TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SAJJAD NOURI

DESIGN AND IMPLEMENTATION OF SOFTWARE DEFINED RADIO ACCELERATORS USING AN ADAPTIVE COARSE-GRAIN RECONFIGURABLE ARRAY AND PROCESSOR SOFTWARE

Master of Science Thesis

Examiners: Prof. Jari Nurmi
Dr. Waqar Hussain

Examiners and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 8th of April 2015

ABSTRACT

SAJJAD NOURI: Design and Implementation of Software Defined Radio Accelerators Using An Adaptive Coarse-Grain Reconfigurable Array and Processor Software
Tampere University of Technology
Master of Science Thesis, 67 pages
May 2015
Master's Degree Programme in Information Technology
Major: Communication Systems and Networks
Examiners: Prof. Jari Nurmi
Dr. Waqar Hussain
Keywords: Software Defined Radio, WLAN, OFDM, RISC Processors, CGRA, COFFEE, CREMA, Run-time Reconfiguration, Time Synchronization, Correlation, Frequency Offset Estimation, Channel Estimation, Demapping, FPGA

Over the past few decades, the development of wireless communication systems in both hardware and software calls for the speed-up in the execution of the involved functions. Moreover, in embedded systems which are including different types of communication systems, a large number of computations yet with short execution time are needed while power consumption is required to be minimized. There is an increasing demand to use application-specific accelerators assisting general-purpose RISC processors.

This thesis focuses on designing the application-specific accelerators for Orthogonal Frequency Division Multiplexing (OFDM) IEEE 802.11a receiver blocks using CREMA (Coarse-grain REconfigurable array with Mapping Adaptiveness). At first, some of the common techniques used in OFDM receivers are presented. Then, the basic structure of COFFEE RISC processor as the main implementation platform is described. In addition, the definition of different reconfigurable architectures has been discussed. The experimental part of this research work covers the design and implementation of three different application-specific accelerators for OFDM receiver blocks. The accelerators work particularly for COFFEE RISC core firmly integrated with a Direct Memory Access (DMA) device.

The performance of the accelerators is evaluated in terms of the number of clock cycles, resource utilization and synthesis frequency on an Altera Stratix-IV Field Programmable Gate Array (FPGA) device. It is observed that the designed accelerators give speed-up of $4.8\times$ to $18.6\times$ in comparison with COFFEE RISC processor software.

PREFACE

This thesis work has been completed in the Department of Electronics and Communications Engineering at Tampere University of Technology to pursue Master of Science degree in the program of Information Technology.

First and foremost I would like to express my sincere gratitude to my supervisor, Professor Jari Nurmi, who made possible for me to do my research work at Tampere University of Technology. His expertise, patience, understanding, and motivation added considerable value to my master thesis. Furthermore, I appreciate his support that helped me to manage the accomplishment of my master thesis. I thank Dr. Waqar Hussain, who guided me in my first steps and for all his support and advices that led to accomplish this thesis. In addition, I would like to express all my deepest acknowledgments to Professor Markku Renfors and Jukka Rinne for their support and guidance, who granted me useful feedback whenever I asked and provided valuable and insightful comments to improve this research work.

I would like to extend my appreciation to all my friends at Tampere University of Technology in particular Farid Shamani, Orod Raeesi, Nader Daneshfar, Mona Aghababaee, and Vida Fakour Sevom for their warm friendship and nice moments we have shared together and also, for their support throughout my master's degree program.

I am also grateful to my dear friends, Mohsen Shahshahan, Farid Mehrabkhani, Alireza Zare, Ramin Ghaznavi, Zeinab Ashjaei, Zahra Abbaszadeh, Amir Fard, Armin Mousavi, Morteza Rajabpoor, Mehdi Joafshani, Keyvan Abdi, Ata Meshkinzar, and Mohammad Alitavoli for helping me to stay positive and focused. Thank you for your long support and friendship.

Finally, I wish to express my deepest gratitude and respect to my mother Tayyebeh Sadeghi Moghadam and my father Majid Nouri for their patience, enormous love and invaluable support in all moments throughout my entire life to bring me up to this stage. I owe all my achievements to them and without their support, none of this would have been possible. Also, I am grateful to my grandparents for their unexplainable support during my life.

Tampere, May 2015
Sajjad Nouri

I dedicate this thesis to my parents, whose love and affection, encouragement and prayers of day and night make me able to get such success and honor.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

ACK	ACKnowledgment
ADC	Analog-to-Digital Converter
AGC	Automatic Gain Control
ALU	Arithmetic Logic Unit
ALUT	Advanced Look-Up Table
ASIC	Application Specific Integrated Circuit
ASK	Amplitude Shift Keying
ATM	Asynchronous Transfer Mode
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
CC	Clock Cycle
CCB	Core Configuration Block
CFO	Carrier Frequency Offset
CGRA	Coarse Grain Reconfigurable Array
CIR	Channel Impulse Response
CISC	Complex Instruction Set Computing
CP	Cyclic Prefix
CPU	Central Processing Unit
CREMA	Coarse grain REconfigurable array with Mapping Adaptiveness
DAC	Digital-to-Analog Converter
DC	Delay and Correlate
DFT	Discrete Fourier Transform
DMA	Direct Memory Access
DSP	Digital Signal Processor
FPU	Floating Point Unit
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FSK	Frequency Shift Keying
FSM	Finite State Machine
FU	Functional Unit
Gbps	Giga Bit Per Second
GCC	GNU Compiler Collection
GI	Guard Interval
GOPS	Giga Operation Per Second
GPP	General Purpose Processor

GUI	Graphical User Interface
HDD	Hard Decision-based Detection
HDL	Hardware Description Language
HW	HardWare
I	In-Phase
IEEE	Institute of Electrical and Electronics Engineers
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse Fast Fourier Transform
I/O	Input/Output
IP	Intellectual Property
I/Q	In-phase and Quadrature-phase
ISI	Inter-Symbol Interference
LTS	Long Training Symbols
LUT	Look Up Table
MAC	Medium Access Control
MCM	Multi-Carrier Modulation
ML	Maximum Likelihood
MT	Mobile Terminal
MVM	Matrix Vector Multiplication
NOP	No-OPeration
NoC	Network-on-Chip
OFDM	Orthogonal Frequency Division Multiplexing
PAPR	Peak-to-Average Power Ratio
PCB	Peripheral Control Block
PE	Processing Elements
PN	Pseudo Noise
PSDU	Physical layer Service Data Unit
PSK	Phase Shift Keying
PTS	Partial Transmit Sequences
Q	Quadrature-phase
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
RF	Radio Frequency
RISC	Reduced Instruction Set Computing
RPU	Reconfigurable Processing Unit
RTL	Register Transfer Level
RTOS	Real-Time Operating System
SDD	Soft Decision-based Detection
SDR	Software Defined Radio

SER	Symbol Error Rate
SLM	SeLected Mapping
SNR	Signal-to-Noise Ratio
SoC	System-on-Chip
STS	Short Training Symbols
TUT	Tampere University of Technology
VC	Virtual Carrier
VHDL	Very-high-speed integrated circuit Hardware Description Language
VLIW	Very Long Instruction Word
WiFi	Wireless Fidelity
WLAN	Wireless Local Area Network

1. INTRODUCTION

Currently, mobile communication systems are one of the most fruitful markets for embedded processors. All users expect their smart phones to work perfectly without any interruptions. In order to have short execution time and high production volumes, reconfigurability plays a significant role for embedded systems. Many embedded applications require a large number of computations every second while power consumption needs to be minimized. In addition, it could be observed that as computational requirements are increasing, RISC processors are not able to provide all kind of computations especially for multimedia and telecommunication applications. Therefore, there is an increasing demand to use application-specific accelerators coupled to general-purpose RISC processors. Moreover, application-specific accelerators can diminish the computation time while operating at low frequencies. In general, accelerators are required in computer hardware to perform a single computationally intensive task. In other words, by utilizing accelerators, there is no need to implement the instructions one by one in processors and the overall computational power of a processor system is increased. Furthermore, in embedded systems, accelerators are essential because of the limited area and energy. Basically, there are two kinds of accelerators for general purpose microprocessors: general-purpose accelerators and run-time reconfigurable accelerators. Accelerators can differ from one another which can be related to their designing objective, implementation technology, interface for communicating with the other parts of a system and architecture.

Reconfigurable architectures are one of the most successful platforms containing different levels of configurability and parallelism. Dynamic reconfigurability allows behavior and functionality at the run-time for several applications. The functionality of reconfigurable devices can be specified by a user at system design time and can be changed at runtime, therefore permits to add functionality to the same chip without using more silicon . Reconfigurable systems are characterized by their granularity, programmability, reconfigurability (either static or dynamic), interface and computation model [1]. The reconfigurable devices can be classified according to the level of granularity into Fine-Grain, Middle-Grain and Coarse-Grain [2].

During recent years, Coarse-Grain Reconfigurable Array (CGRA) has become a popular platform for several research groups due to its high level of granularity which allows us to map 8-, 16- and 32-bit arithmetic or logic operation onto a single Processing Element (PE). Coarse-grain architecture is a favorable solution for industrial and academic environments because of its energy consumption and programmability. The product of total power consumption and the execution time is equal to the energy consumption which is low for CGRAs as they are not active most of the time. In addition, an array of predefined Processing Elements (PEs) is used in CGRAs to provide high computational power. Since an application can be written entirely in C, its programming is much easier for an application developer. CGRA is a 2D array of PEs that are either connected using a Network-on-Chip (NoC) or dedicated point-to-point connections. CGRAs provide high level of data parallelism and throughput because of the symmetry in their structure [3]. Moreover, CGRAs are ideal for research groups to be used for digital signal processing applications and processing of streams. On the other hand, CGRAs are very resource demanding due to the large amount of interconnections and operations. Considering this issue, CREMA (Coarse-grain REconfigurable array with Mapping Adaptiveness) was introduced so that the designers can assign those resources which are required for a particular application [4]. CREMA is developed to work as an accelerator with general-purpose RISC processor in order to have easiness for producing application-specific accelerators while flexibility is conserved. In other words, they are integrated together to work in a processor/coprocessor model to yield the benefits of general- and special-purpose processing.

