figure 4.7, where a Rayleigh fading channel with AWGN noise is generated, and the total path number is four, including one main path and three multipaths. We can see that under the same SNR, the BER in the STBC case is much smaller than the BER in the VBLAST case, and the gap becomes bigger and bigger as SNR increases.

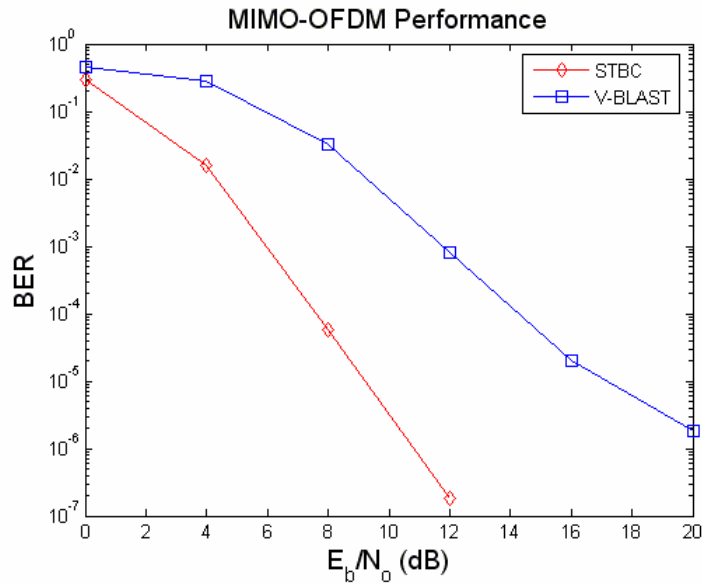


Figure 4.7: Floating-point system performance

4.2.2 Fixed-Point Verification

Before fixed-point verification, we need to convert the floating-point MATLAB code to the fixed-point MATLAB code by the quantization algorithm first. Since a quantization procedure is very complicated and time-consuming due to its nonlinear characteristics, we leave the detailed descriptions of this part to Chapter 5. In this chapter we will only show the results available in the quantization algorithm.

Table 4.1 shows the result word lengths available in the quantization algorithm, where EM means error metrics, which can represent the level of the quantization error. The bigger the EM is, the more serious quantization error we get. Otherwise, R means the required hardware resources; m and p indicate the integer and fractional lengths. The detailed definitions of these parameters will be detailed explained in Chapter 5.

Based on the word lengths shown in Table 4.1, we then construct the fixed-point system model on MATLAB, and perform the fixed-point verification. Figure 4.8 shows a fixed-point BER to SNR system performance. We observe that no matter in the STBC or VBLAST case, the curves drift to right-upper side when the number of EM increases, which indicates the system performs worse as the level of quantization error increases. Furthermore, we also observe that in low SNR, channel noise dominates the system performance, therefore the differences between EMs are not obvious. On the other hand, in high SNR, the quantization error noise dominates the system performance, therefore the gap between different EMs becomes bigger and bigger.

Table 4.1: Word lengths under different EMs

EM_{target}	1	5	10
EM_{fine_STBC}	0.9402	4.1791	8.756
EM_{fine_VBLAST}	0.9622	4.496	9.626
$m_{ifft} + p_{ifft}$	10	7	6
$m_{RRC} + p_{RRC}$	13	12	9
$m_{rxRRC} + p_{rxRRC}$	11	8	7
$m_{fft} + p_{fft}$	14	11	11
$m_{lnfft} + p_{lnfft}$	10	8	8
$m_{ch} + p_{ch}$	12	9	7
$m_{ph} + p_{ph}$	12	8	8
R	13808	10338	9268

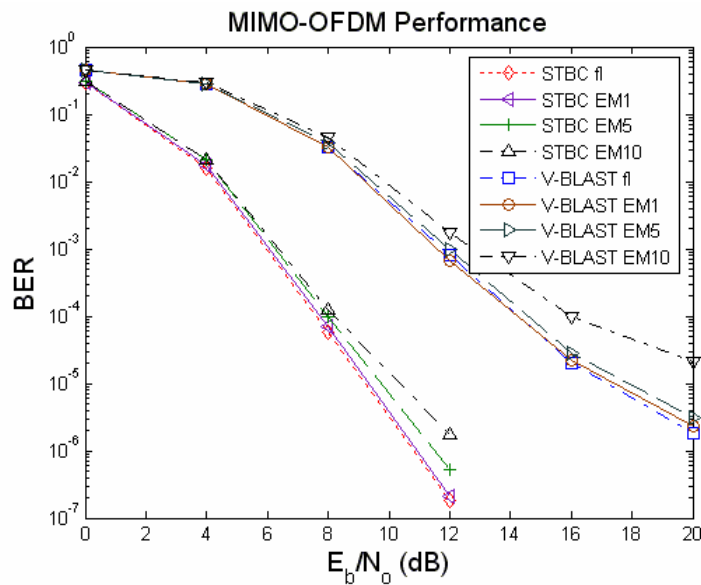


Figure 4.8: Fixed-point system performance

4.3 FPGA Realization

With the introduction of advanced Field-Programmable Gate Array (FPGA) architectures which provide built-in DSP support such as embedded multipliers and block RAMs on the Xilinx Virtex-II and the multiply-accumulators, DSP Blocks, and MegaRAMs on the Altera Stratix, a new hardware alternative is available for designers who can get even higher levels of performances than those achievable on general purpose DSP processors.

In our implementation, we adopt Xilinx Virtex-II series as our FPPA and VHDL as our hardware description language. The programming concepts that deserved to be mentioned in high level language like MATLAB and in hardware description language like VHDL are quite different. In general, high level language keeps its temporary data in a form of variables, and simply assigns the stored variable to another one which is used to be the input of next stage or functions if necessary, whereas hardware description language may need extra data buffer and related components to perform the same task. Since we have no choice but to add RAMs, register, as data buffers, some index-related jobs can be performed in the same time, such as zero padding, bit reversing, or adding cyclic prefix and so forth. The following sections will give readers more concepts and clear description about how we design in FPGA.

4.3.1 Design Principles

Before going through the circuit design of our components in the MIMO-OFDM system, some important design principles need to be mentioned first.

1. **Parallel processing:**

Since FPGA has a highly flexible architecture, and can support any level of parallelism, we can significantly enhance the data throughput by parallel processing. Taking a generally used function, FIR filter, for example, an FIR filter consists of many multiply-and-accumulate operations, and it will be time-consuming if only a single, fixed multiply-and-accumulate unit is used. Hence sufficient multiply-and-accumulate units are always realized.

The STBC encoder in the transmitter is another good example of parallel processing, since data of two successive symbols are required to do STBC encoding, we then can transfer them simultaneously from the previous block to the STBC encoding block. By doing so, we don't need to buffer the first symbol and start to do STBC encoding until another symbol serially arrives.

2. **Avoid critical path problem:**

Critical path problem is often faced by designers. When the delay of a circuit is determined by the delay of its longest sensitizable paths (such paths are called critical paths), the problem of dealing with the delay of a circuit is called critical path problem. Sometimes a complicated numerical computation is carried out in a block, and thus many summations and multiplications are serially executed in a path within a single clock period. Although high speed multiply-accumulators are embedded inside Xilinx FPGAs, these operations cannot be completely executed in time within a single clock period; therefore error results will be fed out. In order to avoid this kind of problem, registers will be inserted in the path and therefore the whole computation will be separated into few sections and can be executed within few clock periods depending on how many registers are inserted.

3. **Reuse RAMs or ROMs:**

In our design, RAMs and ROMs are widely used, and most of FPGA resources are occupied by them. Under this circumstance, to save or reuse RAMs and ROMs becomes the most crucial key to save total circuit area. Therefore in some proposed circuit blocks, alternatively reusing of few small-sized RAMs is adopted instead of only one-time using a single huge-sized RAM.

4. **Substitute real number computation for complex number computation:**

Inevitably, large amount of complex number computations are included in our MIMO-OFDM system. In MATLAB, these complex number computations can be easily computed, whereas become inconvenient in VHDL since complex number operations cannot be carried out directly in VHDL. Hence, in

order to deal with complex number computations, the original complex number arithmetic is separated into many real number segments. For example, one simple complex number computation, $(X+Y)/Z$, will become $(ae+ce+bf+df)/(e^2+f^2)+(-af-cf+be+de)i/(e^2+f^2)$ after the rearrangement, where $X=a+bi$, $Y=c+di$, $Z=e+fi$, and $a, b, c, d, e,$ and f are real numbers.

4.3.2 Circuit Design

In the following paragraphs, components are roughly divided into transmitter components and receiver components, and all circuits follow the principles introduced in the previous section. Additionally, every component is hierarchically designed.

4.3.2.1 Circuit Design of Transmitter

Figure 4.9 shows the overview of the circuit design of the transmitter. All circuits are synchronous, and a pipelined architecture is adopted where the clock is used to control the data transfer simultaneously. All delay blocks make use of components SRL16 to implement a progressive delay line, where SRL16 is an exclusive feature of Virtex architecture that allows users to save a lot of room and increase tremendously the performance. Detailed circuit designs of function blocks are described as follows.

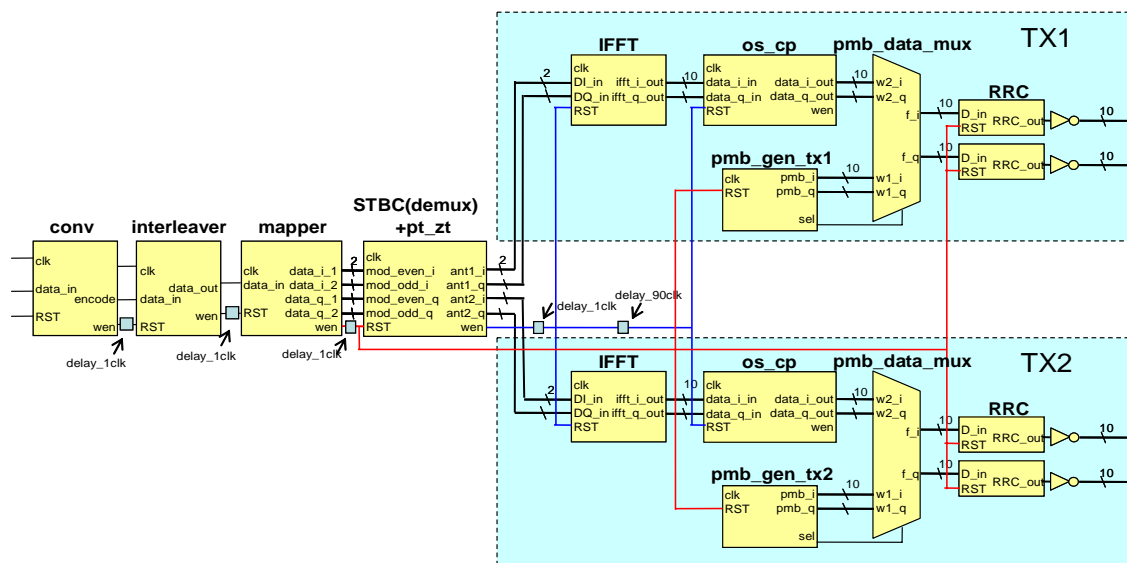


Figure 4.9: Circuit design of transmitter

(1) **Convolutional encoder:**

Figure 4.10 shows our circuit design of the convolutional encoder, named **conv**. First the source data are fed into an encoder, **conv_encoder**, which encoding rule follows what has been discussed in Section 2.2.1. Then three parallelly generated bits *da*, *db*, and *dc* are passed into the next circuit – **conv_ram_ctrl**. The main function of **conv_ram_ctrl** is to generate control signals, such as write enable signal (*wen*), and writing or reading addresses (*address_1-3*). Besides control signals, *da*, *db*, and *dc* are buffered for one clock period for the sake of the synchronization of data and control signals. Finally, a particular RAM is used, where it allows three input signals simultaneously to be written into three different allocations according to respective addresses in writing stage, and output only one signal in reading stage. RAM size depends on which MIMO technique is adopted in the system. When spatial diversity scheme is chosen, it requires at least $48 \times 6 \times 3 = 576$ memories to store, therefore a 1024×1 RAM is realized. On the other hand, a 2048×1 RAM is chosen in spatial multiplexing scheme. In addition, the write enable signal *wen* is fed out so that the next stage can know when to read by recognizing *wen* changing from high to low (from writing mode to reading mode).

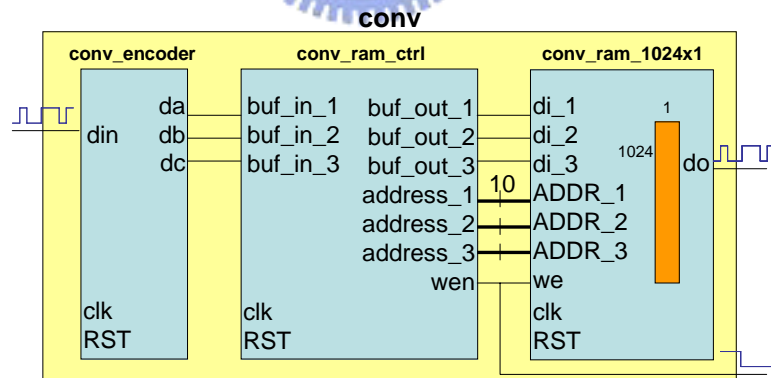


Figure 4.10: Circuit design of convolutional encoder

(2) **Interleaver / de-interleaver:**

The circuit designs of the interleaver and de-interleaver are similar; therefore here we just explain the design concept of the interleaver. As we mentioned in Section 2.2.2, a block interleaving scheme is adopted. Hence the

interleaving will take place repeatedly in every symbol, which contains 96 bits through totally 576 bits (take the signal length of the spatial diversity scheme for example). By this reason, there are only two 128×1 RAM instead of a 1024×1 RAM embedded inside, which can dramatically save FPGA resources. **Inter_ram_ctrl** will generate a writing address with a interleaved order, and simultaneously generate sequential reading address. Then, by scheduling of multiplexer and de-multiplexer, the two 128×1 RAMs will alternately interleaved store and then sequentially fed out the data from one RAM to another. In this way, the input signals are successfully interleaved.

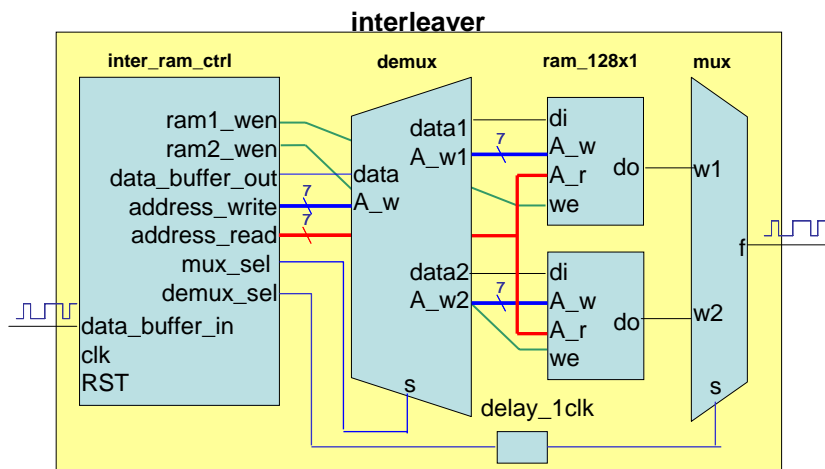


Figure 4.11: Circuit design of interleaver

(3) Mapper / de-mapper:

In our design, the de-mapper block is integrated with another block, which will be discussed in the following paragraphs. Here we see the circuit design of mapper first. In mapping stage, two successive input bits are going to be modulated into in-phase and quadrature parts, hence a adopted QPSK modulation scheme is carried out in **mapper_ram_ctrl**, and then the modulated signals, including I part and Q part, are fed into adopted 512×2 RAMs along with control signals. Here word lengths are chosen to be 2 in order to represent +1 and -1 in 2's complement method. For the sake of convenience in the next stage, the successive mapped data symbols are fed out simultaneously by arranging the order of reading addresses deliberately.

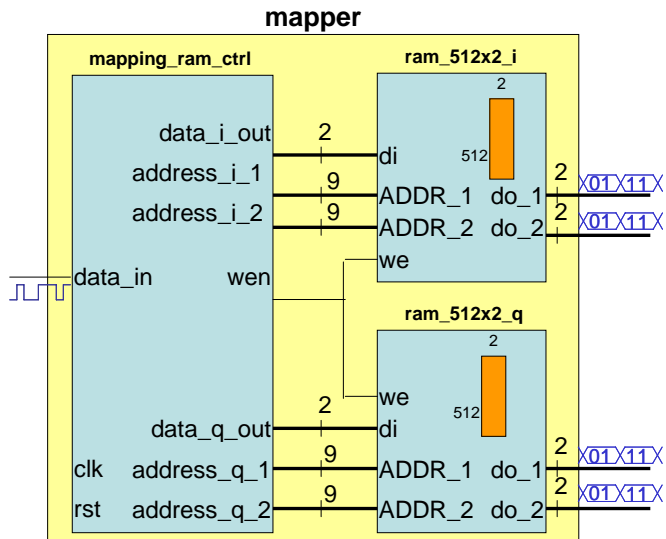


Figure 4.12: Circuit design of mapper

(4) STBC encoder with pilot zero tones adder:

This circuit is suitable for the spatial diversity scheme, and the structure is very similar to what we have introduced in previous paragraphs. After feeding in successive mapped data symbols, the STBC scheme is executed in **stc_ram_ctrl** and then separate data into two streams, which are respectively stored into four 512x2 RAMs. These RAMs are designed to allow two signals written in the writing stage and one signal read in the reading stage. Moreover pilot tones and zero tones are also imbedded into the output signals of this stage.

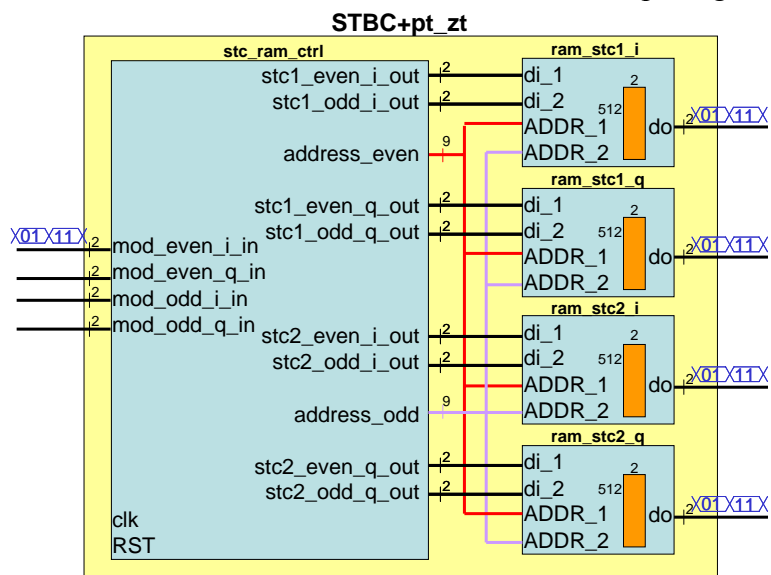


Figure 4.13: Circuit design of STBC encoder with pilot zero tones adder

(5) De-multiplexer with pilot zero tones adder:

Opposite to the previous circuit, this circuit is designed for the spatial multiplexing strategy. Since it just separates the input signal into two independent streams without advanced space-time coding, the circuit design is much simpler than the previous circuit.

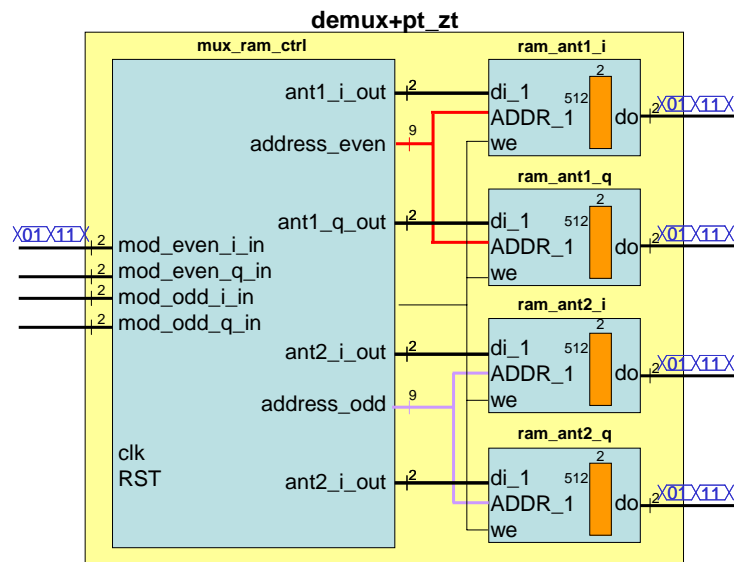


Figure 4.14: Circuit design of de-multiplexer with pilot zero tones adder

(6) FFT / IFFT:

Fast Fourier transform (FFT) is a type of discrete Fourier transform (DFT), but only faster with fewer computations (summations and multiplications). A DFT takes N^2 computations to calculate a transform for N points, whereas the FFT takes around $N \log_2 N$ computations to complete the same thing. Here we adopt a 64-tap FFT which is provided by Xilinx and can operate 20-bit complex (20-bit real, 20-bit imaginary) samples, and a rough design concept is illustrated in Figure 4.15.

A pipelined implementation of a 64-point FFT requires a simple pipeline consisting of 6 butterfly computation modules. This method operates on two data points per clock cycle, yielding an effective data rate that is twice the clock rate, but requires customized butterfly computation modules for each stage of the FFT computation. Since a butterfly computation is carried out, the output signal will be in bit-reverse order.

Finally, according to the results available in the quantization algorithm, the word length of IFFT (FFT) outputs should be truncated from 20 bits to 10 bits (14 bits) Furthermore, an additional saturation circuit is attached behind IFFT to mitigate PAPR problem.

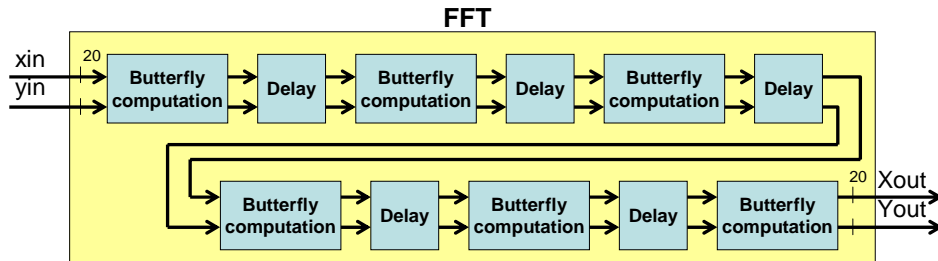


Figure 4.15: Circuit design of fast Fourier transform

(7) Oversampler and cyclic prefix adder:

Signals must be not only oversampled but also cyclic prefix added in this circuit. Moreover, the post-IFFT bit-reverse ordered signal need to be sorted to be sequential order. We achieve these three purposes by merely arranging the order of address when data is written into and read out from 1024×10 RAMs. The arrangement of address order in writing mode will not only sort signal to sequential order but lead to zero padding between every signal, also called oversampling; whereas the arrangement in reading mode will let output signal look like that a copy of the last 1/4 part of OFDM symbol is attached to the front of itself, that is, cyclic prefix.

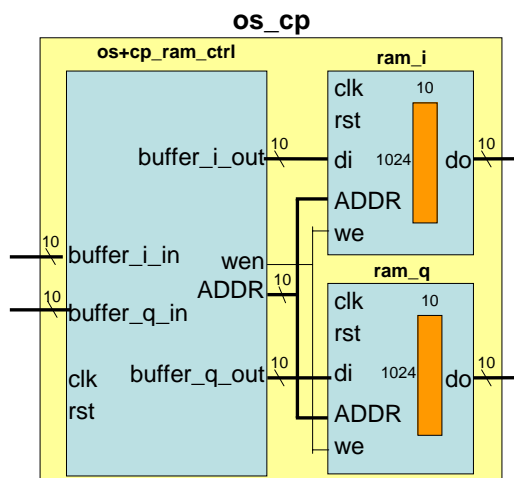


Figure 4.16: Circuit design of oversampler and CP adder

(8) Preamble generator:

As we described in Chapter 2, preambles are BPSK modulated and consist of ten short preambles and two long preambles. In our design, one oversampled short preamble and one oversampled long preamble are stored in 256x2 ROMs. We repeatedly read out the short preamble ten times and the long preamble two times, then total series of data form a complete preamble channel. Since preambles are also STBC encoded, a **pmb_gen_stbc** block is equipped in first antenna preamble generator.

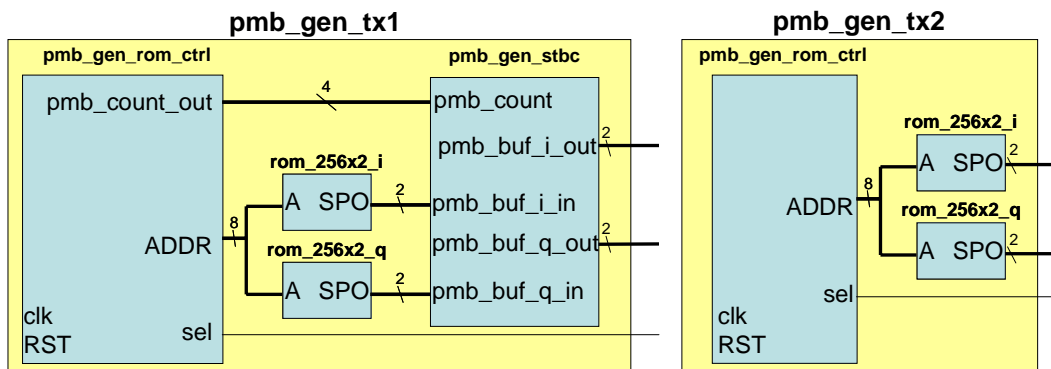


Figure 4.17: Circuit design of first and second preamble stream generators

(9) Root raised cosine filter:

Figure 4.18 shows the circuit design of the 25-tap RRC filter, and we can see that all 25 multiply-and-accumulate operations are executed in one clock cycle; such parallel processing can maximize data throughput. Furthermore, basing on the results available in the quantization algorithm, coefficients embedded inside are truncated to 13 bits and the output word lengths in the transmitter RRC and receiver RRC are also truncated to be 10 bits and 11 bits.

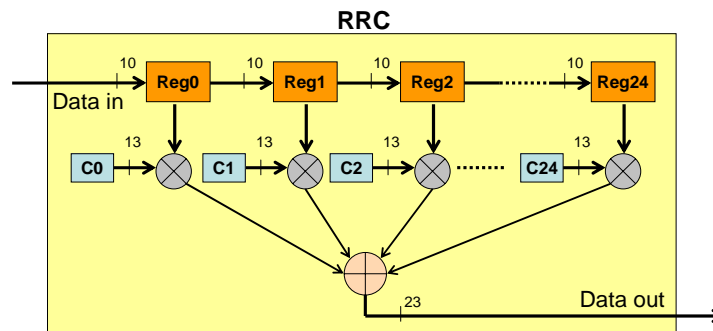


Figure 4.18: Circuit design of RRC filter

4.3.2.2 Circuit Design of Receiver

Figure 4.19 shows the overview of the circuit design in the receiver. Certainly, a pipelined architecture is adopted and more delay blocks are adopted than in transmitter. The circuit designs of function blocks are given as follows.

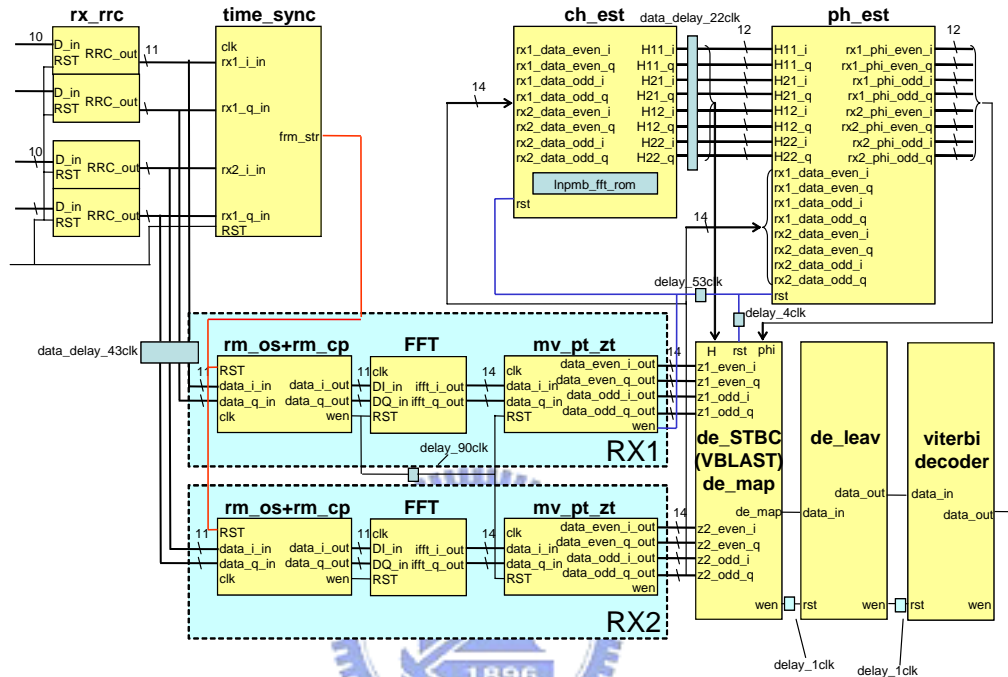


Figure 4.19: Circuit design of receiver side

(1) Timing synchronizer:

The timing synchronization block diagram is shown in Figure 4.20. First, in order to do two-time downsampling, the **switch** block is designed to generate two times period clock sources. After that, the following matched filter blocks (**sp_match**) are designed to match short preambles. After every matched filter successfully matches short preambles and generates ten impulses, the successive series of comparators (**s_comp**) are processed to find out the maximum absolute value among 8 paths, e.g., to find out the most reliable reference. Next, this maximum sequence is delayed by 16 clocks and sum up to enhance the peak values by the **delay_sum** block. Finally, an FIR filter (**FIR**) with response of some repeated $\{0,0,0,\dots,0,1\}$ is applied to rake the values of

each impulse, and then generates a hill-like output waveform, where the time index of the summit of this hill can be regarded as a start time of the packet. The circuit design of the matched filter and the FIR filter is similar to the RRC filter, where a maximum data throughput is achieved by parallel processing.

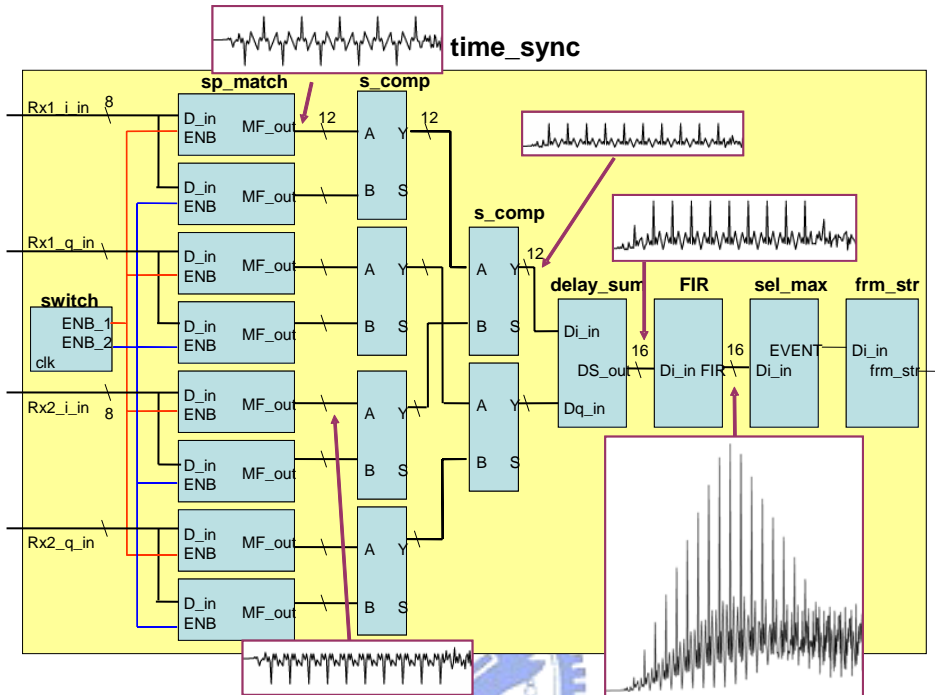


Figure 4.20: Circuit design of timing synchronizer

(2) Oversample and cyclic prefix remover:

Based on the timing information available in the **time_sync** block, oversample and CP are removed by arranging the address order in this block.