1.1 Objective and Scope of Thesis

This research aims to generate application-specific accelerators for Software Defined Radio (SDR) base-band prcoessing using CREMA template. CREMA is a suitable candidate for wireless applications as a powerful computing engine that is tightly coupled with general-purpose COFFEE RISC processor. COFFEE is an open source 32-bit Reduced Instruction Set Computing (RISC) processing core [5]. The main objective of this thesis work is to design and implement special-purpose accelerators for Orthogonal Frequency Division Multiplexing (OFDM) receiver baseband processing on CREMA platform. In addition, the performance of the designed accelerator for each block of OFDM receiver is evaluated in terms of the number of clock cycles, resource utilization and maximum operating frequency by synthesizing on Altera Stratix-IV family of Field Programmable Gate Arrays (FPGA). In an OFDM transciever, the processing of some blocks are too computationally-intensive to be processed by a processor core. For instance, at the receiver side of IEEE 802.11a [10], there is need to compute Time Synchronization and Fast Fourier Transform (FFT) for each received data symbol which are computationally very intensive and demand high computational power from a processor. As the first step of this work, the baseband processing of OFDM receiver is described. Then some limitations are identified in order to design accelerators based on CREMA for different applications. For instance, there are no predefined functions to compute some mathematical operations related to this work (e.g., division and `ATAN`) in COFFEE RISC core. Accordingly, some parts of OFDM receiver are implemented in software on COFFEE RISC core by using different algorithms, e.g., Taylor series. Furthermore, division process is implemented using two different algorithms on both CREMA platform and processor software.

1.2 Thesis Outline

This thesis is organized as follows; Chapter 2 describes the basic structure of OFDM WLAN receiver baseband signal processing under IEEE 802.11a specifications. In addition, different methods for each block of the OFDM receiver are explained. In chapter 3, several examples of CGRA are presented as a literature review. Chapter 4 presents a survey of COFFEE platform architecture and structure of CGRAs, in particular CREMA. Chapter 5 explains the mapping of several applications on CREMA and execution details of baseband algorithms. Moreover, the performance of each accelerator is evaluated in terms of different metrics using both simulation and synthesis results are also presented. Finally in Chapter 6, concluding remarks and future work is presented.

2. OFDM WLAN OVERVIEW

OFDM is one of the most popular digital multi-carrier modulation techniques for achieving higher data rate. By using OFDM technique, there is a possibility to cope with attenuation of high frequencies in a long wire, Inter-Symbol Interference (ISI) and frequency selective fading because of multipath propagation in wireless communication. It should be mentioned that ISI can be reduced by transmitting several symbols in parallel and increasing the symbol duration. Moreover, frequency selective fading issue could be resolved by converting frequency-selective channel into several adjacent flat fading sub-channels. OFDM is breaking higher bit rate encoded data stream into several lower rate ones and sending them on different sub-carriers in parallel while orthogonality is kept between them [6]. This operation can be easily done in the transmitter by using N-point Inverse Fast Fourier Transform (IFFT) [7]. Unlike single carrier system, OFDM is a mixture of Multi-Carrier Modulation (MCM) and Frequency Shift Keying (FSK) [8]. FSK means that data are transmitted on one carrier where there is a set of orthogonal carriers. The most important advantages of OFDM, as can be understood from its name, is orthogonality between subcarriers. Orthogonality can be attained by using IFFT for modulation and isolating the carriers by an integer multiple of the inverse of the symbol duration. In order to preserve orthogonality, transmitter and receiver must be in same modulation frequency and time-scale. One of the advantages of OFDM, as it is shown in Figure 2.1, is saving the bandwidth where adjacent sub-channels are overlapped with each other.

As other single-carrier and multiple access methods, OFDM has some advantages. Main ones are listed as following [7]

- Robustness against multipath fading
- High spectral efficiency
- Interference elimination by using cyclic prefix
- Narrow-band interference mitigation that may occur due to the radio non-linearities

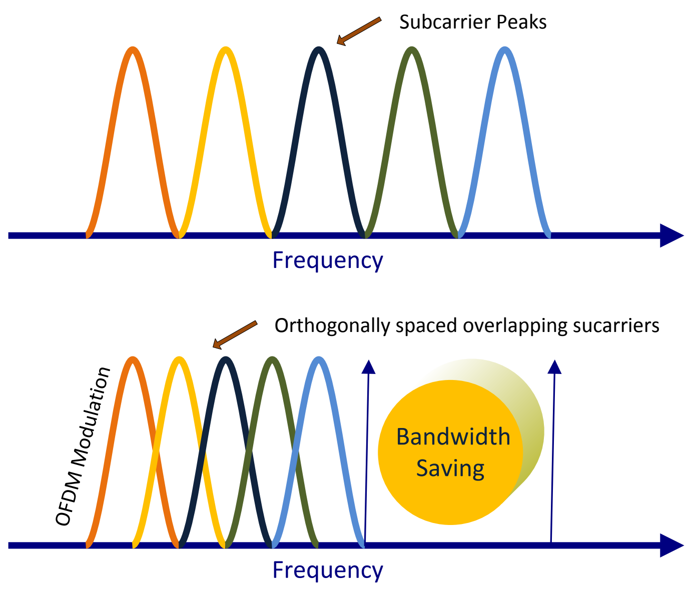


Figure 2.1 *Non-overlapping and overlapping multi-carrier modulation*

- Adaptive modulation

On the other hand, there are some drawbacks for OFDM like: [7]

- Sensitivity to phase noise and frequency synchronization errors
- High Peak-to-Average Power Ratio (PAPR) which can be reduced by using different methods like Selected Mapping (SLM), Partial Transmit Sequences (PTS), Tone Injection and Tone Reservation [9].

This chapter is organized as follows: during first two sections, description of MAC frame structure of OFDM and its physical layer specifications are provided. Furthermore, the general structure of OFDM including the transmitter, channel and the receiver will be discussed. After that, we will proceed with an explanation of each OFDM receiver block, considering IEEE 802.11a standards specifications.

2.1 MAC Frame Structure for WLAN Standards

Presently, there are three accepted WLAN standards in the world which are different from each other in terms of Medium Access Control (MAC). These standards are listed in Table 2.1 [8]. The first two are used in Europe and North America and the last one is utilized in Japan. Moreover, since we are using the most widely used MAC, IEEE 802.11, it is described in detail in the following.

***Table 2.1** OFDM WLANs Standards*

S/No	Standard	S/No	Type of MAC
1.	IEEE 802.11a	1.	Distributed MAC based on Carrier Sense Multiple Access with Collision Avoidance
2.	HiperLAN/2	2.	Centralized and scheduled MAC based on wireless Asynchronous Transfer Mode (ATM)
3.	MMAC	3.	Both of mentioned MACs

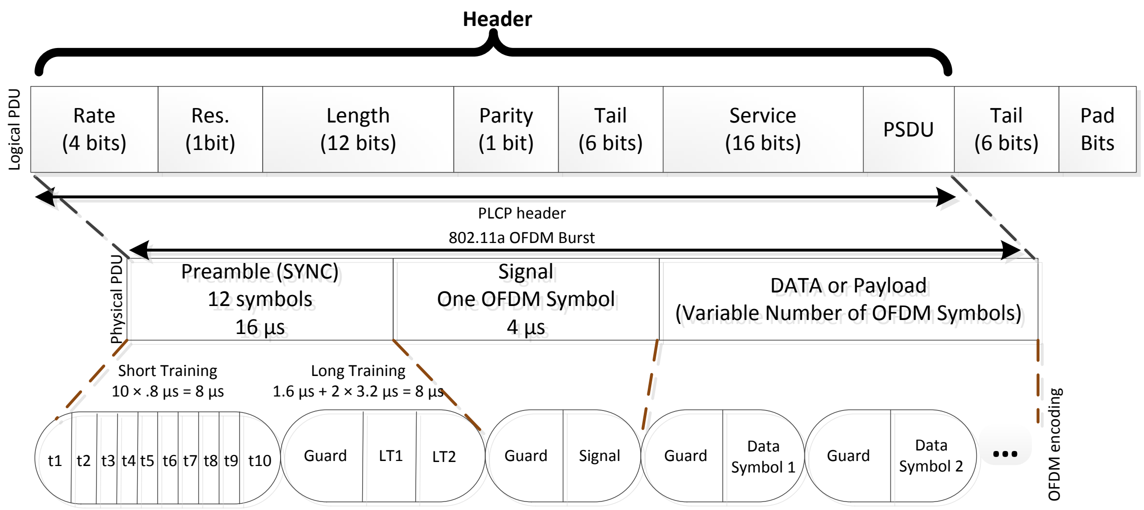


***Figure 2.2** PLCP Protocol Data Unit (PPDU) in 802.11a ©IEEE, 1999*

Figure 2.2 shows the MAC frame structure of IEEE 802.11a. In order to access to the network, Mobile Terminal (MT) ask data from the Access Point (AP). After transmitting the packet, MT has to wait for an acknowledgment (ACK) frame which is necessary for avoiding collisions. The header file of received packet composed of information about the transmission rate, the length of the payload and is transmitted via Binary Phased Shift Keying (BPSK) which is a modulation technique and will be discussed later in detail [8]. In the following, the brief definition of header parameters can be seen:

- Rate: Type of modulation and coding rate of the entire packet
- Length: Number of bytes in Physical Layer Service Data unit (PSDU) that might be varied between 1 and 4095
- Tail: Return the convolutional encoder to the ”zero state” [10] and play out the code trellis in the decoder
- Service: The bits from 0-6 are set to zeros and are used for synchronizing the descrambler, the last 9 bits are reserved for the future.

PLCP header consists of a preamble, signal and data field. There are 10 short training symbols and 2 long training symbols in the preamble which can be used for packet detection, time synchronization, frequency offset estimation and channel estimation. Totally, preamble is composed of predefined samples which are known to the receiver and can be used for synchronization purposes. As can be seen from Figure 2.2, length of both training symbols is 8.0 μs with the total time of 16.0 μs.

Short Training Sequence (STS) is composed of ten short symbols with 12 subcarriers, each of them based on specific repetitions (every 4th subcarrier has equal magnitude) given in frequency domain in Equation 2.1. The reasons behind this choice are good correlation properties and low peak-to-average power ratio. Also, in the transmitter, a 64-point IFFT is required in order to create a time domain sequence. Since STS is known for the receiver, it can be used in different blocks like packet detection or time acquisition by using its correlation peaks properties [11]. Moreover, it is needed for frequency offset estimation because of repetition of samples which will be described it in more detail later.

$$\begin{aligned} S_{-26,26} = \sqrt{\frac{13}{6}}\{&0,0,1+j,0,0,0,-1-j,0,0,0,1+j,0,0,0,-1-j,0,0,0,-1-j,\\ &0,0,0,1+j,0,0,0,0,0,0,0,-1-j,0,0,0,-1-j,0,0,0,1+j,0,0,\\ &0,1+j,0,0,0,1+j,0,0,0,1+j,0,0\} \end{aligned} \tag{2.1}$$

Long Training Sequence (LTS) is an another preamble sequence with 53 subcarriers in each one of two 3.2 μs long training symbols that can be seen in Equation 2.2. Furthermore, 1.6 μs Guard Interval (GI) is needed between short and long training symbols for combating with Inter-Symbol Interference (ISI). LTS is used in more accurate time synchronization and fine frequency offset estimation [11].

$$\begin{aligned} L_{-26,26} = \{&1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,\\ &0,1,-1,-1,1,1,-1,1,-1,1,-1,-1,-1,-1,-1,1,1,-1,-1,1,-1,1,-1,\\ &1,1,1,1\} \end{aligned} \tag{2.2}$$

The next field in the header of PLCP is the signal field, which has information about Rate and Length of the TXVECTOR. The rate is used for representing the type of modulation and encoding. This single OFDM symbol is always BPSK-modulated and its duration is equal to 4.0 μs. It should be mentioned that this field can be found just in 802.11a standard [10].

The next and most important part is the data field with variable number of OFDM symbols (depends on the modulation type). There are 52 subcarriers in each data symbol which are composed of 48 data subcarriers and 4 pilot subcarriers. Furthermore, there is one IFFT per symbol with the length of 64, thus, each data has 12 unused subcarriers. Pilots shall be put in subcarriers -21, -7, 7 and 21 while the type of their modulation is always BPSK in order to avoiding the generation of spectral lines. In addition, data modulation could be BPSK, QPSK, 16-QAM and 64-QAM which are same for each burst. The contribution of pilot subcarriers can be observed in Equation 2.3.

$$\begin{aligned} P_{-26,26} = \{&0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0, \\ &0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,-1,0,0,0,0,0\} \end{aligned} \tag{2.3}$$

The frequency spacing of each subcarrier is equal to 312.5 kHz (20 MHz for all 64 subcarriers). In the following, Table 2.2 lists the timing parameters related to 802.11a signal [10].

Table 2.2 *IEEE 802.11a Timing Analysis*

S/No	Parameter	Value	Description
1.	Symbol Interval Time	4.0 μs	$(T_{GI} + T_{FFT})$
2.	Data Interval Time	3.2 μs	$1/F_{SP}$
3.	IFFT/FFT Duration	3.2 μs	$1/F_{SP}$
4.	SIGNAL Symbol time	4.0 μs	$(T_{GI} + T_{FFT})$
5.	Training Symbol GI Duration	1.6 μs	$(T_{FFT}/2)$
6.	Preamble	16.0 μs	$(T_{SHORT} + T_{LONG})$
7.	Short Training Sequence	8.0 μs	$(10 \times T_{FFT}/4)$
8.	Long Training Sequence	8.0 μs	$(T_{GT} + 2 \times T_{FFT})$
9.	Guard Interval Duration	0.8 μs	$(T_{FFT}/4)$

Furthermore, it should be noticed that for each data symbol, the maximum number of bits per each frame is equal to 4096 which means 1024 samples or 16 symbols.

2.2 Physical Layer Specifications for WLAN Standards

Presently, OFDM is used in many recently standardized broadband communication systems toward combating with frequency-selective fading. During this section, it can be seen that what will be occurred exactly for the data in OFDM transmitter, channel and especially in the receiver. OFDM is working at 2.4 GHz operating frequency that enables data transmission at a rate of 6, 9, 12, 18, 24, 36, 48, or 54 Mbps with 1/2, 9/16, 2/3 and 3/4 coding rate [8]. The simplified version of OFDM transceiver is shown in Figure 2.3.

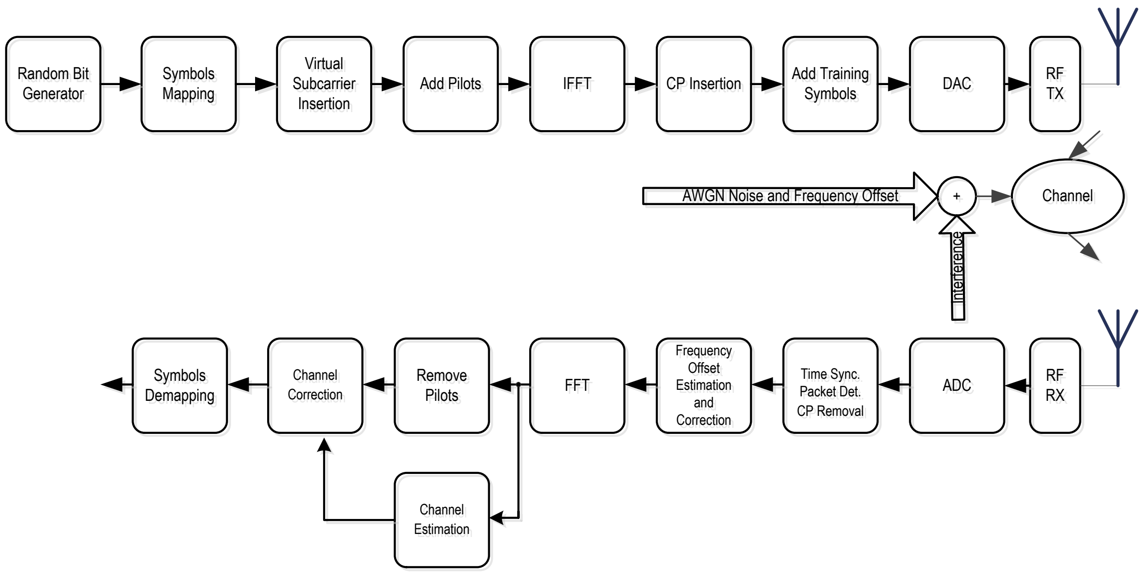


Figure *2.3 IEEE 802.11a simplified transceiver architecture*

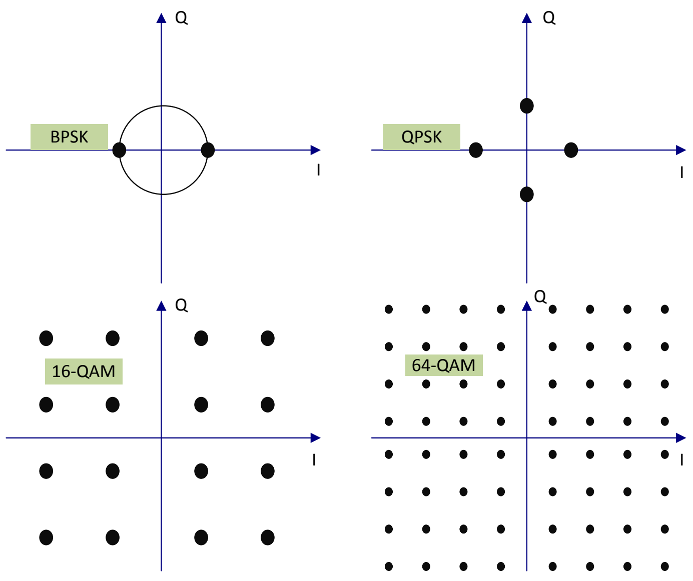


Figure *2.4 IEEE 802.11a constellations*

After generating data randomly, QAM mapper is the first block in the transmitter. As it is mentioned before, there are different constellations which are used in IEEE 802.11a standard and can be observed from Figure 2.4. The modulations are two-dimensional for using both In-phase (I) and Quadrature (Q) carrier waves and can be implemented by changing the amplitude, phase or frequency, while the last one is unused in OFDM systems because of destroying the orthogonality. Thus, in IEEE 802.11a, there are two methods for doing modulation: Phase shift keying (PSK) and Quadrature Amplitude Modulation (QAM). In PSK, information is transmitted by altering the phase of the carrier waveform that is shown in Equation 2.4 where s(t)