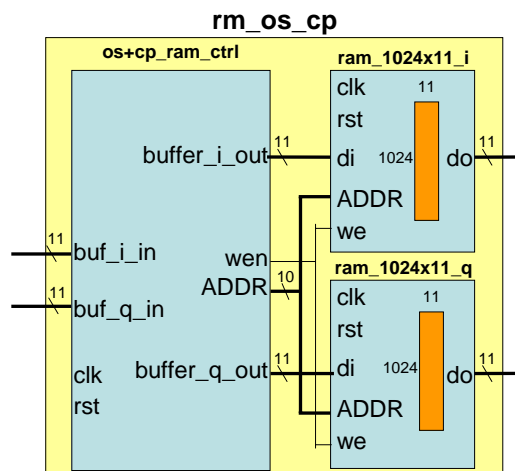


Figure 4.21: Circuit design of oversample and CP remover

(3) **Pilot zero tones mover:**

In this stage, we reserve all data tones, remove zero tones, and gather pilot tones together for convenience of the following procedures. Remembering that these post-FFT data are in bit-reverse order, therefore the order of writing address will be extremely complicated. Here we store our address order in a ROM named **index_rom** in advance, and sequentially read out the data to be the writing address. Furthermore, to achieve a higher data rate, the stored data are parallelly fed out symbol by symbol from RAMs, therefore the following stages, such as channel estimator, phase estimator, STBC detector, or VBLAST detector, can easily process succeeding symbols at the same time.

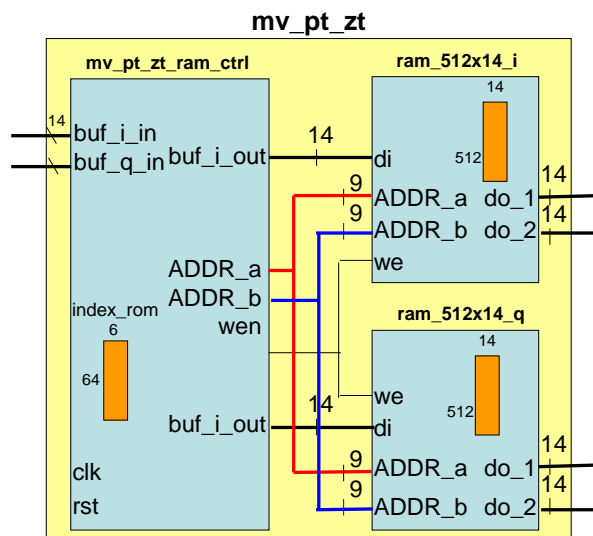


Figure 4.22: Circuit design of pilot zero tone mover

(4) **Channel estimator:**

Long preambles are used to carry out the major task of channel estimation. As shown in Figure 4.23, the frequency domain chips of the original long preambles is stored in ROMs, where the FFT output of the received long preamble is also fed into this stage and decoupled by the rule of STBC in accumulator. Therefore, the channel frequency response can be obtained simply by dividing these two outputs in **ch_est_div** blocks. Finally, the successive blocks truncate, modify the output of dividers and then store them in RAMs.

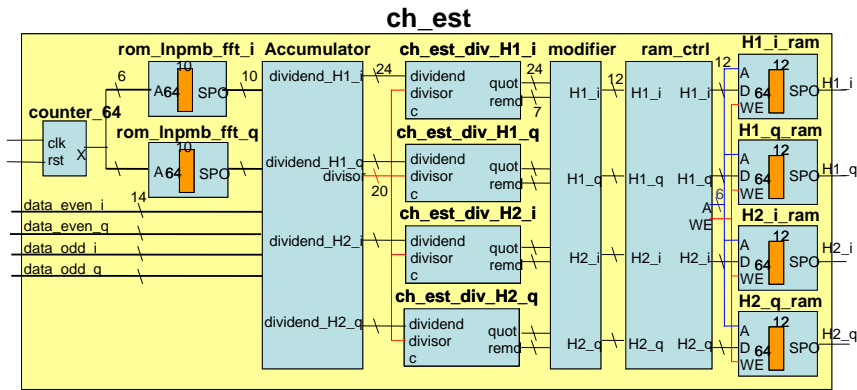


Figure 4.23: Circuit design of channel estimator

(5) **Phase estimator:**

In Section 2.3.4, we have discussed the phase estimation algorithm, and the estimated phase can be figured out by Eq. 2.8 with an arc tangent operation \angle . However, when it turns to FPGA realization, we care about the relative amplitude between real and image parts of the estimated phase but not the exact angle value, since we can compensate phase shift by multiplying the conjugate estimated complex phase and the received data. Therefore, no arc tangent circuit is implemented in the phase estimation block shown below, and no sine or cosine circuits are required to be implemented in the STBC detector or VBLAST detector, either. That can save a lot of hardware resources.

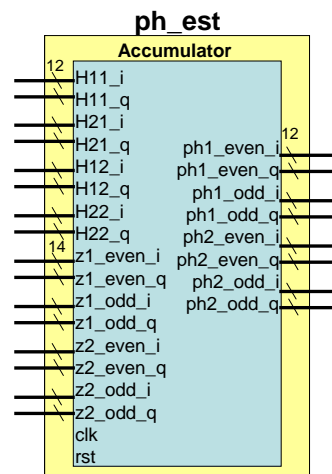


Figure 4.24: Circuit design of phase estimator

(6) **STBC decoder:**

The STBC decoder and the de-mapper are integrated in a single block in order to save additional RAMs. The estimated channel information, estimated phase information, and post-FFT data are first fed into **Accumulator** block, where complicated complex number computations are executed inside. Here we do not divide the diversity gain $(|H_{11}^k|^2 + |H_{21}^k|^2 + |H_{12}^k|^2 + |H_{22}^k|^2)$ as shown in Eq. 2.14 since only a signed bit is required to be checked in the following QPSK de-mapping block. The detected results are transferred to the next block **ram_ctrl** where de-mapping is executed inside. Because the estimated channel, phase, and received data are parallely sent into this stage, the de-mapped data are parallely figured out and then stored into a 1024x1 RAM.

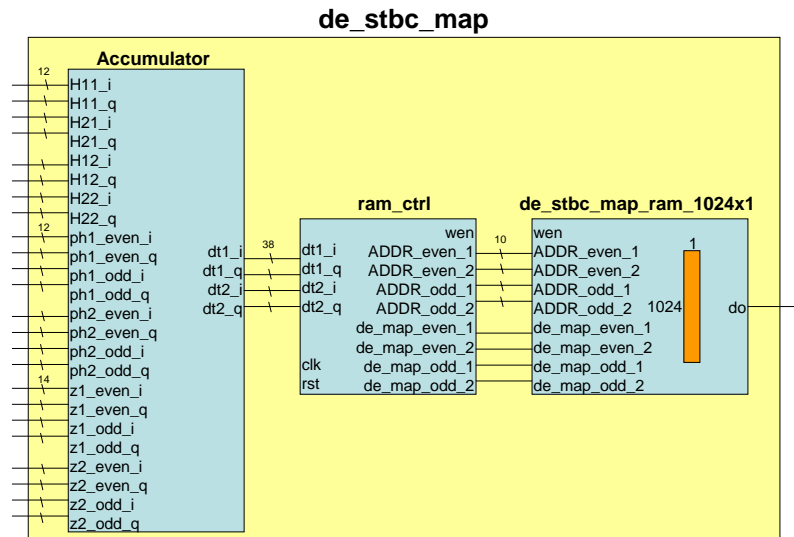


Figure 4.25: Circuit design of STBC decoder

(7) **VBLAST detector:**

The computations in the VBLAST detector is much more complicated than what in the STBC detector, therefore the critical path problem is much severer too. In order to explain how we deal with this problem, we must first introduce how we separate the original complex number computations. Figure 4.25 shows our separating strategy, where whole complex number arithmetic is separated into 7 blocks and detailed real calculating tasks in every block are shown in Table 4.2.

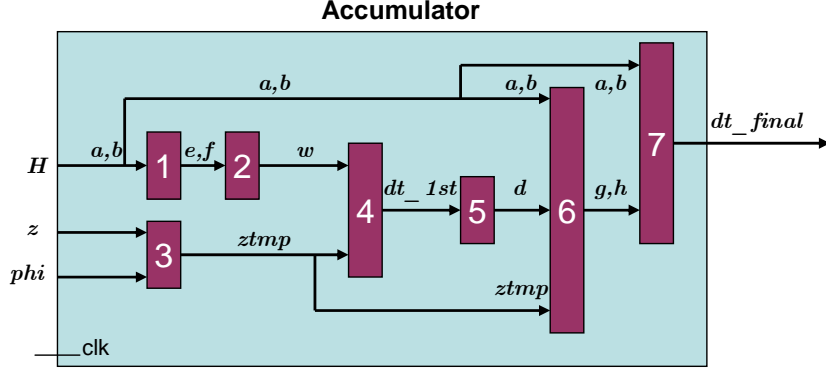


Figure 4.26: Original real number calculating strategy in VBLAST

$\begin{matrix} a, b \\ \rightarrow \\ \text{1} \\ \leftarrow \\ e, f \end{matrix}$	$e1=a11*a22-b11*b22-a12*a21+b12*b21;$ $f1=a11*b22+a22*b11-a12*b21-a21*b12;$
$\begin{matrix} e, f \\ \rightarrow \\ \text{2} \\ \leftarrow \\ w \end{matrix}$	$w11_i=(a22*e1+b22*f1)/(e1*e1+f1*f1); \quad w11_q=(-a22*f1+b22*e1)/(e1*e1+f1*f1);$ $w21_i=(-a21*e1-b21*f1)/(e1*e1+f1*f1); \quad w21_q=(a21*f1-b21*e1)/(e1*e1+f1*f1);$ $w12_i=(-a12*e1-b12*f1)/(e1*e1+f1*f1); \quad w12_q=(a12*f1-b12*e1)/(e1*e1+f1*f1);$ $w22_i=(a11*e1+b11*f1)/(e1*e1+f1*f1); \quad w22_q=(-a11*f1+b11*e1)/(e1*e1+f1*f1);$
$\begin{matrix} z \\ \rightarrow \\ \text{3} \\ \leftarrow \\ phi \\ \rightarrow \\ ztmp \end{matrix}$	$z1tmp_i=z1_i*phi1_i+z1_q*phi1_q; \quad z1tmp_q=-z1_i*phi1_q+z1_q*phi1_i;$ $z2tmp_i=z2_i*phi2_i+z2_q*phi2_q; \quad z2tmp_q=-z2_i*phi2_q+z2_q*phi2_i;$
$\begin{matrix} w \\ \rightarrow \\ \text{4} \\ \leftarrow \\ dt_1st \\ \rightarrow \\ ztmp \end{matrix}$	$dt1_1st_i=w11_i*z1tmp_i-w11_q*z1tmp_q+w21_i*z2tmp_i-w21_q*z2tmp_q;$ $dt1_1st_q=w11_i*z1tmp_q+w11_q*z1tmp_i+w21_i*z2tmp_q+w21_q*z2tmp_i;$ $dt2_1st_i=w12_i*z1tmp_i-w12_q*z1tmp_q+w22_i*z2tmp_i-w22_q*z2tmp_q;$ $dt2_1st_q=w12_i*z1tmp_q+w12_q*z1tmp_i+w22_i*z2tmp_q+w22_q*z2tmp_i;$
$\begin{matrix} dt_1st \\ \rightarrow \\ \text{5} \\ \leftarrow \\ d \end{matrix}$	Do first time QPSK decision
$\begin{matrix} a, b \\ \rightarrow \\ \text{6} \\ \leftarrow \\ d \\ \rightarrow \\ g, h \\ \rightarrow \\ ztmp \end{matrix}$	$g1=z1tmp_i-a21*d2_i+b21*d2_q; \quad h1=z1tmp_q-a21*d2_q-b21*d2_i;$ $g2=z2tmp_i-a22*d2_i+b22*d2_q; \quad h2=z2tmp_q-a22*d2_q-b22*d2_i;$ $g3=z1tmp_i-a11*d1_i+b11*d1_q; \quad h3=z1tmp_q-a11*d1_q-b11*d1_i;$ $g4=z2tmp_i-a12*d1_i+b12*d1_q; \quad h4=z2tmp_q-a12*d1_q-b12*d1_i;$
$\begin{matrix} a, b \\ \rightarrow \\ \text{7} \\ \leftarrow \\ dt_final \\ \rightarrow \\ g, h \end{matrix}$	$dt1_final_i=a11*g1+b11*h1+a12*g2+b12*h2;$ $dt1_final_q=a11*h1-b11*g1+a12*h2-b12*g2;$ $dt2_final_i=a21*g3+b21*h3+a22*g4+b22*h4;$ $dt2_final_q=a21*h3-b21*g3+a22*h4-b22*g4;$

Table 4.2: Tasks in VBLAST accumulator

However, by experimenting, these seven tasks cannot be completely executed in a single clock period. Therefore, the original dividing strategy is modified as shown in Figure 4.27. We can see that two accumulators are realized, and only three to four real calculating tasks are required to be executed in a clock period, thus the accurate results can be figured out in time. By this kind of rearrangement, critical path problem is resolved. Final circuit design of the VBLAST detector with integration of de-mapper is shown in Figure 4.28.

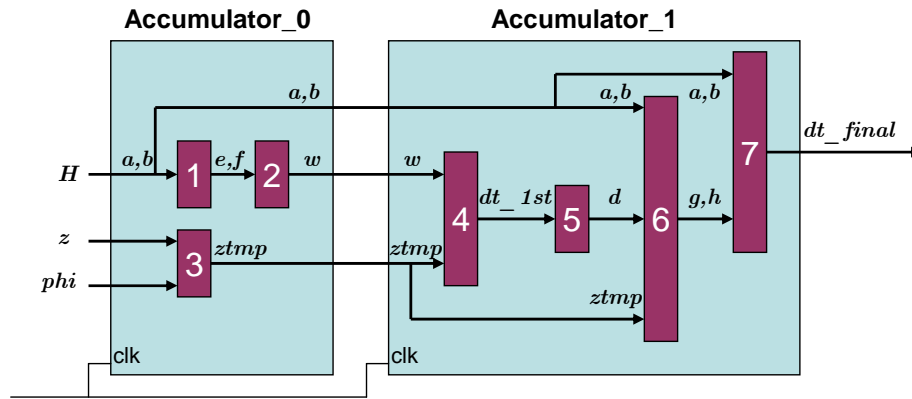


Figure 4.27: Modified real number calculating strategy in VBLAST

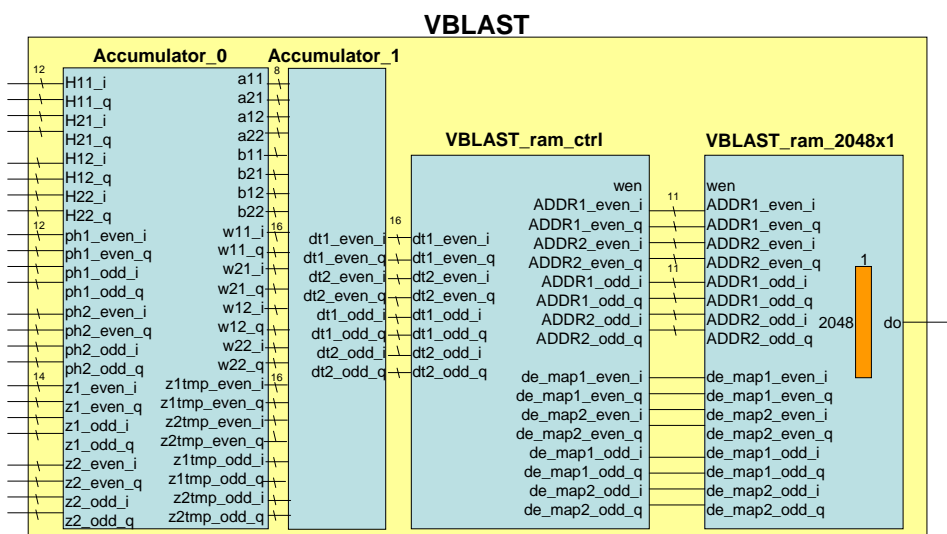


Figure 4.28: Circuit design of VBLAST detector

(8) Viterbi decoder:

The circuit design of the Viterbi decoder is shown in Figure 4.29. Three main blocks are included: branch metric generator (BMG); add, compare, and select (ACS) block; and the trace back unit (TBU). The BMG unit generates the branch metrics for each symbol of the input sequence by comparing the received code symbol with the expected code symbol for each connection of the trellis (state) and counts the number of different bits. For a 1/3 rate code adopted in our system, there are eight possible symbol combinations in the encoded sequence: 000, 001, 010, 011, 100, 101, 110, and 111; therefore eight BMG units are implemented in BMG block as shown in Figure 4.30.