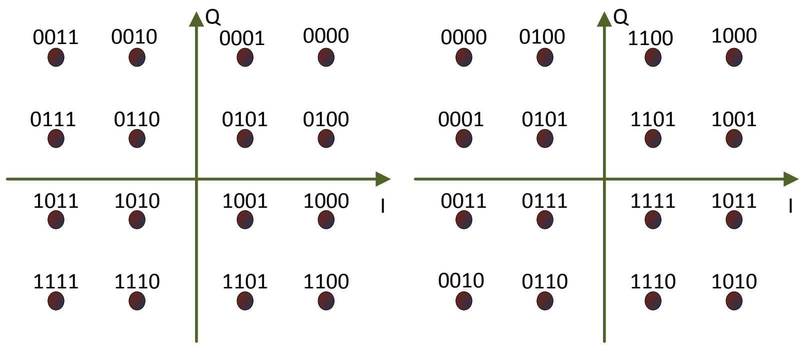


Figure 2.5 *QAM natural order (left) and Gray coding (right)*

is a transmitted signal. The benefits of PSK is the PAPR (equal to 1) and simplified RF design for transceiver [8].

$$s(t) = cos(\omega t + \phi_k) \tag{2.4}$$

QAM is the composition of ASK and PSK, it means QAM changes both amplitude and phase of the carrier as can be observed in Equation 2.5. Furthermore, amplitude and phase of carriers can be calculated from Equation 2.6 and Equation 2.7.

$$\begin{aligned} s(t) &= I_k cos(\omega_c t) - Q_k sin(\omega_c t) \\ &= A_k cos(\omega t + \phi_k) \end{aligned} \tag{2.5}$$

$$A_k = \sqrt{I_k^2 + Q_k^2} \tag{2.6}$$

$$\phi_k = tan^{-1}\left(\frac{Q_k}{I_k}\right) \tag{2.7}$$

All constellation points must be labeled by assigning a bit pattern (mapping). This issue can be done in two ways: natural order or Gray coding which is shown in Figure 2.5. The difference between natural coding and Gray one is that the first one assigns decimal numbers from 0 to 15 in order, but in the second one, all samples are different from the adjacent one in just one bit. Hence, two-bit errors (most common type of error) for symbol errors between neighboring points can be reduced by using Gray coding, which means decreasing bit error rate (BER) and symbol error rate (SER) [8].

Once all data bits are mapped to the specific bit pattern, virtual subcarrier insertion block adds 12 unused subcarriers based on the standard prior to the 64-point IFFT. That means zero padding is implemented by inserting 6 zeros to both sides of data. Then, as it is discussed before, four pilot symbols are added to data. The next block

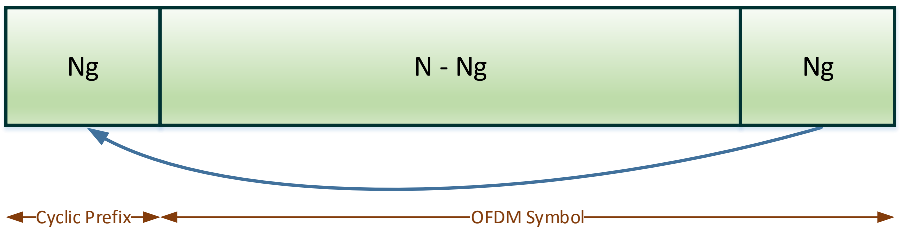


Figure 2.6 *Adding Cyclic Prefix in OFDM Symbols*

and one of the most important ones is inverse FFT (IFFT) that is explained in the following briefly.

Once solution to many engineering problems is using Fourier series as a tool in order to predict the output of the linear time-invariant system by breaking up the input signal into simple signals and knowing how the system responds to these simple signals. The Fourier Transform is an extension of the Fourier series that can be applied to continuous and periodic functions. Given a sequence of N samples *f(n)* (time domain), Discrete Fourier Transform (DFT) is defined as *F(k)* (frequency domain) which is shown in Equation 2.8. In other words, DFT converts the signal from the time domain to frequency domain. This procedure can be reversed in order to calculate *f(n)* from *F(k)* by using IDFT that is shown in Equation 2.9 [7].

$$F_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f(n) e^{-j2\pi kn/N} \tag{2.8}$$

$$f_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} F(k) e^{j2\pi nk/N} \tag{2.9}$$

Thus, IFFT is implemented on the frequency domain QAM subcarriers to produce time domain sum of sinusoids. In addition, it is the simplest way to control the amplitude and phase of the subcarriers in the frequency domain and modulate data onto orthogonal subcarriers.

Once IFFT is performed, Guard Interval (GI) or Cyclic Prefix (CP) is added to the output of IFFT [8]. As can be observed from Figure 2.6, to add the cyclic prefix, 16 samples (0.8 μs) from the end of the OFDM symbol are appended to the beginning of the OFDM symbol. The cyclic prefix is introduced in order to cope with Inter-Symbol Interference (ISI). ISI is essentially caused by receiving several copies of the transmitted signal due to multipath effect and dispersion of the channel [12]. Let us assume that there are two OFDM symbols, it can be seen that the last portion of the first OFDM symbol creates interference with the first portion of second OFDM symbol upon it is received, thus amplitude and phase of subcarriers might be deviated. For this reason, the cyclic prefix is really critical in order to solve this problem and its length must be more than delay spread. Accordingly, delayed portion of the first OFDM symbol can be absorbed via cyclic prefix of the second OFDM symbol [13].

After cyclic prefix addition, IEEE 802.11a preambles are generated, which are composed of short and long training symbols. Before transmitting the signal over air interface by using an antenna, the signal must be converted from digital domain to analog one via Digital to Analog Converter (DAC). Also, it should be noticed that upon samples go through the DAC, a reconstruction filter is needed in order to remove replication of the spectrum. The filter design is becoming much easier if there is oversampling by a factor of two because of the reason that after it, spectra replicas are much further apart. Oversampling can be done by using null subcarriers which should be located around middle subcarriers. For instance, for a sequence x={1,2,3,4}, for two times oversampling, (2-1)×4=4 zeros should be added around the center subcarrier. The new signal is equal to X={1,2,0,0,0,0,3,4}.

In mobile wireless communications, transmission channel generates different undesired changes in the information signal caused by reflections and diffractions. These changes might be attenuation, noise, interference and distortion (affected by the non-ideal response of the communication system). Channel can be modeled as a linear time invariant transfer function with Additive White Gaussian Noise (AWGN). It means received signal is y(t) = x(t) + n(t), because the noise n(t) is added to transmitted original signal x(t). The fundamental type of the noise source is the thermal noise which is random in nature and has zero mean Gaussian distribution. The noise is called white if the power spectrum density of the thermal noise is the same over a wide frequency band [14]. In other words, noise level is completely flat at every frequency. As it is mentioned before, received signal is composed of information bearing message and noise. Signal strength relative to the noise can be measured by using Signal-to-Noise Ratio (SNR) which is measured in decibels (dB). As can be seen from Equation 2.10, SNR is the ratio between the power of original transmitted signal and unwanted background noise.

$$SNR_{dB} = 10\log_{10}\left(\frac{\bar{x^2}}{\bar{n^2}}\right) = 10\log_{10}\left(\frac{P_{signal}}{P_{noise}}\right) \tag{2.10}$$

Figure 2.7 shows Power Spectral Density (PSD) of OFDM spectrum by using `PWELCH` function in `MATLAB` according to the IEEE 802.11a specifications after transmitting

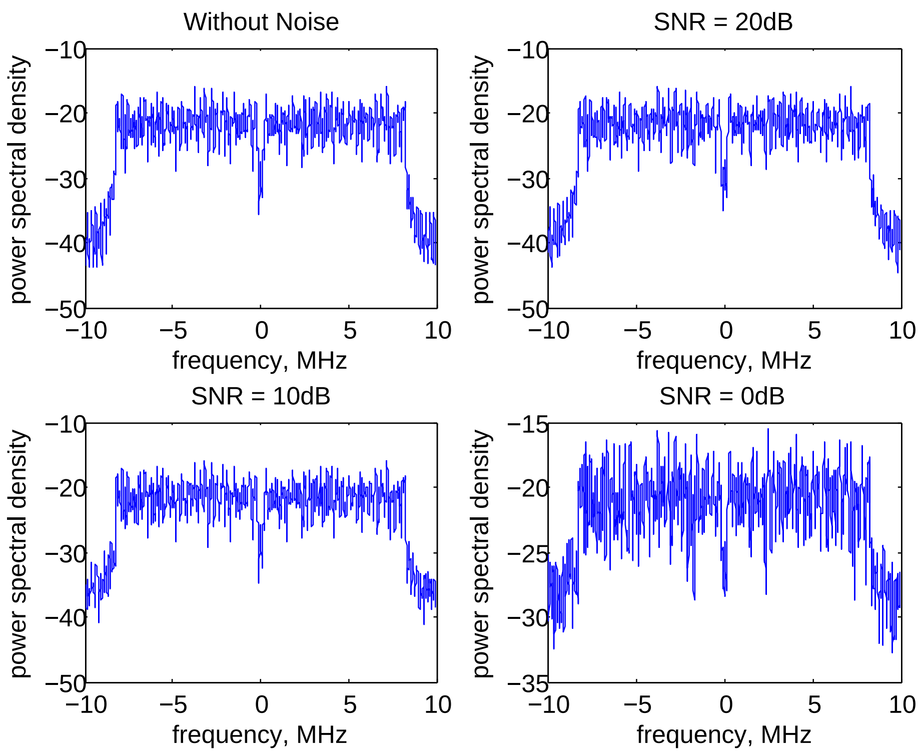


Figure 2.7 *Transmit spectrum of OFDM based on 802.11a specification*

16-QAM modulated signal across different four channels in terms of the amount of SNR. It is clear that the quality of signal is improved as SNR is raised and vice-versa. It should be noted that in the case where there is no noise on the channel, the PSD still looks noisy, since the data bits are generated randomly and the number of subcarriers are limited.

On the receiver side, the entire process which is required in the transmitter, must be accomplished in reverse order. The received signal is composed of training symbols (short and long), OFDM signal and data symbols. The first block is Analog to Digital Conversion (ADC) that converts the signal from an analog domain to the digital one for further processing. Furthermore, Automatic Gain Control (AGC) must be computed in order to control the gain of signals (more gain is applied on weaker signals and less gain on stronger signals) and to make sure the signal is not out of ADC dynamic range [15]. Once ADC is done, as it is shown in Figure 2.3, the next block performs packet detection and time synchronization. Packet detection is used in order to detect the beginning of the packet. This can be done by using correlation with short training symbols. Furthermore, time synchronization specifies the start point of received packets by correlating the inbound packet with known training symbols or delayed version of itself. Then the cyclic prefix or guard interval of data symbols is removed. It should be noticed that removing cyclic prefix must be done after time synchronization since before that there is not any information about the exact start point of the incoming packet. After removing cyclic prefix, frequency offset estimation is required in order to estimate the amount of frequency offset which is added to the transmitted signal in the channel and affected by clock deviation between the transmitter and the receiver. Once the frequency offset is estimated, received signal must be corrected. The corrected signal goes to FFT block for converting the time domain signal to the frequency domain. The subsequent block is channel estimation for estimating the channel impulse response by comparing received pilots and known transmitted ones. The comparing result must be used for the whole packet by interpolation. Finally, pilots are removed from the subcarriers and data carriers should be corrected in the channel correction block by dividing the data carriers by the estimated channel response. In the last stages, corrected data is demodulated based on chosen constellation (BPSK, QPSK, 16-QAM and 64-QAM) and the symbols are converted into a bitstream [8].

So far a general OFDM receiver system is described briefly, the next section provides full behavior of five main blocks of a receiver which are essential and vital for OFDM systems.

2.3 IEEE 802.11a Receiver

As earlier discussed, the receiver is the most important part of OFDM systems since transmitted symbols must be extracted with highest accuracy. The IEEE 802.11a receiver generally performs time and frequency synchronization, channel estimation, equalization and demodulation. Based on IEEE 802.11a standard specifications, from the received training symbols, the first seven of short training symbols can be used for packet detection, AGC and diversity selection. The remaining three short symbols might be used for coarse frequency offset estimation and timing synchronization. Moreover, the long training symbols should be used for channel and fine frequency offset estimation. In the following subsections, these operations are explained in detail.

2.3.1 Timing Estimation

Timing estimation in OFDM systems is divided into two main tasks: packet detection and symbol timing synchronization. Packet detection is necessary for OFDM systems since a receiver does not have any information about the start point of the received packet. In addition, time synchronization is essential in order to find the precise start point of the OFDM symbols which defines the correct position of the FFT window. As it is mentioned before, short training symbols could be used for detecting the packet since they are known to the receiver. It implies it is better to have a brief explanation of correlation before describing the algorithm of the mentioned block.

Correlation, is a method to determine the level of similarity between two signals. Here are two kinds of correlation: cross-correlation and autocorrelation.

Cross-correlation means correlation between two different signals while autocorrelation stands for correlation between a signal with its delayed or shifted version and is maximum when two signals are exactly matched with each other. Since correlation is computationally intensive and time-consuming, there are different fast algorithms for solving this problem. For instance, correlation of two signals might be computed by using their Euclidian distance $d(\hat{x}, \hat{y})$ in frequency domain [19] which is given by Equation 2.11.

$$\begin{aligned} d(\hat{x}, \hat{y}) &= \mid x - y \mid = \sqrt{\sum_{i=1}^{n} \mid x_i - y_i \mid^2} \\ corr(x, y) &= 1 - \frac{1}{2} d^2(\hat{x}, \hat{y}) \end{aligned} \tag{2.11}$$

Packet detection can be implemented by using delay and correlate algorithm where the received signal is correlated with its delayed version [8]. The output of this algorithm, c_n can be seen from Equation 2.12

$$c_n = \sum_{k=0}^{L-1} y_{n+k} y^*_{n+k+D} \tag{2.12}$$

Here y_n stands for received packet, D is equal to 16 which is the period of short training symbols in IEEE 802.11a and L is the length of correlation. Moreover, received signal power (p_n) might be applied in order to normalize c_n during the correlation period calculated in Equation 2.13 [17]

$$p_n = \frac{1}{2} \sum_{k=0}^{L-1} \mid y_{n+k} \mid^2 + \mid y_{n+k+D} \mid^2 \tag{2.13}$$

In order to find out that when two different correlation windows match completely (peak of correlation), decision metric (m_n) is computed from Equation 2.14

$$m_n = \frac{\mid c_n \mid^2}{(p_n)^2} \tag{2.14}$$

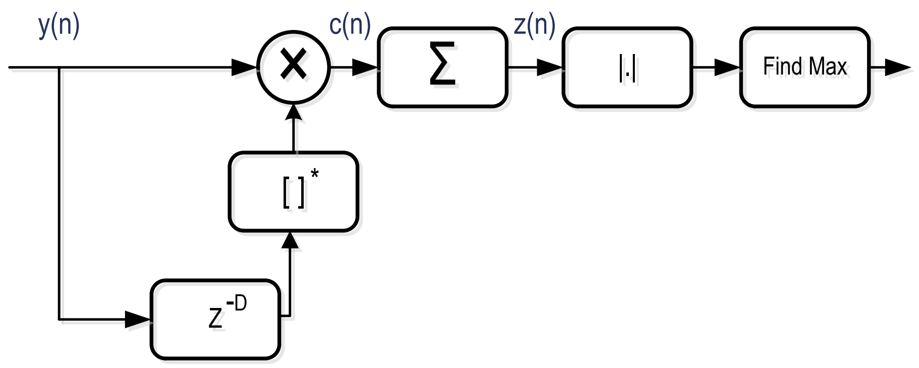


Figure 2.8 *Signal flow structure of the time synchronization by using delay and correlate algorithm [18]*

.

Thus detection is made once a correlation peak is observed.

Timing synchronization or **timing acquisition** could be implemented by using two different methods [20]:

- Using special symbols, e.g., training symbols, null symbols, PN-sequence.
- Cyclic Prefix (CP) or Guard Interval (GI) correlation method.

In the first method, a particular symbol is transmitted by the transmitter which is known to the receiver and the start point of the actual data carrying OFDM symbols can be found. In IEEE 802.11a, the end of short training symbols or the long training symbols of a received data packet might be used for timing synchronization which can be observed from Equation 2.15

$$z_n = \sum_{k=0}^{L-1} y_{n+k} t_k^* \tag{2.15}$$

where y_n is received signal, t_n is representing the known symbols and * stands for the complex conjugate operation.

Whenever there is not any information about the data content, the second method is used which is the most common way in OFDM systems. As it was discussed before, cyclic prefix or guard interval is used to combat against ISI. Thus in this method, received signal is correlated with its delayed version. The signal flow structure can be observed from Figure 2.8. The amount of delay z^{-D} is equal to the length of cyclic prefix which is 16 based on IEEE 802.11a standard specifications.

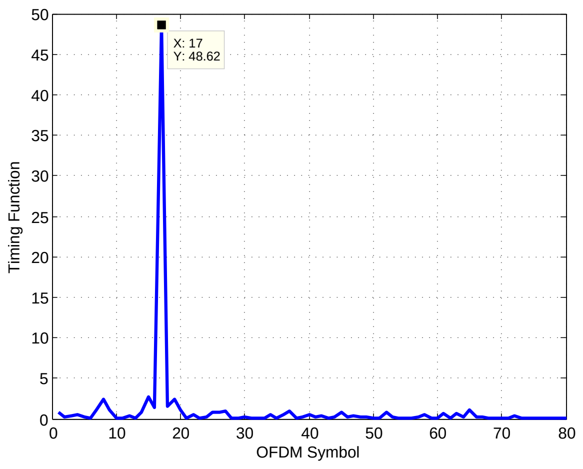


Figure 2.9 *GI correlation, SNR = 20 dB*

.

This method will product an output c_n and z_n which is given by Equation 2.16 and Equation 2.17, respectively.

$$c_n = c_n y^*_{n-D} \tag{2.16}$$

$$z_n = \sum_{i=0}^{L-1} c_{i+n} \tag{2.17}$$

Once a correlation is finished, pursuant to Equation 2.18, its largest peak must be recognized in order to compute the index of time offset which specifies the edge of the first FFT window.

$$\hat{\tau}_s = \underset{n}{\operatorname{argmax}} \mid z_n \mid \tag{2.18}$$

Figure 2.9 shows treatment of $\mid z_n \mid$ in a noisy channel without any multipath propagation. When the length of received data symbol in 802.11a is equal to 80 (64 FFT and 16 CP), if there is no multipath propagation, the data symbol correlated with itself has got one peak ($\hat{\tau}_s$) of which location minus one is exactly equal to the length of the cyclic prefix. Once the time offset is found, samples before the peak value have been skipped (corresponding to cyclic prefix removal) and the data without this offset is fed to the frequency offset estimation and correction module in the receiver for the subsequent processes.