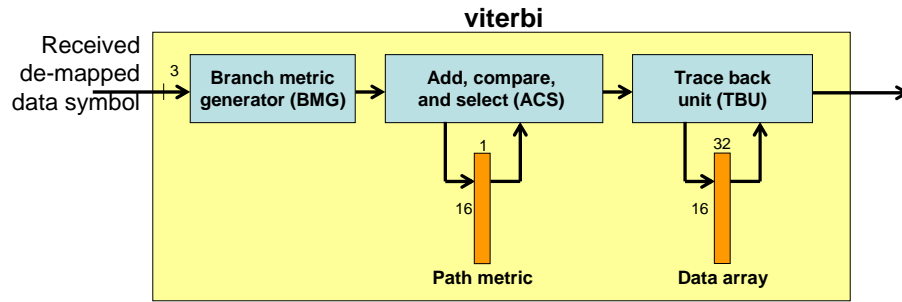


Figure 4.29: Circuit design of Viterbi decoder

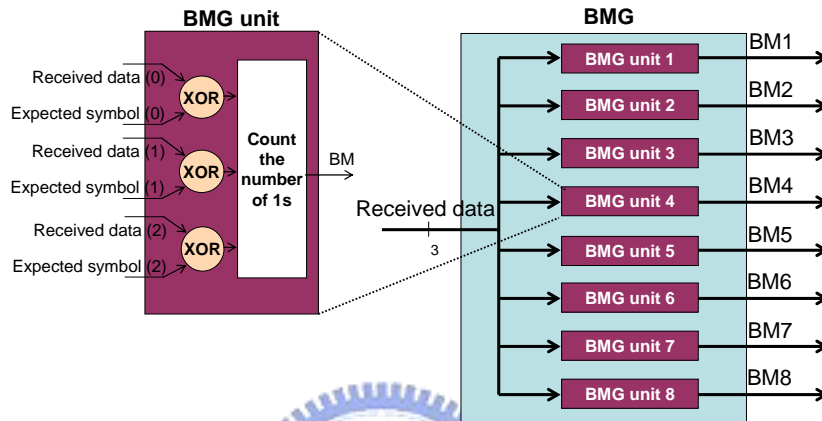


Figure 4.30: Circuit design of branch metric generator

The ACS unit is the heart of the Viterbi decoder. Each node in the trellis diagram corresponds to an ACS unit in the corresponding Viterbi decoder. Therefore, referring to the trellis diagram shown in Figure 2.8, there should be totally 16 ACS units in the ACS block as shown in Figure 4.31. The ACS unit has 4 inputs (two branch metrics and two path metrics) and two outputs (the new path metric and the survivor bit). The survivor bit is the most important information generated by the ACS unit. It indicates which sum between an input path metric and a branch metric generated the smallest result and was selected as the output path metric or local winner.

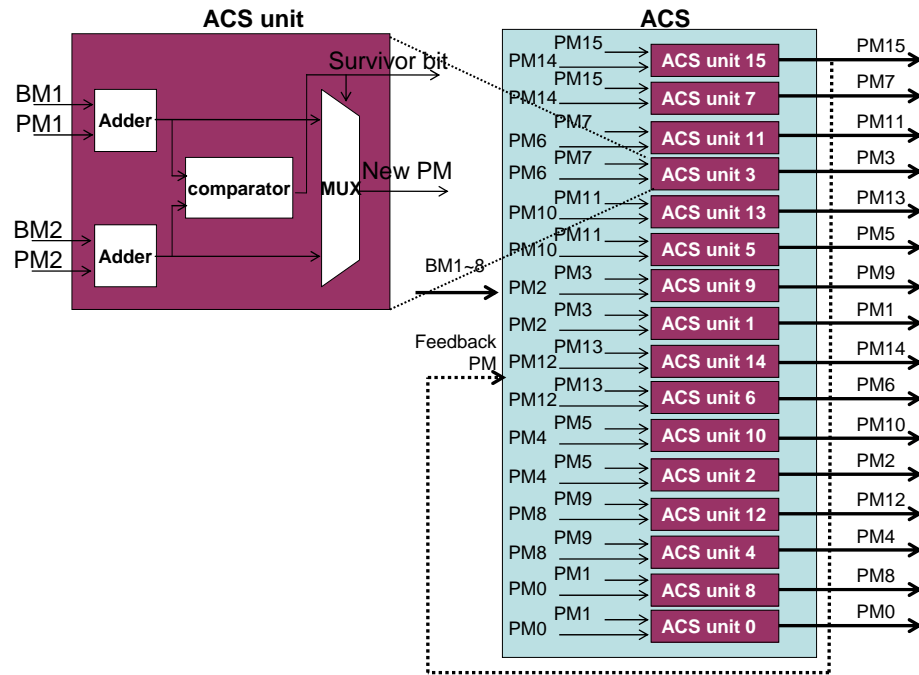


Figure 4.31: Circuit design of add, compare, and select block

The ACS block assigns the measurement functions to each state, but the actual Viterbi decisions on encoder states are based on the trace back operation to find the path of the states. Using the trace back operation, every state from a current time is followed backwards through its maximum likelihood path. The point at which the corrected bit streams starts is called the merger point (also called the trace back depth). The performance of Viterbi decoder largely depends upon the trace back depth. The increase in trace back depth increases the complexity and hardware exponentially so one has to trade off between the performance level and the complexity and hardware.

Normally for decoders using non-punctured codes, the trace back depth equals five-times constraint length, which is sufficient to decode the correct output in the presence of noise. In our system, the constraint length is 5, therefore twenty-five trace back depth is required. We adopt a 16×32 register array to store the path of the states. Comparing with original 16×192 (STBC) or 16×384 (VBLAST) register array, a large amount of FPGA resources are saved.

4.4 ModelSim simulation

When developing an FPGA system, ModelSim simulation can help designers developing efficiently and accurately. It can pull out all signals and simulate how they work simultaneously without the limitation of the number of debugging pins, therefore, designers can save a lot of time downloading to FPGA and directly examine the changes and interactions between signals. Figure 4.32 and 4.33 shows the data flows in STBC and VBLAST system. In STBC case, total flow spends approximately 110 us. On the other hand, VBLAST case spends approximately 150 us. VBLAST spends more time than STBC because that two times source data are required to be dealt with.

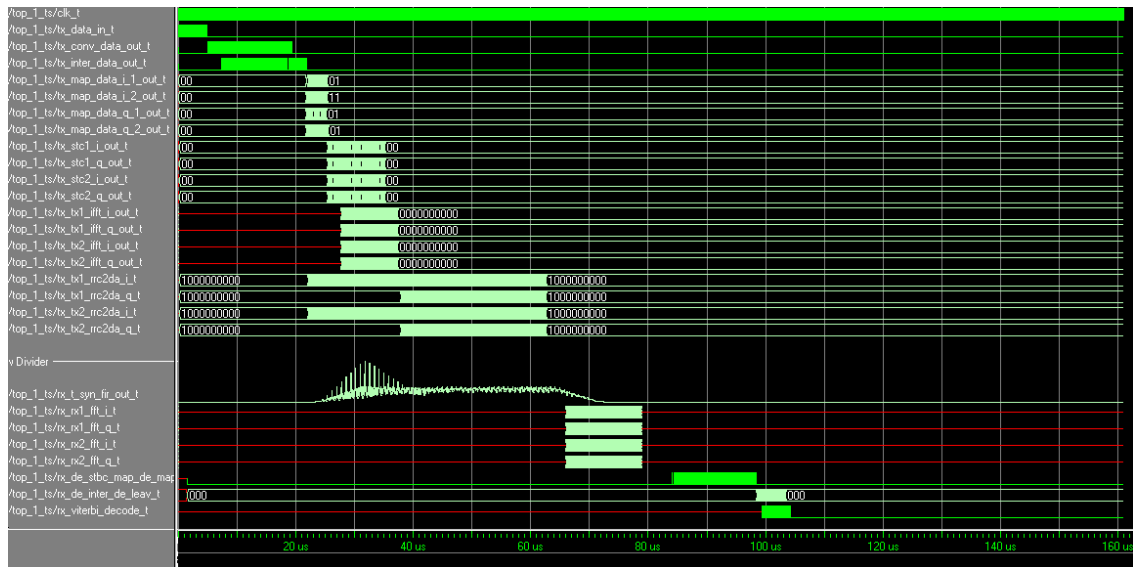


Figure 4.32: STBC ModelSim simulation result

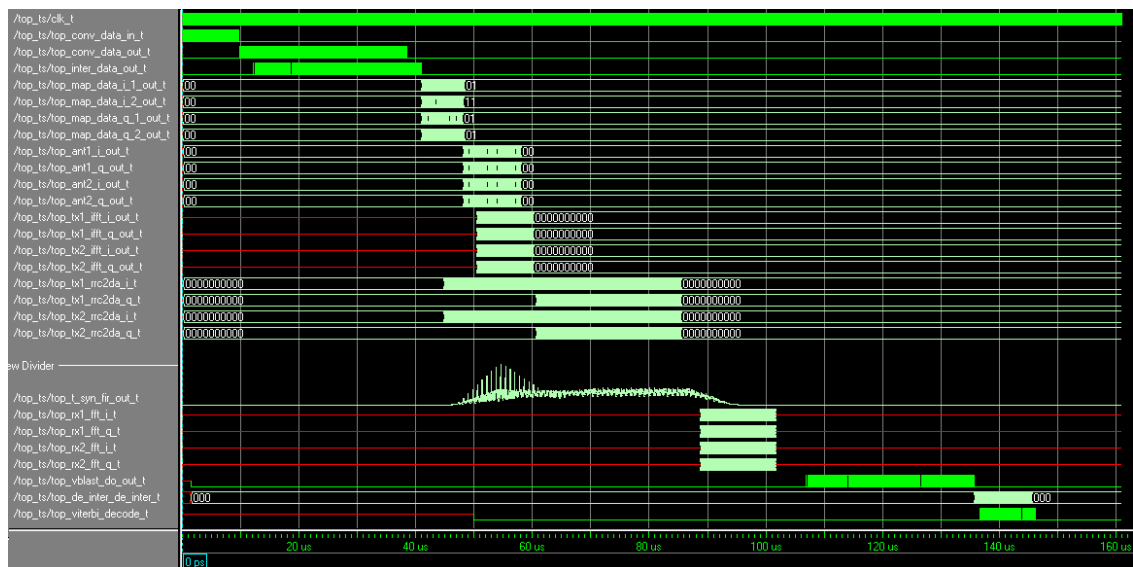


Figure 4.33: VBLAST ModelSim simulation result

In addition to total data flow period, some design concepts such as parallel processing and overlapped processing also can be observed through above two figures. Figure 4.34 shows the transmitted waveform. Signal in first 16 us is preamble channel, which is BPSK modulated; the rest of data are OFDM symbols.

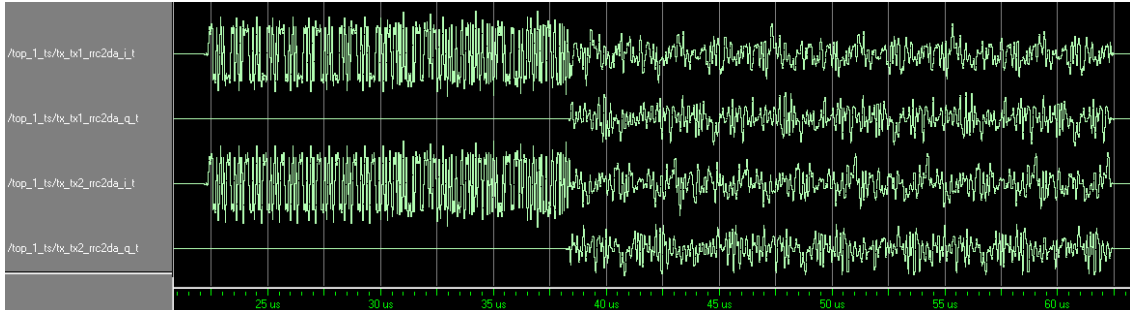


Figure 4.34: Transmitted waveform of MIMO-OFDM system

4.5 Experimental Results

There are two platforms for us to download our baseband codes and perform the advanced verification. In order to take account of interaction between other modules such as DSP, USB, and AD/DA on these two platforms, modification needs to be executed frequently. Therefore to synthesis, map, and place and route iteratively seems to be unavoidable and always waste a lot of time. Besides time consuming, the insufficiency of FPGA gate count becomes another problem, especially on the VBLAST receiver side. Therefore we must try our best to save gate count, and that is an important reason why we try to find out a quantization algorithm that can minimize the hardware resource requirement. Table 4.3 shows time and area consumption in our developing flow, where whole design flow including developing transmitter and receiver takes 2 to 4 hours. Therefore to test a system is quite time consuming.

Table 4.2: Synthesis and P&R information

	STBC		VBLAST	
	Tx (VirtexE 2000)	Rx (Virtex2 6000)	Tx (VirtexE 2000)	Rx (Virtex2 6000)
Synthesis Time	20mins	30mins	42mins	1hr 30mins
Place and Route Time	30mins	45mins	40mins	50mins
Total LUTs	32925 (85%)	45228 (66%)	35035 (91%)	59717 (88%)
Block Multipliers	NA	132 (91%)	NA	144 (100%)

4.5.1 Fast Prototyping Platform

In the fast prototyping platform, we successfully integrate FPGA, DSP, USB, and AD/DA modules. First the web camera catches the real time images continuously as the data source, and then passes it to DSP module. DSP, without any processing, directly pass the data to FPGA, and FPGA performs MIMO-OFDM transmitter algorithm. After the processing of transmitter, data are passed through DA and received by AD. Subsequently AD passes data to receiver FPGA, and start to decode the received data. Finally, the decoded data are sent back to PC through DSP and USB module, and shows through the self-developed application software in PC. We can provide an user interface to demonstrate the real time transmitted and received images, as shown in Figure 4.35. In this figure, a 3×3 images set is located. The three columns represent transmit images, receive images, and error images respectively, whereas the three rows represent the synthesized images of all antennas, first antenna, and second antennas respectively. The real time bit error rate is also calculated and shown in the right hand side.