2.3.2 Frequency Synchronization

As it is discussed earlier, OFDM waveform is composed of multiple sinusoidal components. In wireless communication systems, as it is shown in Figure 2.10, a signal must be upconverted (baseband to passband) to carrier frequency before transmis-

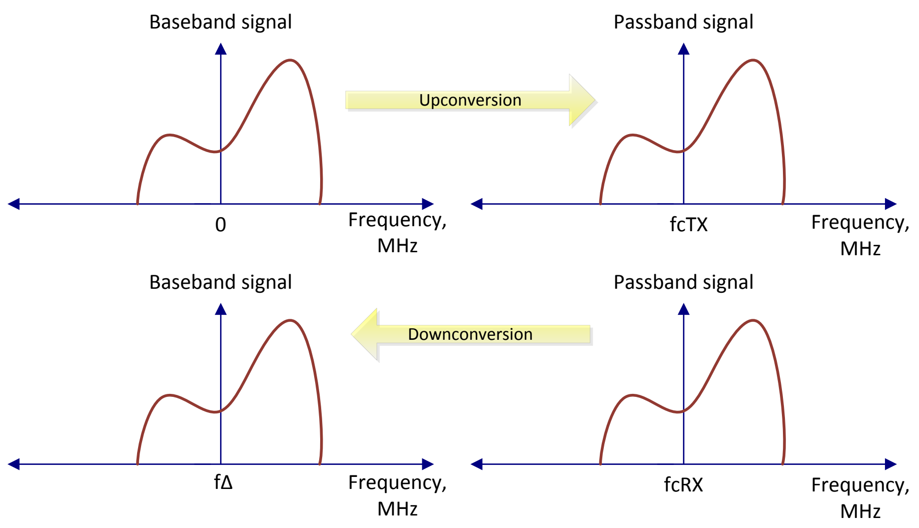


Figure 2.10 *Upconversion and Downconversion of signal in transmitter and receiver*
.

sion. On the receiver side, the signal is downconverted (passband to baseband) from the same carrier frequency prior to demodulation.

One of the disadvantages of OFDM is its sensitivity to carrier frequency offset which is due to device impairments [8]. It means that when carrier frequency of the receiver is not exactly the same to the transmitter one, the received baseband signal will be centered at a f_Δ instead of zero. Thus, f_Δ is equal to the difference between the carrier frequencies in transmitter and receiver side that can be observed from Equation 2.19.

$$f_\Delta = f_{Tx} - f_{Rx} \tag{2.19}$$

In other words, Carrier Frequency Offset (CFO) is created in OFDM systems due to inconformity of frequencies between the oscillators of the transceivers or due to the Doppler spread [16]. This issue causes a rotation of demodulated symbols in the constellation and may lead to intersymbol interference [17]. There are different methods to estimate the amount of CFO in OFDM systems [8]:

- Using special training symbols that are added in the transmitter
- Analyzing received signal in frequency domain (post FFT)
- Using cyclic prefix

Here, the first method will be explained in detail.

Frequency synchronization must be operated very accurately at the receiver in order to avoiding losing orthogonality between the samples. Frequency offset estimation in time domain could be performed by using Maximum Likelihood estimator. For this purpose, it is possible to use short training sequences while the duration of each of them is equal to $0.8\mu s$. Let us assume that x_n is our transmitted signal, then by considering Figure 2.10, passband signal y_n could be modeled from the complex baseband one as

$$y_n = x_n e^{j2\pi f_{Tx} n T_s}, \tag{2.20}$$

where f_{Tx} is carrier frequency of the transmitter. As described earlier, once the signal is received, it must be downconverted to baseband signal r_n with a carrier frequency f_{Rx} that can be seen from Equation 2.21. Also, f_Δ stands for frequency offset.

$$\begin{aligned} r_n &= s_n e^{j2\pi f_{Tx} n T_s} e^{-j2\pi f_{Rx} n T_s} \\ &= s_n e^{j2\pi (f_{Tx} - f_{Rx}) n T_s} \\ &= s_n e^{j2\pi f_\Delta n T_s} \end{aligned} \tag{2.21}$$

Frequency offset could be computed by using the same delay and correlate method which is illustrated in Equation 2.22. Based on IEEE 802.11a specifications, the amount of delay, D, calculated from the period of short training symbols ($0.8\mu s \times 20MHz(f_s)$) is equal to 16.

$$\begin{aligned} y_{\hat{\tau}} &= \sum_{n=0}^{L-1} r_n r_{n+D}^* \\ &= \sum_{n=0}^{L-1} s_n s_{n+D}^* e^{j2\pi f_\Delta n T_s} e^{-j2\pi f_\Delta (n+D) T_s} \\ &= e^{-j2\pi f_\Delta D T_s} \sum_{n=0}^{L-1} \mid sn \mid^2 \end{aligned} \tag{2.22}$$

Once above multiplication between the received signal and the complex conjugation of its delayed version is performed, the frequency offset estimate can be represented as

$$\hat{f_\Delta} = -\frac{1}{2\pi D T_s} \angle y_{\hat{\tau}}, \tag{2.23}$$

where T_s is giving the sampling period and $\angle$ takes the angle of $y_{\hat{\tau}}$ which is a correlation output from last equation. Frequency offset correction could be performed by utilizing the frequency offset estimated above and multiplying the received signal according to Equation 2.24 where $r_n{}'$ is the corrected signal, n is sample index and N is the number of samples in a symbol.

$$r_n{}' = r_n e^{-j2\pi f_\Delta \frac{n}{N}} \tag{2.24}$$

Furthermore, noting that phase calculation by using angle function is limited from $-[\pi, +\pi)$, thus the minimum and maximum value of frequency offset that could be estimated is ± 625 kHz [10].

2.3.3 Demodulation

One of the features of OFDM systems is a simple structure of modulator and demodulator to be performed by using IFFT and FFT, respectively. After calculation of the time offset and correction of the received signal in terms of frequency offset, transmitted data bits must be recovered. Based on IEEE 802.11a specifications, demodulation is operated by applying 64-point FFT within 3.2 μs. As it is mentioned before, FFT is a particular type of DFT since the number of multiply-accumulate operations is reduced considerably. On the other hand, it is still among the most computationally intensive blocks. The DFT of a signal x may be defined by [21]

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi \frac{nk}{N}}, \tag{2.25}$$

where the sequence of N complex numbers is transformed into an N-periodic sequence of complex numbers. Also, $e^{-j2\pi \frac{nk}{N}}$ is called twiddle factor (W_N^{nk}). Different kinds of FFT algorithms could be used in OFDM systems based on a particular communication standard or special concern during system design. One of the most common FFT algorithms is the radix-2 algorithm proposed in [22] and its complexity is equal to $O(NLogN)$ (while a DFT can compute the same in $O(N^2)$ operations). First of all, N-point (N is a power of 2) data sequence is divided into two $\frac{N}{2}$-point data sequences which is expressed as Equation 2.26.

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n W_N^{nk} \\ &= \sum_{n_{even}} x_n W_N^{nk} + \sum_{n_{odd}} x_n W_N^{nk} \\ &= \sum_{m=0}^{\frac{N}{2}-1} x_{2m} W_N^{2mk} + \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1} W_N^{(2m+1)k} \\ &= \sum_{m=0}^{\frac{N}{2}-1} x_{2m} W_N^{2mk} + W_N^k \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1} W_N^{2mk} \end{aligned} \tag{2.26}$$

According to the theory behind radix-2 algorithm, a 64-point FFT needs six stages ($2^6 = 64$) to be performed [23]. Furthermore, in order to reduce the number of stages,

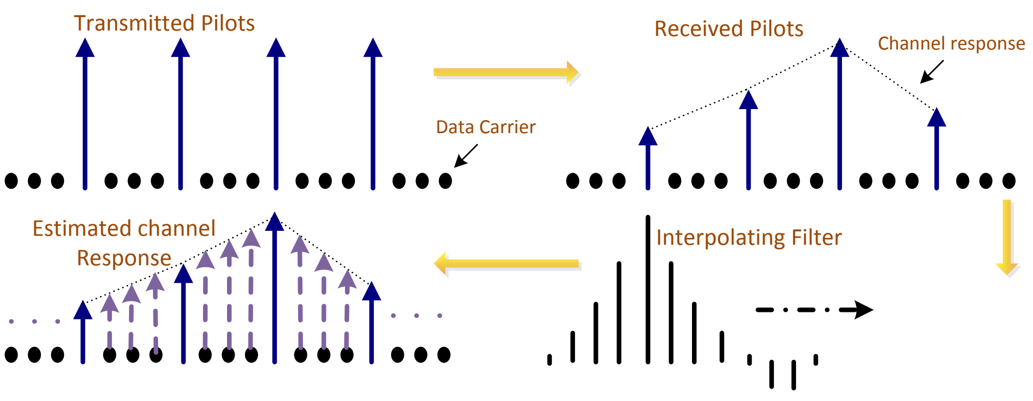


Figure 2.11 *Channel estimation based on linear interpolation*

.

increasing the processing speed, reducing the complexity and decreasing the number of operations, other radix-N algorithms have been developed based on radix-2, such as radix-4 and radix-8. Solving a 64-point FFT by using Radix-4 and Radix-8 algorithms requires three and two processing stages, respectively. Designing accelerators for FFT algorithms has always been challenging. There are different methods for doing this, one of which is using CREMA (discussed later on in detail) as a CGRA accelerator for RISC processors. The implementation of N-point FFT using the algorithms of radix-2 and radix-4 has been presented in [3] and [24]. The execution details of radix-N algorithms are very well described in the above-mentioned reference papers.

2.3.4 Channel Estimation

Transmitted symbols after passing through the wireless channel will get destroyed because of various impairments. Hence, these channel impairments require correction. Once data symbols are recovered after FFT, the frequency spectrum of the received signal must be estimated [8]. It is implemented by a channel estimation block which is located next to the FFT block. Received and demultiplexed OFDM symbols, Y_n, can be written as

$$Y_n = X_n H_n + N_n \tag{2.27}$$

Here n is representing the subcarrier number, H_n stands for channel response and N_n is the additive noise. Thus, in the case of a linear channel, there are two steps for performing channel correction [7]:

- Channel estimation attempts to estimate H_n

- Channel equalization attempts to correct Y_n based on X_n

There are different methods for performing channel estimation. Pilot-assisted linear interpolation and least square algorithm is one of them and is explained in the following. As it is discussed earlier, in WLAN OFDM some training symbols like pilots are transmitted which are known for the receiver and could be used for different purposes [25]. Based on IEEE 802.11a specifications, four specific values are inserted as pilots between data subcarriers in the transmitter. In the receiver side in order to accomplish the channel estimation, first of all, a diagonal matrix (is a square matrix in which the entries outside the main diagonal are all zero), M, is formed from transmitted pilots as

$$M = \begin{bmatrix} M_{1,1} & 0 & \cdots & 0 \\ 0 & M_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & M_{k,k} \end{bmatrix} \tag{2.28}$$

then the channel impulse response must be computed as follows

$$\tilde{H}_k = M^{-1} P_{Rx}, \tag{2.29}$$

where k is the number of pilots, P_{Rx} is representing the received pilots that might be noisy and $\tilde{H}_k$ is standing for channel impulse response of received pilots. Once channel response of the received pilots is found, since there is no information about the other subcarriers, the whole channel frequency response of the remaining subcarriers needs to be found. This can be done by linear interpolation as can be seen from Figure 2.11 [26]. Interpolation refers to up-sampling the signal. In other words, it means adding samples in between the existing values by using different techniques like linear, spline and so on. Linear interpolation is the method of approximating the value at each position between two samples. Thus, samples are joined by a straight line to each other. Equation 2.30 is extracted from `interp1` function in `MATLAB` [27] where channel frequency response of all subcarriers $\hat{H}_n$ is estimated by expanding channel frequency response of four received pilots $\tilde{H}_k$.

$$\hat{H}_n = \sum_{i=1}^{N_p-1} \sum_{j=1}^{N_s} \tilde{H}_k(i) + ((\tilde{H}_k(i+1) - \tilde{H}_k(i)) \times \underbrace{\frac{j-1}{N_s}}_{\mu}) \tag{2.30}$$

Here N_p is representing the number of pilots, N_s stands for the number of samples between each two pilots and μ is the step size and naturally is small value. Once

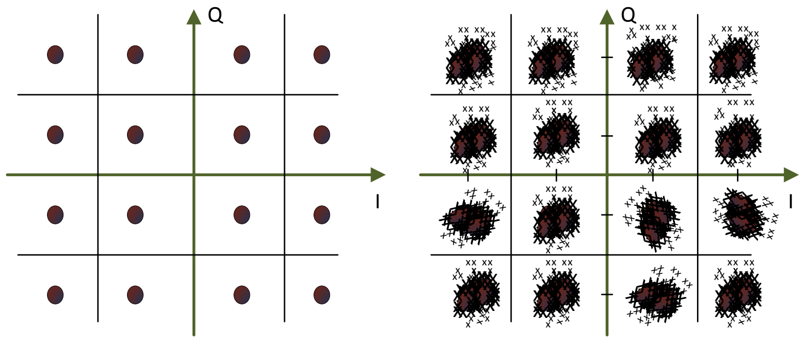


Figure 2.12 *Dividing complex plane into decision regions*

.

channel frequency response is estimated, channel equalization is required to rectify the received noisy OFDM data symbols Y_n as close as possible to X_n. It could be implemented by dividing the received signal by its channel frequency response expressed as Equation 2.31.

$$\hat{Y_n} = \frac{Y_n}{\hat{H}_n} \tag{2.31}$$

Subsequent to channel equalization, it can be observed that the amount of Bit Error Rate (BER) versus Signal-to-Noise Ratio (SNR) is reduced after implementing channel estimation in comparison with AWGN channel.

2.3.5 Symbols Demapping

Subsequent to carrying out all synchronization operations and demodulation, the next and last essential part of OFDM receiver is symbols demapping where the actual value of received data bits has to be decided. In other words, the main task of symbols demapping is converting the received data symbols to data bits without losing any precision. In the past, we have discussed the mandatory modulation formats (BPSK, QPSK, 16-QAM and 64-QAM) which are used in IEEE 802.11a standard. According to the utilized modulation type, decisions about received data bits must be performed. Based on the amount of information about each transmitted bit, decisions are divided into hard and soft [8].

Hard Decision

A hard decision demodulator might be used whenever the number of transmitted data bits is equal to the number of received ones. Also, consider that the received data bits could be noisy which will form a Gaussian cloud around the points in the constellation as can be observed in Figure 2.12. In such a case, the problem is to make a decision about transmitted data symbols, based on the received ones. Pursuant to the maximum-likelihood decision, if the received bit is closest to the constellation point, assigning bits is done by using hard decision. Thus, the complex plane could be divided into the set of points that are closest to a certain symbol.

Soft Decision

In this case, decision is made by using information bits instead of intermediate decisions about transmitted symbols along with giving better performance in terms of execution complexity in comparison with hard decision [28]. To perform soft decision, demodulator has to maximize the probability of similarity between the bit (x) transmitted and the bit (y) received as

$$P(x \mid y) = \frac{P(y \mid x)p(x)}{p(y)}, \tag{2.32}$$

where $P(y \mid x)$ is the conditional density of the received symbol when the transmitted one is known and $P(x)$ is the prior density of transmitted bit. Noting that since occurrence probability of all constellation points are equal, maximizing $P(y \mid x)$ is equivalent to $P(x \mid y)$ [29].

Once received data symbols are demapped to data bits, the quality of OFDM systems might be measured in terms of bit error rate (BER). BER is the percentage of bits with errors divided by the total number of bits that have been transmitted and is changing as a function of SNR. BER is declining by increasing the SNR. Furthermore, it should be noticed that in the same SNR environment, BER is dependent on modulation type. For instance, BER in BPSK is lower (higher performance) than in 64-QAM with the same SNR. The formula for bit error rate can be written as

$$BER = Q\left(\sqrt{\frac{2E_b}{No}}\right) = \frac{1}{2}erfc\left(\sqrt{\frac{E_b}{N_o}}\right), \tag{2.33}$$

where E_b is representing energy per bit, N_o is the noise power spectral density and totally, $\frac{E_b}{N_o}$ stands for SNR [30].

3. RECONFIGURABLE ARCHITECTURES

Currently, reconfigurable architectures are one of the most successful platforms containing different levels of reconfigurability and parallelism. Reconfigurability means modifying functionality at run-time for several applications. Most important features of reconfigurable computing systems can be listed as following [35]:

- Granularity: Data size for operations of the Reconfigurable Processing Unit (RPU) of a system.
- Depth of Programmability: The number of reconfiguration programs or contexts inside the RPU.
- Reconfigurability: In order to perform several applications, it is required by reconfigurable processing unit to be reconfigured at different times.
- Computation Model: It could be SIMD or MIMD or even in some cases, systems may follow the VLIW model.

The reconfigurable devices can be classified according to the level of granularity into Fine-Grain, Middle-Grain and Coarse-Grain. The last one is most popular between different reconfigurable architectures because of playing an important role for the digital base-band signal processing. In the following, we will describe some of the examples of CGRA briefly.

3.1 AVATAR and SCREMA

CREMA is a 4×8 PE CGRA template with two 32-bit local memories which will be described in detail in Chapter 4 as part of the implementation platform for this work. AVATAR is a scaled-up version of CREMA [47]. It consists of 64 PEs (4×16), therefore, it has more computational power than CREMA. In AVATAR, there are 32 inputs in the first row of PEs which means 32 of the 32×1 multiplexers are required in each I/O buffer. It can be observed that AVATAR energy consumption is almost the same as for CREMA while being 1.3X faster [31].

If we are going to scale-up AVATAR, next generation could be 4×32 PE CGRA which offers additional parallelism. On the other hand, it can be very resource demanding. As CREMA and AVATAR were CGRAs of fixed sizes, SCREMA was introduced as a CGRA platform with a scalable number of rows and columns (number of columns must be power of 2, such as 4, 8, 16 and 32). The basic structure of CREMA and SCREMA is the same, so, the functionality of PEs and routing between them is similar. Furthermore, user can make a decision about the number of PEs, thus, accelerators can be generated based on the user's design while it is possible to scale SCREMA at the compilation time. It shows the flexibility of SCREMA in order to have different sizes between CGRA templates [32].

3.2 ADRES

ADRES (Architecture for Dynamically Reconfigurable Embedded System) is Very Long Instruction Word (VLIW) processor tightly coupled to a CGRA [33]. Talking about the advantages of this integration, increasing the performance, declining communication cost and decreasing programming complexity could be mentioned. It is a platform executing at 40 MOPS/mW (Mega Operation Per Second) implemented in 90nm technology [34]. ADRES is a reconfigurable array of 8×8 elements where each of them is composed of Functional Units (FU) and Register Files (RF), which are connected in a certain topology (the simplest option is mesh). Moreover, since physical specifics of FUs and RFs are fundamentally the same, resources might be shared in order to have remarkable cost-saving. The FUs communicate through a multi-port global Data Register File (DRF) along with one destination and at most three source ports. In addition, the data bus width between FUs and DRFs is 32-bits. Furthermore, data access to the main memory could be done by using load and store operations which are accessible on FUs. The FUs support Single Instruction Multiple Data (SIMD) for high data level parallelism purposes. There are 1-bit Predicate Register Files (PRF) that store the predicate signal and other RFs can store intermediate data. The number of words in local and global RF is 16 and 64, respectively. It should be noticed that in ADRES just fixed-point operations can be executed. The ADRES instances can be generated using an XML-based architecture description language which is transformed into the VHDL files for further processes.