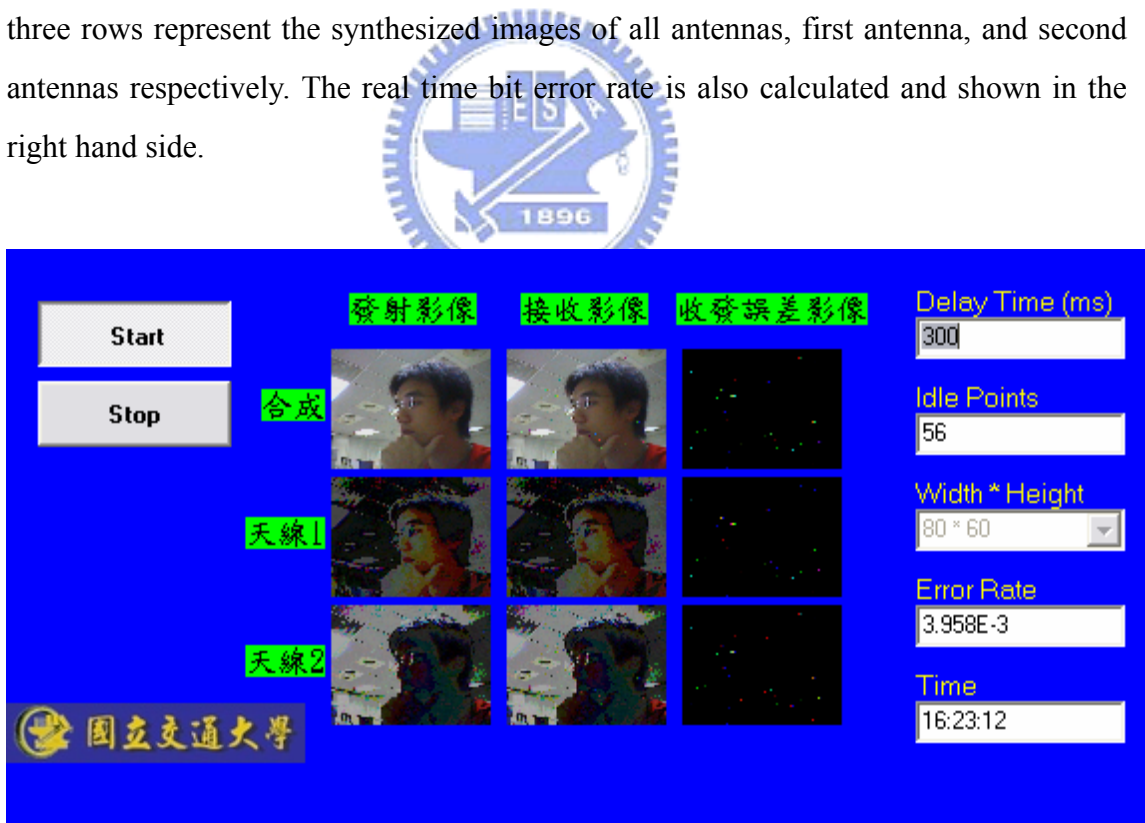


Figure 4.35: Prototyping platform experimental result

4.5.2 Self-designed Platform

In the self-designed platform, we attempt to establish a real wireless environment, under which the adopted algorithm can be tested. Figure 4.36 shows the experimental environment which has been shown in Chapter 3. First, source data are stored in a ROM in FPGA, and passed to DA after processing by transmitter algorithm on FPGA. Next, data are transmitted on the 5.2 GHz frequency band by the RF module, and a receive antenna is allocated near the RF module. Subsequently data are received by the receive antenna and passed to spectrum analyzer E4443A and vector signal analyzer 89600S. Finally, received data are analyzed and shown on PC. Figure 4.37 shows the analyzed result, which can represent the effects of a real wireless channel. We can see that in frequency domain, the measured center frequency is 5.200152 GHz, and the occupied bandwidth (OBW) is approximately 20 MHz. In time domain, due to the mismatch between mixers in transmitter and receiver, preamble data (only transmitted in real part) is distributed into real and image parts in the receiver. Otherwise, owing to the effect of frequency offset, the slight swing in envelop of received data also can be observed.

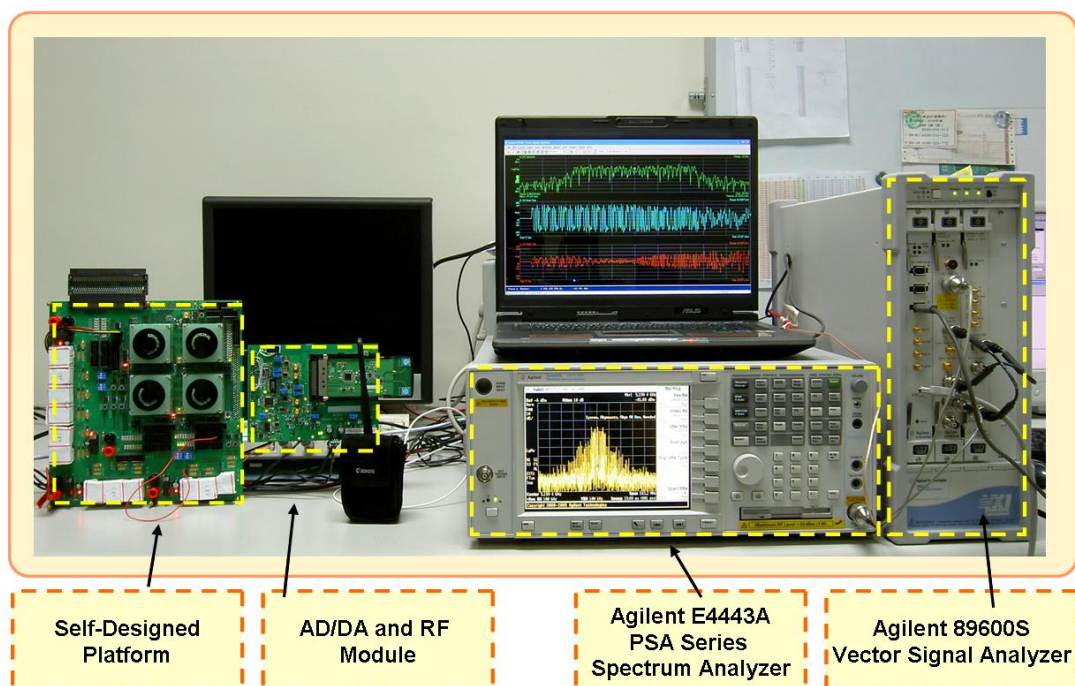


Figure 4.36: Self-designed platform development environment

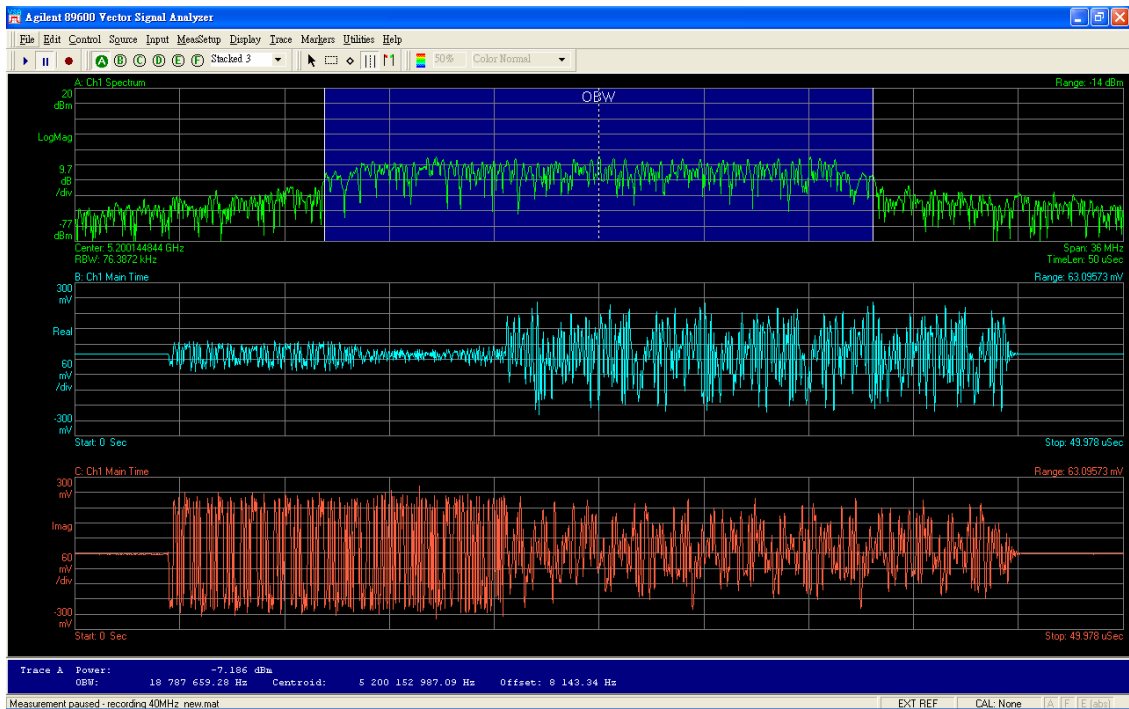


Figure 4.37: Self-designed platform experimental result: received spectrum and waveforms on PSA and VSA

After the received data pass through AD converter, all signals are digitalized and therefore can be measured by the logic analyzer easily. Figure 4.38 shows the waveform of timing synchronizer measured by the logic analyzer. As simulated in MATLAB and ModelSim, timing synchronization output forms a hill and the peak time index is regarded as the packet start time.

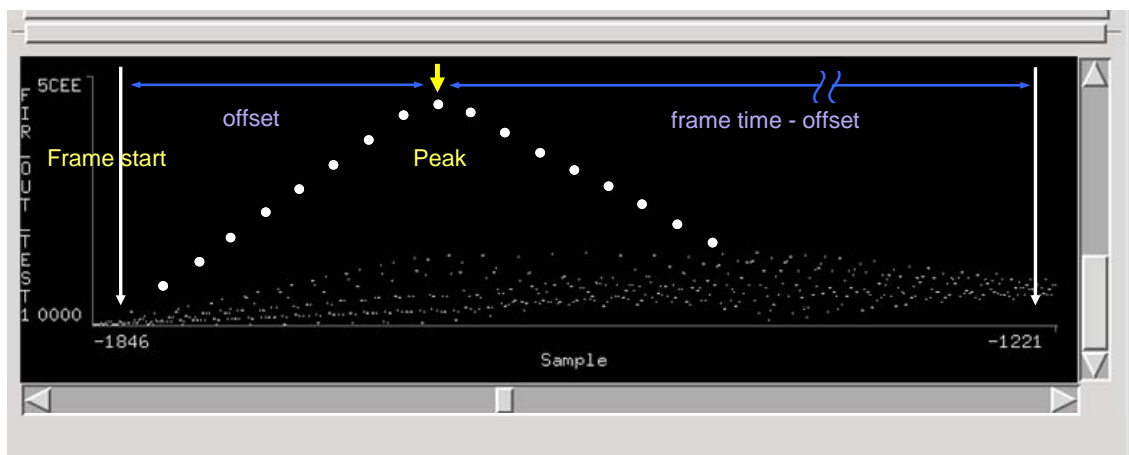


Figure 4.38: Self-designed platform experimental result: timing synchronization waveform on LA

Figure 4.39 shows the source data stream in transmitter, transmitted data stream, and detected data stream in the receiver, where the source data stream and the detected data stream are specially expanded below. By comparing the source data stream with detected data stream we can find out that they are exactly the same, which confirms that our algorithm does work successfully.

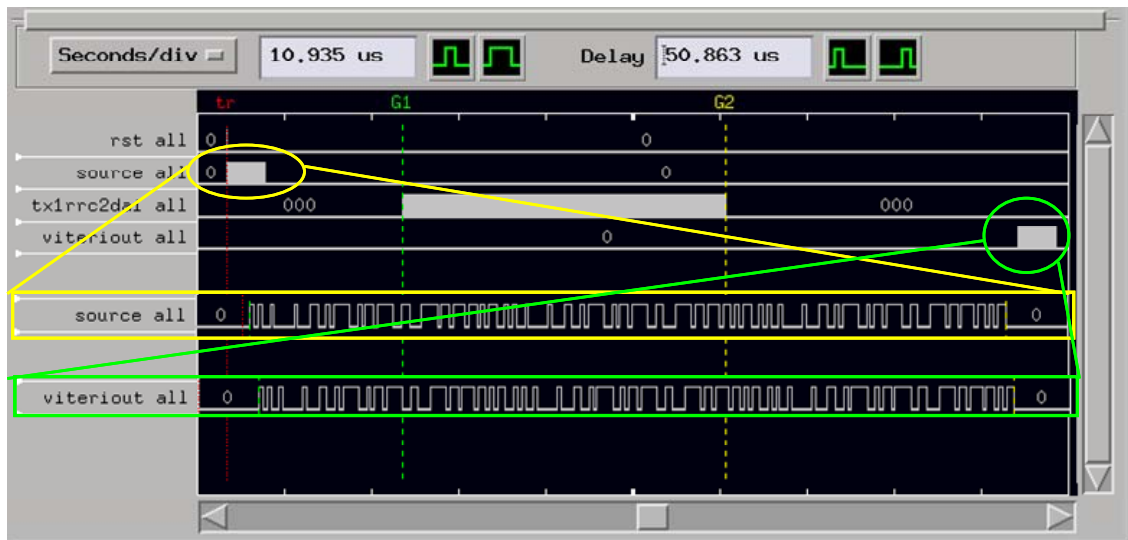


Figure 4.39: Self-designed platform experimental result: source data and detected data waveform on LA

4.6 Summary

In this chapter, a complete communication system design flow is proposed, including MATLAB verification, FPGA realization, ModelSim simulation, and experimental results. Through this design flow, we finish developing a 2×2 MIMO-OFDM system on two FPGA-based platforms, e.g., fast prototyping platform and self-designed platform. On the fast prototyping platform, we integrate our communication algorithm with web camera, and demonstrate real time video on the self-developed software interface. On the self-designed platform, real wireless channel effects can be generated by means of RF module, and some RF debugging instruments, which makes our system become much closer to real communication system.

Chapter 5

Proposed Quantization Algorithm with Minimum Hardware Requirement

The algorithms used by DSP systems are typically specified as floating-point DSP operations. On the other hand, most digital FPGA implementations of these algorithms rely solely on fixed-point approximations to reduce the cost of hardware while increasing throughput rates. The essential design step of floating-point to fixed-point conversion is not only time consuming, but also complicated due to the nonlinear characteristics and the massive design optimization space. In a bid to achieve short product cycles, the execution of floating to fixed-point conversion is often left to hardware designers, who are familiar with VLSI constraints. Comparing with the algorithm designers, this group often has less insight into the algorithm and depends on ad hoc approaches to evaluate the implications of fixed-point representations. The gap between algorithm and hardware design is even aggravated as algorithms continue to become more complex. Thus, a systematical method for floating to fixed-point conversion is urgently called for.

In this chapter, a quantization algorithm which is especially suitable for communication systems is proposed, where hardware resources are minimized, and the equivalent quantization error is constrained within a specified limit.

5.1 Introduction of Quantization

Numeric representation in digital hardware may be either fixed or floating-point. In fixed-point representation, the available bit-width is divided and allocated to the integer part and the fractional part, with the extreme left bit reserved for the sign (2's complement). In contrast, a floating-point representation allocates one sign bit and a fixed number of bits to an exponent and a mantissa. In fixed-point, relatively efficient implementations of arithmetic operations are possible in hardware. In contrast, the floating-point representation needs to normalize the exponents of the operands for addition and subtraction. Synthesizing customized hardware for fixed-point arithmetic operations is obviously more efficient than their floating-point counterparts, both in terms of performance as well as resource usage. In the following paragraphs, some fixed-point quantization examples will be introduced.

The first quantization example is shown in Figure 5.1, which illustrates two different fractional quantization methods. The full precision number "0011011110101000" represents 28496, and once we take the position of decimal point into account, the original number needs to divide 2^{14} and therefore becomes 1.7392578125. If we want to quantize the fractional part of this number from 14 bits to be 9 bits, two methods can be alternated, e.g., **truncation** and **rounding**. Truncation means to discard bits to the right of the least significant bit, that is, to remove right side "10000" directly, and the original number will lead to 1.73828125. Otherwise, rounding denotes to round the original number to the nearest representable value or the value farthest from zero if there are two equidistant nearest representable values. In rounding case, the quantized number will become 1.740234375, which is closer to full precision number than truncation case. Obviously, rounding performs better than truncation, yet will complicate the circuit and occupy more hardware resources.

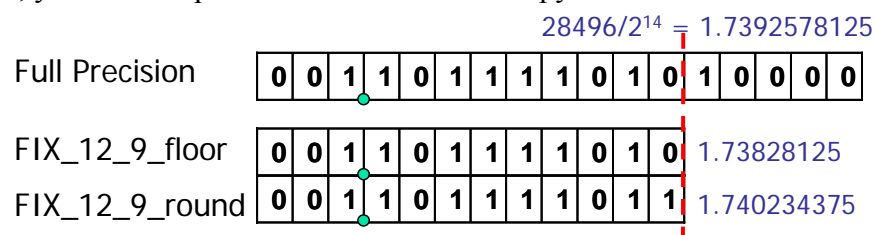


Figure 5.1: Quantization example 1: truncation and rounding

Besides fractional part quantization, truncating the integer part is also feasible. Figure 5.2 figures out the second example about how to truncate the integer part. Full precision number is 13.6875, and there are 5 bits including 1 sign bit to represent integer part. If we want to truncate integer part to 3 bits, **saturation** or **wrapping** can be alternatively adopted. In saturation case, the original number is saturated to the largest positive (or maximum negative) value; whereas wrapping case discards any significant bits beyond the most significant bit. In this example, saturated number is 3.9375, and wrapped number is -2.3425, where wrapping causes a significant quantization error. In communication system, especially in OFDM system, designers will face serious PAPR problem, where dealing with overflow problem becomes an important issue. As the results of saturation and wrapping shown in this example, wasting a little circuit complexity and hardware resources to realize a saturation circuit instead of wrapping circuit after IFFT in order to mitigate PAPR effects is intensely recommended.

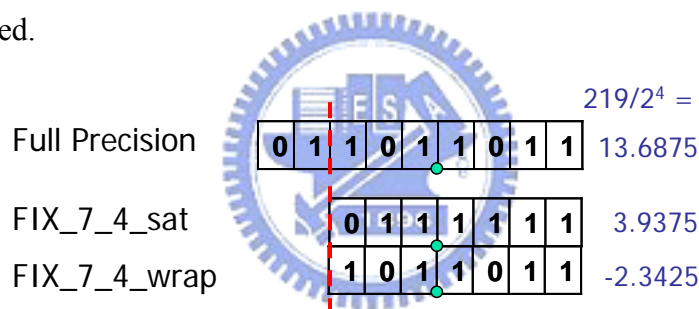


Figure 5.2: Quantization example 2: saturation and wrapping

MATLAB provides some functions that can translate floating-point values into fixed-point values, which enable designer to design, model, and simulate the system and to carry out the arithmetic in fixed-point domain. Fixed-point support is provided using the MATLAB quantization functionality that comes with the Filter Design and Analysis (FDA) Toolbox. This support is provided in the form of a quantizer object and two methods or functions that come with this object, namely, “*quantizer()*” and “*quantize()*.” The “*quantizer()*” function is used to define the quantizer object, which allocates the bit-widths to be used along with whether the number is signed or unsigned, what kind of rounding is to be used, and whether overflows saturate or wrap. The “*quantize()*” function applies the quantizer object to numbers, which are inputs to

and outputs from arithmetic operations. For example, a quantization model of type signed fixed-point, with 40 total bits with one sign bit, 8 integer bits, and 32 fractional bits, handling overflow with saturation is defined as follows in MATLAB:

```
q1 = quantizer('fixed','floor','saturate',[40,32]);  
Xq = quantize(q1,X);
```

This quantizer object is used to quantize an arbitrary numerical value “ X ” (which may be a scalar or a multidimensional vector) as shown above. The resulting number “ Xq ” has a double floating-point representation in MATLAB, but can be exactly represented by a 40-bit fixed-point signed number with 8 integer and 32 fractional bits.

5.2 Previous Work

The strategies for floating-point to fixed-point conversion can be roughly categorized into two groups [31]. The first one is basically an **analytical approach** coming from those algorithm designers who analyze the finite word length effects due to fixed-point arithmetic. The other approach is based on **bit-true simulation** originating from the hardware designers. The analytical approach started from attempts to model quantization error statistically; then it was expanded to specific linear time invariant (LTI) systems such as digital filters, FFT, etc. In the past three decades, numerous papers have been devoted to this approach [26]-[31]. The bit-true simulation method has been extensively used recently [32]-[35]. Its potential benefits lie in its ability to handle non-LTI systems as well as LTI systems.

Our proposed approach is very closely related to the approach of Roy and Banerjee [35], where the authors have developed a simulation-based method to determine the optimum word lengths for DSP algorithms. Although the authors claim that [35] the proposed approach can minimize the hardware resources while constraining the quantization error, the way they adopt to estimate the hardware resources (regard the bit precision of all quantizers as hardware resources) is too rough, therefore the final experimental result is not precise enough. Moreover, only few quantization methods are adopted in the approach, that is, truncation and wrapping,

which are insufficient to satisfy the characteristics such as PAPR in communication systems. To modify these two defects, the concept of hardware resource weighting is introduced in our quantization algorithm, which makes hardware resources estimation much more accurate; moreover in order to fit our special purposed system, e.g., communication system, not only truncation and wrapping but also saturation are adopted in our quantization algorithm.

5.3 Proposed Quantization Algorithm

Our algorithm attempts to minimize the hardware resource requirement while constraining quantization error within a specified limit, depending on the requirement of the user or application. Especially, the concept of hardware resource weighting is introduced therefore our algorithm can accurately estimate the hardware resources requirement. The quantization algorithm consists of the following passes, which are explained in detail in the next paragraphs:

- Pre-quantization works
- Determine hardware resource weightings
- Determine integer lengths
- Determine fraction lengths

5.3.1 Pre-quantization Works

Before executing the proposed quantization algorithm on our MIMO-OFDM system, some pre-quantization works need to be operated first.

1. **Separate all blocks into quantization-related and quantization-irrelevant blocks, and find out the performance-dominated data flow path**

In order to convert floating-point MATLAB code into fixed-point MATLAB code, first we need to separate all MIMO-OFDM function blocks into two groups, e.g., quantization-related blocks and quantization-irrelevant blocks:

- **Quantization-related blocks:**

Function blocks are called quantization-related as long as there are DSP

operations, such as the four fundamental operations of arithmetic, FFT/IFFT and so on, between input signals and output signals. When we are dealing with fixed-point variables without truncation, these DSP operations will always make variable's word length become longer and longer, and therefore we must face quantization issues. In our MIMO-OFDM system, the following function blocks are distributed into the quantization-related group:

- IFFT/FFT
- RRC filter
- Timing synchronizer
- Channel estimator
- Phase estimator
- STBC detector
- VBLAST detector

● **Quantization-irrelevant blocks:**

The definition of quantization-irrelevant blocks is right opposite to quantization-related blocks. Besides the quantization-related blocks shown above, the rest of blocks in our MIMO-OFDM system are categorized into quantization-irrelevant blocks. When we are dealing with the floating-point to fixed-point conversion, these blocks will remain the same because they will suffer neither rounding nor truncation issues.

Figure 5.3 shows the distribution of quantization-related and quantization-irrelevant blocks in our system, and shows two data flow paths in the system. The first data flow path, named major data flow path, starts from convolutional encoder and ends to Viterbi decoder; the second data flow path, named minor data flow path, starts from convolutional encoder and ends to timing synchronizer. Since quantization will cause additional error, called quantization error, it can be easily observed that the quantization errors along major data flow path will dominate the system performance much more severely than what along the minor data flow path; therefore our further works will focus on determining the word lengths of the coefficients or variables in the quantization-related function blocks along the major data flow path.

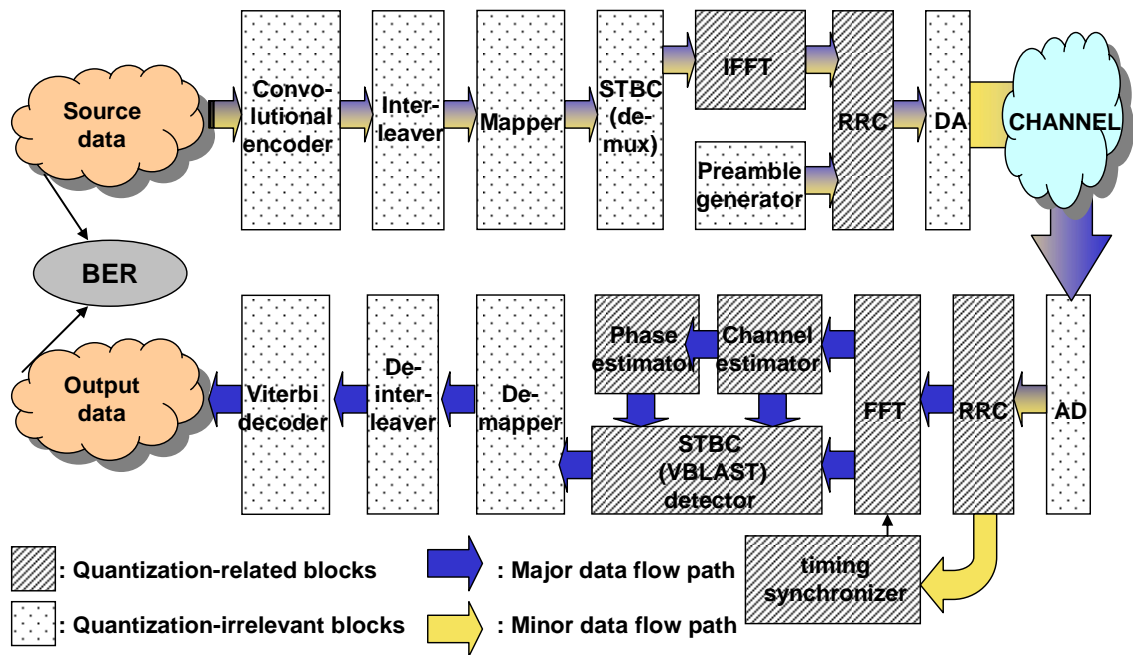


Figure 5.3: Data flow paths and distribution of quantization-related blocks and quantization-irrelevant blocks in MIMO-OFDM system

2. Find out the variables which need to be quantized in quantization-related blocks along the main data flow path

Here we list the variables whose word lengths are required to be determined in the function blocks along the major data flow path:

- IFFT output data
- RRC coefficients
- AD/DA length
- Word length of RRC output data in receiver side
- FFT output data
- Post-FFT long preamble chips embedded in channel estimator block
- Channel estimator output data
- Phase estimator output data

Note that although AD and DA are not quantization-related blocks, there available word lengths are inherently limited by hardware circuit design to be 10 bits therefore AD and DA's word length also needs to be taken into consideration as we convert floating-point codes into fixed-point codes.

3. Parameters settings and definitions

To concentrate on dealing with quantization issues purely, channel effects such as multipath, AWGN noise, Rayleigh fading and so on are all neglected. Furthermore, the detected constellation of floating-point case is regard as the basis, where the difference between the output vectors for the original floating-point and the fixed-point MATLAB code is regarded as error vector, denoted by e .

$$e = \text{outdata}_{\text{float}} - \text{outdata}_{\text{fixed}} \quad (5.1)$$

We next define an error metric (EM) using the following definition:

$$EM = \text{norm}(e) / \text{norm}(\text{outdata}_{\text{float}}) \times 100 \quad (5.2)$$

where the vector norm is defined by

$$\text{norm}(e) = \left(\sum_i |e_i|^2 \right)^{1/2} \quad (5.3)$$

After that, we define the following terms that are used in the algorithm

- fl_dt : floating-point detected constellation
- fx_dt : fixed-point detected constellation
- max_i : range of the i th variable
- p_i : the i th precision (fraction length)
- m_i : the i th integer length (including sign bit)
- q_i : the i th quantizer
- w_i : the i th hardware resource weighting
- R : total hardware resource requirement

Additionally, in spite of some special variables, signed fixed-point value with truncation and wrapping quantization methods are carried out. However, since we try to operate this quantization algorithm on our MIMO-OFDM system, some communication characteristics require to be taken into account and therefore the following quantization settings should be modified:

- IFFT: In order to mitigate PAPR problem, we choose saturation method to quantize the output signals of IFFT.

- AD/DA: With limit of the circuit design of AD/DA module, the available bit-widths is constrained, e.g., ten bits in our system. Moreover, the signals fed into DA and fed out from AD are limited to unsigned values, therefore quantization method must be unsigned too. Furthermore, quantization methods are also constrained, where truncation and wrapping are adopted by DA; truncation and saturation are adopted by AD.

Here we list all quantizers and their quantization settings in Table 5.1.

Table 5.1: Quantizers and their settings in MIMO-OFDM system

Variables	Parameters and Quantizers
IFFT output data	$q_{ifft} = \text{quantizer}(\text{'fixed'}, \text{'floor'}, \text{'sat'}, [m_{ifft} + p_{ifft}, p_{ifft}])$
RRC coefficients	$q_{RRC} = \text{quantizer}(\text{'fixed'}, \text{'floor'}, \text{'wrap'}, [m_{RRC} + p_{RRC}, p_{RRC}])$
DA output data	$q_{DA} = \text{quantizer}(\text{'unfixed'}, \text{'floor'}, \text{'wrap'}, [m_{DA} + p_{DA}, p_{DA}])$
AD output data	$q_{AD} = \text{quantizer}(\text{'unfixed'}, \text{'floor'}, \text{'sat'}, [m_{AD} + p_{AD}, p_{AD}])$
RRC output data in receiver	$q_{rxRRC} = \text{quantizer}(\text{'fixed'}, \text{'floor'}, \text{'wrap'}, [m_{rxRRC} + p_{rxRRC}, p_{rxRRC}])$
FFT output data	$q_{fft} = \text{quantizer}(\text{'fixed'}, \text{'floor'}, \text{'wrap'}, [m_{fft} + p_{fft}, p_{fft}])$
Post-FFT long preamble chips	$q_{lnfft} = \text{quantizer}(\text{'fixed'}, \text{'floor'}, \text{'wrap'}, [m_{lnfft} + p_{lnfft}, p_{lnfft}])$
Channel estimator output data	$q_{ch} = \text{quantizer}(\text{'fixed'}, \text{'floor'}, \text{'wrap'}, [m_{ch} + p_{ch}, p_{ch}])$
Phase estimator output data	$q_{ph} = \text{quantizer}(\text{'fixed'}, \text{'floor'}, \text{'wrap'}, [m_{ph} + p_{ph}, p_{ph}])$

5.3.2 Determine Hardware Resource Weightings

Hardware resources weightings (w_i) can help our proposed algorithm estimating hardware resources requirement accurately, where it indicates the corresponding hardware overhead as one additional bit is added. To acquire weighting factors correctly, designers must know circuit design very well. Otherwise, synthesis tools such as Synplify Pro or Synopsis is also needed to count corresponding hardware resources. However, too much hardware resources usage information such as embedded RAMs, block multipliers, registers, and LUTs is included in a single synthesis report therefore to define a equivalent weighting factor to represent total hardware resources usage is very difficult. Based on experimental experience, we find that the usage of LUTs can represent hardware resource most properly, therefore we

adopt the usage of LUTs as our hardware resource weighting factor directly in our quantization algorithm.

There are two ways to get additional usage of LUTs as certain word length of system is changed: first method is to design and synthesis another system, where there are only certain word length and corresponding circuits different from original one; then, comparing the difference between the LUT usage of these two systems, and view the quotient of this difference and additional bit-widths as weighting factor. Another method is to find out the influenced circuits such as subsequent data buffers or calculators when certain word length increase 1 bit, and then synthesize them respectively. Subsequently, sum up their additional LUTs and regard it as weighting factor of certain variable.

Obviously, the first method can calculate weighting factor more accurately than another, however will waste extremely large amount of time on designing and synthesizing new systems. Therefore method two seems to be much more feasible than method one thus is adopted in our algorithm.

Here we illustrate the influenced circuit blocks in Figure 5.4 and Figure 5.5, and show final experimental results in Table 5.2. Notice that our synthesis tool is Synplify Pro 8.2, and target FPGA is Xilinx Virtex2 series.

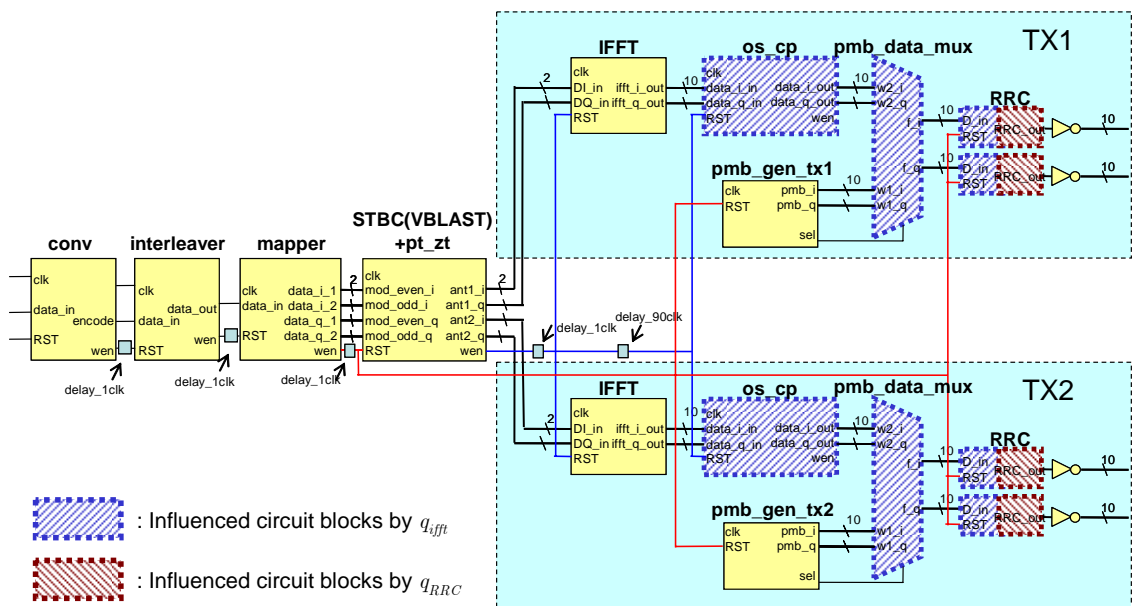


Figure 5.4: Influenced circuit blocks by q_{ift} and q_{RRC}

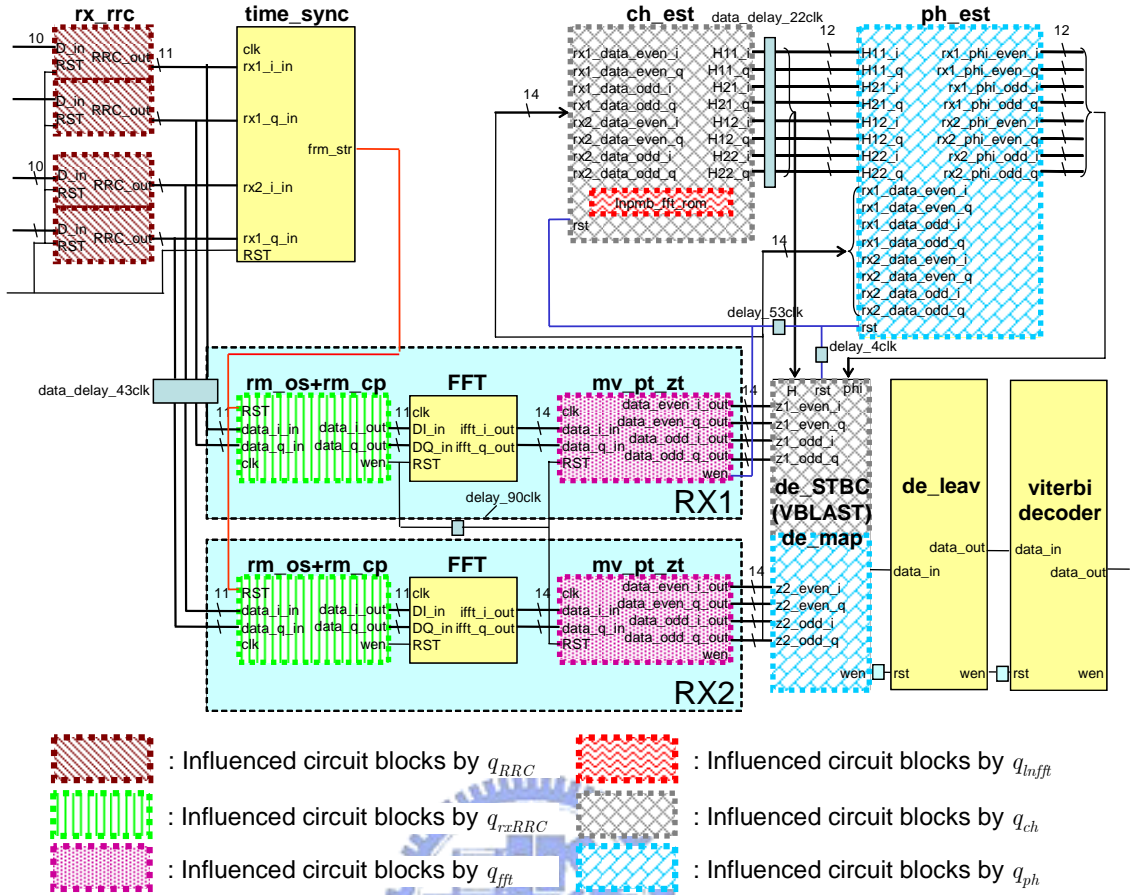


Figure 5.5: Influenced circuit blocks by q_{RRC} , q_{rxRRC} , q_{fft} , q_{inpmb} , q_{ch} and q_{ph}

Table 5.2: Experimental results of hardware resource weightings

Variables	Parameter	Experimental Results
IFFT output data	w_{ifft}	402
RRC coefficients	w_{RRC}	20
RRC output data in receiver	w_{rxRRC}	240
FFT output data	w_{fft}	20
Post-FFT long preamble chips	w_{infft}	416
Channel estimator output data	w_{ch}	184
Phase estimator output data	w_{ph}	35

5.3.3 Determine Integer Lengths

The determined integer lengths must be able to cover all possible maximum and minimum signed values of variables, that is, avoid overflow efficiently. To achieve this

goal, the usage of training sets is necessary. The more training sequences we test, the more reliable variables ranges we obtained. In communication system, we train the system and obtain the ranges of variables by generating various source data sets.

First, floating-point MATLAB code are executed, and then we obtain floating-point detected constellation fl_dt and ranges of every variables max_i , where

$$\max_i = \max(\text{abs}(Max_i), \text{abs}(Min_i)) \quad (5.4)$$

Next, by using the information of max_i , we can easily determine the integer length m_i by the following equation:

$$m_i = \text{floor}(\log_2(max_i)) + 2 \quad (5.5)$$

where one additional bit is used to represent sign bit.

Our experimental results of integer lengths are shown as follows.

Table 5.3: Experimental results of integer lengths

Variables	Parameter	Experimental Results
IFFT output data	m_{fft}	5
RRC coefficients	m_{RRC}	2
RRC output data in receiver	m_{rxRRC}	7
FFT output data	m_{fft}	7
Post-FFT long preamble chips	m_{mfft}	8
Channel estimator output data	m_{ch}	1
Phase estimator output data	m_{ph}	6

5.3.4 Determine Fraction Lengths

Since the integer word lengths are already decided and all variables are able to be covered by corresponding ranges, the rest of work is to decide the precision, that is, the fraction word lengths of every variable. To do so, we propose two modification methods, e.g., **coarse modification** and **fine modification**. Both of these two methods are able to find out a set of fraction lengths that can constrain quantization error within target error metric (EM_{target}) with minimized hardware resources usage. Coarse modification only roughly scales down all fraction lengths from a maximum precision

simultaneously to match the proposed EM_{target} without consideration to hardware resource weighting. On the other hand, fine modification scales fraction lengths each by each and calculates corresponding error metrics, and eventually finds out a set of fraction lengths, which system error metric is smaller than EM_{target} and can minimize the hardware resources with consideration to resource weightings. Usually, we run coarse modification first, and then set up the scaling ranges of variables in fine modification based on the results available in coarse modification. In the following sections, the ideas of these two methods will be detailed explained.

5.3.4.1 Coarse Modification

The purpose of coarse modification is to find out a set of fraction lengths which can be regarded as the reference when we deal with fine modification. First at all, the target error metric (EM_{target}) is decided, which is chosen to be 1, 5, and 10 in our experiment. Then, we start coarse modification, which criterion is to find out a set of minimum identical fraction lengths that corresponding error metric (EM_{coarse}) is smaller than the target error metric.

$$\forall i, p_i = p, \min p \text{ subject to } EM_{coarse} \leq EM_{target} \quad (5.6)$$

- STEP 1: For all i s, set fraction lengths (p_i) to be maximum precision (p_{max}), say 20 bits, in our experiment.
- STEP 2: Run fixed-point MATLAB code, and then obtain fixed-point detected constellation (fx_dt).
- STEP 3: Calculate coarse error metric (EM_{coarse}) by comparing floating-point detected constellation (fl_dt) with fixed-point constellation (fx_dt).
- STEP 4: If $EM_{coarse} < EM_{target}$ for all i s, set $p_i = p_i - 1$, and then redo step 2~4.
- STEP 5: If $EM_{coarse} > EM_{target}$ stop coarse modification.

Table 5.4 shows experimental results of fraction lengths available in coarse modification.

Table 5.4: Experimental results of coarse modification

	Coarse modification		
EM_{target}	1	5	10
EM_{coarse_STBC}	0.953	4.7553	9.83
EM_{coarse_VBLAST}	0.9871	4.6561	9.918
P_{ifft}	10	7	6
P_{RRC}	10	7	6
P_{rxRRC}	10	7	6
P_{fft}	10	7	6
P_{lnfft}	10	7	6
P_{ch}	10	7	6
P_{ph}	10	7	6
R	20522	16616	15314

5.3.4.2 Fine Modification

In fine modification, we attempt to take the hardware resource weighting (w_i) into consideration, and minimize the total hardware resource requirement (R). As shown in Eq. 5.7, we regard the product of the total word length ($m_i + p_i$) and the weighting factor (w_i) as the total hardware resource requirement.

$$R = \sum_i ((m_i + p_i) \times w_i) \quad (5.7)$$

The criterion of fine modification is shown as follows, where a set of fraction length which can minimize hardware resource requirement R and ensure the result error metric EM_{fine} being smaller than the target error metric EM_{target} is found.

$$\arg \min_{p_i} \underbrace{\sum_i ((m_i + p_i) \times w_i)}_R \quad \text{subject to } EM_{fine} \leq EM_{target} \quad (5.8)$$

- STEP 1: Start from the result fraction length (p_i) of coarse modification.
- STEP 2: Decide proper precision variances of these variables.
- STEP 3: Run fixed-point MATLAB code through these precision ranges, and then obtain fine error metric (EM_{fine}) and the total resource requirement (R).
- STEP 4: Find out a minimum R which satisfies $EM_{fine} < EM_{target}$.

Eventually, we can obtain final fraction lengths shown in Table 5.5. Notice that setting proper precision ranges is empirical, and it becomes more time-consuming however is able to reach global minimum of R more accurately as ranges are wider. The principle of setting ranges is giving the variables with larger w_s wider ranges, and giving the variables with smaller w_s narrower ones.

Table 5.5: Experimental results of fine modification

	Fine modification		
EM_{target}	1	5	10
EM_{fine_STBC}	0.94021	4.1791	8.7568
EM_{fine_VBLAST}	0.9622	4.496	9.626
P_{iFFT}	5	2	1
P_{RRC}	11	10	7
P_{rxRRC}	4	1	0
P_{fft}	7	4	4
P_{lnfft}	2	0	0
P_{ch}	11	8	6
P_{ph}	6	2	2
R	13808	10338	9268

5.4 Simulation Results

Figure 5.6 shows the STBC and VBLAST detected constellations under floating-point case, where no channel effects are involved.

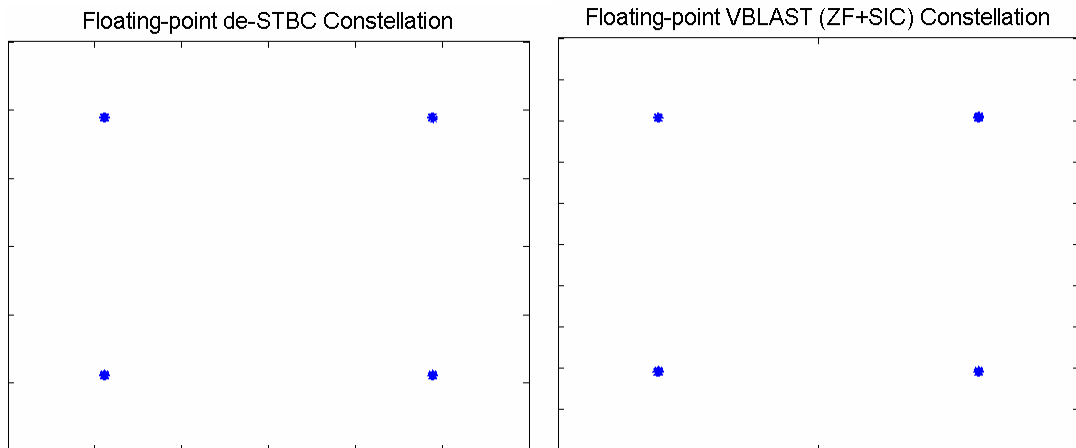


Figure 5.6: Detected constellation under floating-point case

Figure 5.7~5.9 show the other fixed-point detected constellations under different $EM_{target}S$.

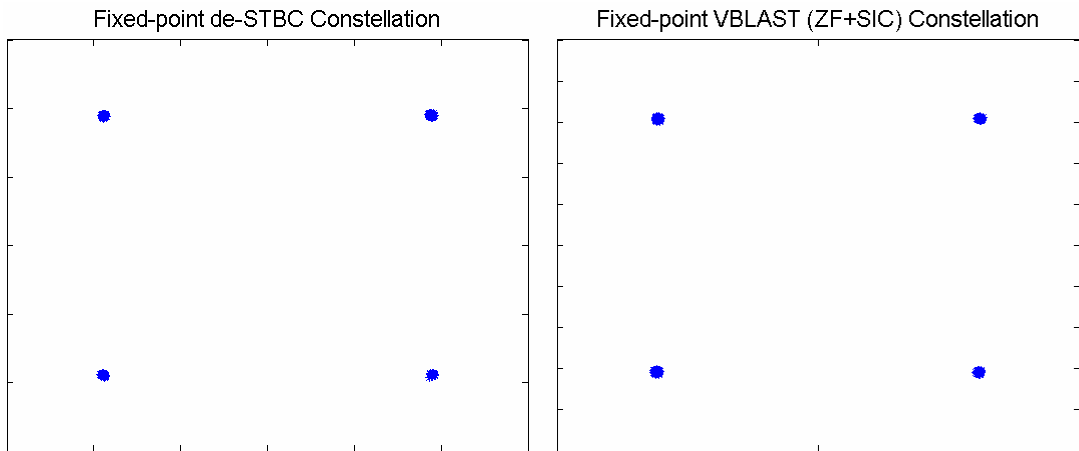


Figure 5.7: Detected constellation under $EM_{target} = 1$

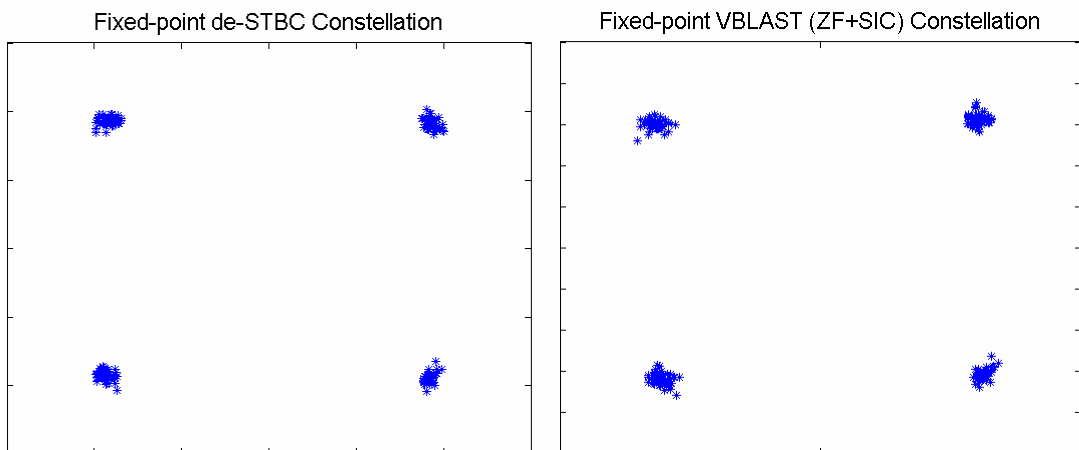


Figure 5.8: Detected constellation under $EM_{target} = 5$

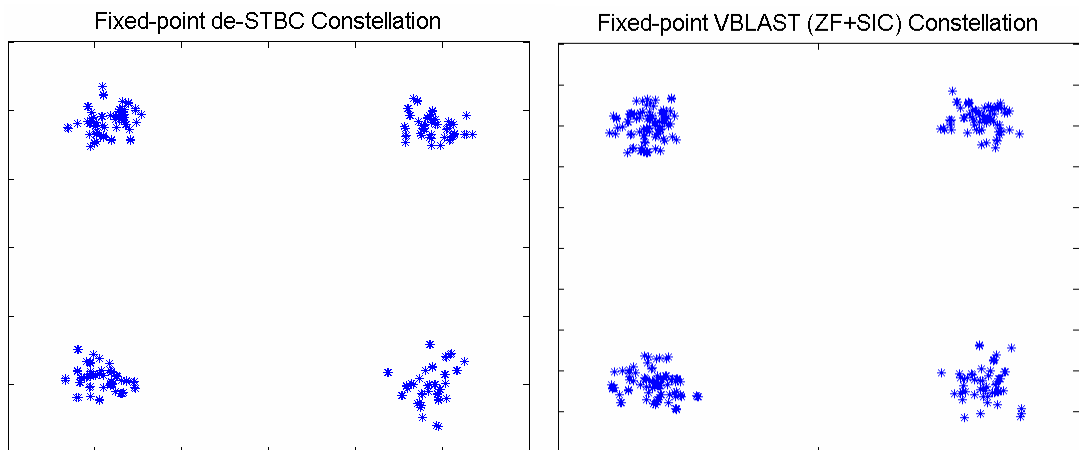


Figure 5.9: Detected constellation under $EM_{target} = 10$

Comparing with the above figures, it can be clearly observed that the quantization error does affect the detected constellations for both STBC and VBLAST cases. The constellations spread out more severely from the original four points in floating-point case as EM_{target} increases, which will cause detection error as channel effects are taken into consideration. To verify the effects of quantization error to system performance, we illustrate a BER to SNR plot under different EM_{target} s in Figure 5.10, where all system settings remain the same to floating-point case except the fixed-point value variables are carried out instead of the floating-point values. Notice that saturation is performed in IFFT output values and all the other quantizers follow Table 5.1.

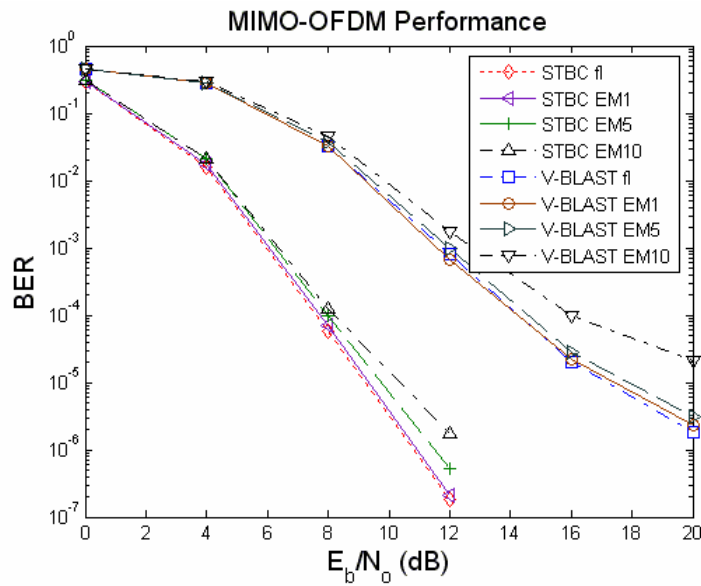


Figure 5.10: System performance under different EMs as q_{ifft} uses saturation

We can easily observe that for both STBC and VBLAST cases, the curves drift to right-upper side when the number of EM increases. It indicates that system performs worse as the level of quantization error increases. Furthermore, we can also observe that in low SNR, channel noise dominates the system performance, therefore the differences between EMs are not obvious. On the other hand, in high SNR, the quantization error noise dominates the system performance, therefore the gap between different EMs becomes bigger and bigger.

Additionally, in order to emphasize the importance of adopting saturation method to fight PAPR problem in IFFT, we perform another case in Figure 5.11, where original IFFT quantizer " $q_{ifft} = \text{quantizer}(\text{'fixed'}, \text{'floor'}, \text{'sat'}, [m_{ifft} + p_{ifft}, p_{ifft}])$ " shown in Table

5.1 is changed to be “ $q_{ifft} = \text{quantizer}(\text{'fixed'}, \text{'floor'}, \text{'wrap'}, [m_{ifft} + p_{ifft}, p_{ifft}])$ ”. That is, wrapping is applied instead of saturation. Clearly, both curves in Figure 5.11 drift to right-upper side much more severely comparing with Figure 5.10, and BER is saturated to about 10^{-5} as SNR increases. The reason is that wrapping is probable to let an extremely large positive value be a negative value or vice versa, which will cause severe quantization error comparing with saturation; that is why we intensively recommend using saturation instead of wrapping when dealing with PAPR problem in IFFT output data.

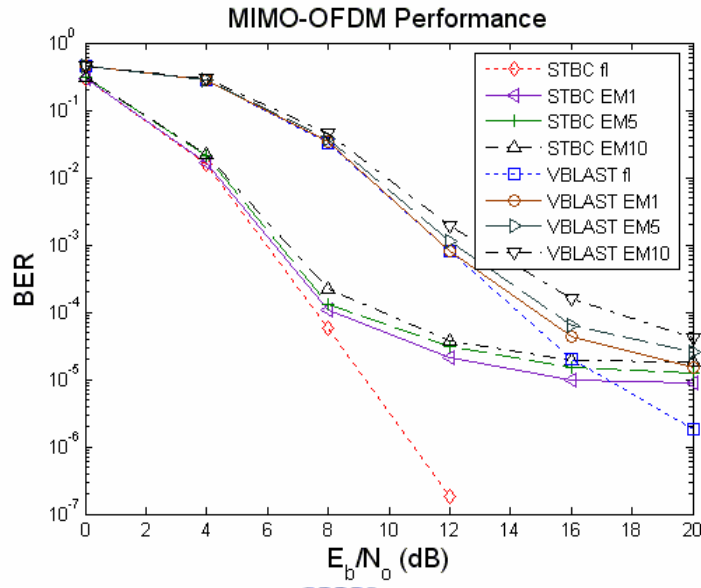


Figure 5.11: System performance under different EMs as q_{ifft} uses wrapping

Finally, total word lengths of variables and error metrics and hardware resource requirements under different EM_{target} s are categorized in Table 5.6. We can detect a trend from this table: additional 1070 LUTs are required as error metric is improved from 10 to 5, however more additional 3470 LUTs are required as error metric is improved from 5 to 1. That is, to achieve a zero quantization error system, the increase of additional hardware resource requirement will grow exponentially.

Table 5.6: Experimental results of final word lengths and EMs

EM_{target}	1	5	10
EM_{fine_STBC}	0.9402	4.1791	8.756
EM_{fine_VBLAST}	0.9622	4.496	9.626
$m_{ifft} + p_{ifft}$	10	7	6
$m_{RRC} + p_{RRC}$	13	12	9
$m_{rxRRC} + p_{rxRRC}$	11	8	7
$m_{fft} + p_{fft}$	14	11	11
$m_{lnfft} + p_{lnfft}$	10	8	8
$m_{ch} + p_{ch}$	12	9	7
$m_{ph} + p_{ph}$	12	8	8
R	13808	10338	9268

5.5 Summary

Since most practical FPGA designs are limited to finite precision signal processing using fixed-point arithmetic because of the cost and complexity of floating-point hardware, a systematical quantization algorithm is important for designers to map original floating-point code into fixed-point code. This chapter describes how the floating-point arithmetic in MATLAB are converted into fixed-point of specific precision for hardware design based on profiling the inputs, intermediate, and output signals. Especially, the idea of hardware resource weighting is inserted and the characteristics of communication system are also considered. Experimental results including integer lengths, fraction length, and total resource requirements under error metric equals 1, 5, and 10 are reported, and fixed-point system BER to SNR performances are also illustrated in this chapter.

Chapter 6

Conclusion

In future wireless communication systems, the demand of higher throughput and higher link quality is urgently called for, since various multimedia or home applications will be provided and thus reliable and affordable technologies are required to realize those contents. Coupled with a robust and efficient OFDM air interface, MIMO technologies can lead to a very attractive high-speed data transmission solution for future wireless systems. Recent years, researches on the topic of MIMO-OFDM system have been exploited greatly, and the MIMO-OFDM based standard, IEEE 802.11n, is just on the stage of competition for two proposals from TGn Sync and WWiSE, respectively. This encourages us to build up a hardware system based on MIMO-OFDM instead of the theoretical analysis only.

This thesis had described the signal processing concepts and algorithms of a 2×2 MIMO-OFDM system in physical layer, including STBC and VBLAST MIMO techniques. Furthermore, two FPGA based platforms are adopted to implement our 2×2 MIMO-OFDM system, e.g., fast prototyping platform and self-designed platform. In the fast prototyping platform, three FPGA modules, one DSP chip, and one USB module are installed; on the other hand, four FPGA modules, USB interface, and RF modules are equipped in the self-designed platform. A complete dataflow including software application interface, web camera, USB transmission, and baseband algorithms on DSP and FPGA are constructed in the fast prototyping platform; whereas a real wireless communication environment containing RF mismatch, multipath effects, and so on are generated through real indoor experimental environment and RF modules

on the self-designed platform. Finally, due to the complexity and time-consuming procedure of floating-point to fixed-point conversion, we have proposed a systematical quantization algorithm which can not only minimize the hardware resource requirement but also constrain quantization error within a specified limit.

To summarize, hardware implementation is highly complicated. Therefore, the availability of MATLAB simulation, proper quantization algorithms, useful HDL simulation software, and powerful debugging tools becomes especially significant. Nevertheless, some future works still remain. For example, higher modulation order such as 16QAM, 64 QAM and so on can be realized; furthermore, total power consuming issues ought to be taken into consideration, too. Finally, although there is a lot of room for improvement, we believe that the MIMO-OFDM system implemented on the FPGA-based platform we proposed is still highly advanced nowadays.



Bibliography

- [1] H. Yang, "A road to future broadband wireless access: MIMO-OFDM-based air interface," *Bell Labs Syst. Tech. J.*, vol. 1, pp. 41-59, Autumn 1996.
- [2] S. M. Alamouti, "A simple transmit diversity technique for wireless communication," *IEEE JSAC*, vol. 16, no. 8, pp. 1451–1458, Oct. 1998.
- [3] G. J. Foschini, "Layered space-time architecture for wireless communication in a fading environment when using multiple antennas," *Bell Labs Syst. Tech. J.*, vol. 1, pp. 41-59, Autumn 1996.
- [4] P. W. Wolnainsky, G. J. Foschini, G. D. Golden, and R. A. Valenzuela, "Detection algorithm and initial laboratory results using V-BLAST space-time communication architecture," *Electronics Letters*, vol. 35, pp. 14-16, Jan. 1999.
- [5] R. van Nee and R. Prasad, *OFDM for wireless multimedia communications*, Boston: Artech House, 2000.
- [6] R. D. Murch and K. B. Letaief, "Antenna systems for broadband wireless access," *IEEE Commun. Mag.*, vol. 40, pp. 76–83, Apr. 2002.
- [7] G. G. Raleigh and J. M. Cioffi, "Spatio-temporal coding for wireless communications," *IEEE Trans. Commun.*, vol. 46, pp. 357–366, Mar. 1998.
- [8] P. Vandenameele, L. V. D. Perre, M. G. E. Engels, and H. J. D. Man, "A combined OFDM/SDMA approach," *IEEE J. Sel. Areas Commun.*, vol. 18, pp. 2312–2321, Nov. 2000.
- [9] J. Kim and J. Cioffi, "Spatial multiuser access with antenna diversity using singular value decomposition," in *Proc. IEEE ICC*, vol. 3, pp. 1253–1257 2000.
- [10] "Wireless LAN medium access control (MAC) and physical layer (PHY) specifications: High-speed physical layer in the 5 GHz Band," *IEEE Std. 802.11a*, Sept. 1999.
- [11] A. van Zelst and T.C.W. Schenk, "Implementation of a MIMO OFDM-based wireless LAN system," *IEEE Trans. Signal Processing*, vol. 52, Issue 2, pp. 483 – 494, Feb. 2004.

- [12] A. N. Mody and G. L. Stüber, “Synchronization for MIMO OFDM systems,” in *Proc. IEEE Global Commun. Conf.*, vol. 1, pp. 509–513, Nov. 2001.
- [13] Y. Hara and Y. Kamio, “Initial synchronization techniques for antenna arrays in the presence of interference signals,” *IEEE Conf. Vehicular Technology*, vol. 4, pp. 1953-1957, Sept. 2002.
- [14] V. Tarokh, N. Seshadri, and A. R. Calderbank, “Space-time codes for high data rate wireless communication: Performance criterion and code construction,” *IEEE Trans. Inform Theory*, vol. 44, pp. 744–756, Mar. 1998.
- [15] G. J. Foschini and M. J. Gans, “On limits of wireless communications in a fading environment when using multiple antennas,” *Wireless Pers. Commun.*, vol. 6, no. 3, pp. 311–335, Mar. 1998.
- [16] A. van Zelst, “Space division multiplexing algorithms,” in *Proc. 10th Mediterranean Electrotech. Conf.*, vol. 3, 2000, pp. 1218–1221.
- [17] M. Sellathurai and S. Haykin, “A nonlinear iterative beamforming technique for wireless communications,” in *33rd Asilomar Conf. on Signals, Syst., and Comput.*, vol. 2, pp. 957–961, Pacific Grove, CA, Oct. 1999.
- [18] D. Lashin and B. Cisneros, “System explorer MP3C reference guide,” Aptix Inc., August 1999.
- [19] J. Bhasker, *A VHDL Primer*, Englewood Cliffs, NJ: Prentice-Hall, 1998.
- [20] 林傳生，*使用VHDL 電路設計語言之數位電路設計*，儒林，2000。
- [21] 國家晶片系統設計中心，*Xilinx (PC)*, July 2004.
- [22] 國家晶片系統設計中心，*VHDL*, July 2004.
- [23] 鄭信源，*Verilog 硬體描述語言數位電路設計實務*，儒林，2000。
- [24] Taxes Instrument, “Code composer studio getting started guide,” spru509, May 2001.
- [25] Cypress Semiconductor Corporation, “CY768013 / EZ-USB FX2 USB micro-controller / High-speed USB peripheral controller,” June 2002.
- [26] L.B. Jackson, “On the interaction of roundoff noise and dynamic range in digital filters,” *Bell System Technical J.*, pp. 159-183, Feb. 1970.
- [27] A.V. Oppenheim, R.W. Schaffer, and J.R. Buck, *Discrete-Time Signal Processing*, second ed. Prentice Hall, 1999.

- [28] K.H. Chang and W.G. Bliss, "Finite word-length effects of pipelined recursive digital filters," *IEEE Trans. Signal Processing*, vol. 42, no. 8, pp. 1983-1995, Aug. 1994.
- [29] L.B. Jackson, K.H. Chang, and W.G. Bliss, "Comments on 'Finite word-length effects of pipelined recursive digital filters,'" *IEEE Trans. Signal Processing*, vol. 43, no. 12, pp. 3031-3032, Dec. 1995.
- [30] R.M. Gray and D.L. Neuhoff, "Quantization," *IEEE Trans. Information Theory*, vol. 44, no. 6, pp. 2325-2383, Oct. 1998.
- [31] C. Shi, "Statistical method for floating-point to fixed-point conversion," MS thesis, Electrical Eng. and Computer Science Dept., Univ. of California, Berkeley, 2002.
- [32] W. Sung and K.I. Kum, "Simulation-based word-length optimization method for fixed-point digital signal processing systems," *IEEE Trans. Signal Processing*, vol. 43, no. 12, Dec. 1995.
- [33] S. Kim and W. Sung, "Fixed-point error analysis and wordlength optimization of a distributed arithmetic based 8X8 2D-IDCT architecture," *Proc. Workshop VLSI Signal Processing*, IX, pp. 398-407, 1996.
- [34] H. Keding, M. Willems, M. Coors, and H. Meyr, "FRIDGE: A fixed-point design and simulation environment," *Proc. Design, Automation, and Test in Europe*, pp. 429-435, 1998.
- [35] S. Roy, and P. Banerjee, "An algorithm for trading off quantization error with hardware resources for MATLAB-based FPGA design," *IEEE Trans. Computers*, vol. 54, Issue 7, pp. 886 – 896, July. 2005.