3.3 MorphoSys

MorphoSys is a system-on-chip which is composed of 8×8 array of Reconfigurable Cells (RCs), a general-purpose RISC processor and a high bandwidth memory interface to exploit data transfers between RCs and external memory [1]. The RC array is divided into four quadrants which are comprised of three hierarchical levels in terms of interconnection network [35]. These levels are nearest neighbor connectivity (2D mesh), intraquadrant connectivity (complete row and column) and interquadrant connectivity (express lane). The RC array follows SIMD model and consists of an ALU-multiplier for fixed-point operations and a register file. Moreover, its configuration memory can store up to 32-bit context word in the context memory for providing dynamic reconfiguration. The host processor of MorphoSys is a 32-bit processor, called TinyRISC which is a four-stage scalar pipeline and handles general-purpose and control operations by adding special instructions. Another important component of MorphoSys is Frame Buffer (FB) that is an internal data memory for enabling stream-lined data transfer between RC array and main memory. FB is physically organized into two sets, each of which is further subdivided into two banks (each bank has 64×8 bytes of storage) of memory. The context memory of MorphoSys stores the configuration program into two context blocks (each block has 8 context sets and each context set has 16 context words) and broadcasts them to the RC array. Since MorphoSys supports regularity and parallelism, all eight RCs share the same context word and perform the same operations in a row or column, respectively. Furthermore, DMA controller is used in MorphoSys in order to control all data movement between frame buffer, context memory and the external memory. Another important feature of MorphoSys is dynamic reconfigurability where the context memory can be reloaded in parallel with RC array accomplishment. Several applications could be simulated using *MorphoSim* which is the VHDL simulator. By utilizing 64, 128 and 256 elements RC array, MorphoSys could show a perfomance of 6.4, 12.8 and 25.6 GOPS (Giga Operations per Second) while performing Discrete Cosine Transform (DCT) and Inverse-DCT at 100.0 MHz.

3.4 PACT-XPP

The eXtreme Processing Platform (XPP) is a runtime-reconfigurable computing architecture composed of a 2D array of coarse-grain, adaptive PEs and interconnection resources [36],[37]. Various types of parallelism like pipelining, instruction level, data flow and task level are provided in the architecture of XPP which is suitable for stream-based applications. The most important feature of XPP is its sophisticated run-time reconfiguration and automatic packet-handling mechanisms. Run-time reconfigurability means that part of PEs might be reconfigured with a new functionality while others keep processing data without any interruption. There are several structures for XPP. The typical XPP is composed of four Processing Array Clusters (PACs) where each of them is attached to the Configuration Memory (CM) responsible for writing configuration data from external memory into the configurable resources of the array. There are two sets of buses for interconnection resources: data bus (identical word length for each device) and event bus (one-bit control information). There is a possibility for XPP to reduce configuration time by prefetching mechanisms where other configurations can be loaded to the CM cache (internal RAM) during loading main configuration onto the array. Another important feature of XPP is the possibility of performing an application containing several phases without any external control by asking self-reconfiguration of the device. However, the phases may contain similarities. For such cases, differential configurations are more effective where only configuration parts of PEs are changed. In order to exploit the performance of the XPP architecture, Native Mapping Language (NML) is developed which is a PACT dedicated language. Also, there is a C compiler (XPP-VC) for translating C functions to NML modules. The peak performance of PACT-XPP is estimated to be 57.6 GOPS when running at 150 MHz.

3.5 MORPHEUS

MORPHEUS ([2],[38]) is a complex SoC and dynamically reconfigurable that is a platform consisting of three main types: fine-grained, middle-grained and coarse-grained, which are Heterogeneous Reconfigurable Engines (HREs). FlexEOS is well suited for fine-grain algorithms. It is SRAM-based and embedded Field Programmable Gate Array (eFPGA) that can be programmed using VHDL. FlexEOS is constructed from high-density multi-function logic cells. DREAM is a middle-grain reconfigurable digital signal processor, composed by a 32-bit RISC core processor and PiCoGA-III reconfigurable datapath (matrix of reconfigurable logic cells). DREAM could be used for various applications (e.g. multimedia and telecom) where instruction level parallelism is required. XPP-III is a coarse-grain reconfigurable signal processor provides high parallel processing performance for streaming applications. In other words, XPP-III is a heterogeneous reconfigurable processor architecture consisting of a dataflow array and VLIW processor. Interconnection of MORPHEUS is divided into three independent parts: data transfer, configuration transfer, control and synchronization. All system modules (HREs, memory units and I/O peripherals) communicate with each other based on a 64-bit NoC. All data transfers between these devices might be Direct Network Access (DNA), Direct Memory Access (DMA) or manipulated directly by ARM (general purpose processor). Consequently, the most important features of MORPHEUS can be runtime reconfigurability, Ethernet-based network, remote updates, energy and time efficiency, real-time protocol decoding and dynamic changes of hardware configuration.

4. PLATFORM ARCHITECTURE

The hardware platform which is used during this research work is a template-based CGRA called CREMA. CREMA is working as an accelerator for a 32-bit general-purpose Reduced Instruction Set Computing (RISC) core. Both CREMA and COFFEE have been designed and implemented at the former Department of Computer Systems, Tampere University of Technology (TUT), Tampere, Finland.

During the first section, the architecture of COFFEE RISC core is explained briefly. In the second part of this chapter, CREMA is described in detail which is a template-based CGRA to generate run-time reconfigurable accelerators. Later, the implementation of SDR related algorithms is shown by using CREMA and its tools.

4.1 COFFEE RISC Core

In this section, different views to the COFFEE RISC core will be described in terms of software, hardware and pipeline structure which is an implementation technique in order to execute multiple instructions that are overlapped.

4.1.1 Introduction to the COFFEE RISC core

As we know, there are different kinds of processor architectures which differ in terms of the number of gates, power consumption, or in general complexity. COFFEE is an open source RISC processor core. By looking at COFFEE RISC core architecture, it can be found that there are some features which make it a typical RISC such as ability of doing one instruction per cycle which can be done by using pipelining, fixed instruction length in order to make decoding of instruction more simple and 32-bit data path width. RISC is also known as load and store machine since only load and store instructions access memory, whereas all data processing is done inside of the core datapath. In addition, addressing modes are simplified by using simple RISC instruction in order to reduce the critical path [5].

The main target of using COFFEE RISC core is setting up embedded systems as a general-purpose platform. In addition, more tasks can be accelerated by coprocessors

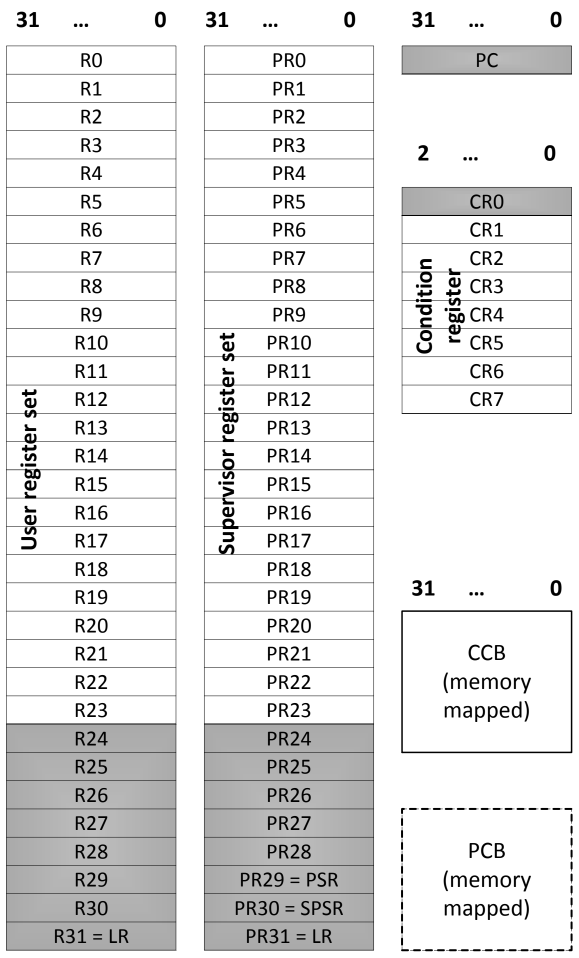


Figure 4.1 *Programmer's View of the Core's Registers [5]*

related to the telecommunications and multimedia applications [5]. The instruction set was designed as for a typical RISC processor and also to enable coprocessors [5]. Based on COFFEE RISC core user manual, it has 66 instructions in which fourteen are arithmetic instructions (add, addi, etc.), ten byte and bit field instructions (exb, exbf, etc.), six Boolean bitwise operations (and, andi, etc.), eight conditional jumps (beq, bnq, etc.), four other jumps (jmp, jmpr, etc.) and six shift instructions (sll, srl, etc.) and can be operated on two register operands, or one register operand and one immediate operand. In COFFEE RISC core, in order to implement conditional branches, it should compare register data and produce condition flags, based on which branch instructions can be executed [5].

4.1.2 Software and Hardware View of the COFFEE RISC core

As can be seen from Figure 4.1, there are two general-purpose register banks, user register set and supervisor register set which are introduced to support real-time operating system (RTOS) for COFFEE RISC core and each of them holds 32 registers [40].

Both register sets along with the full memory are obtainable in supervisor mode, whereas supervisor register set is not available in user mode. Moreover, there are two more blocks in order to enable software configurability and configure peripheral devices around COFFEE RISC core which are core configuration block (CCB) and peripheral control block (PCB), respectively. Talking about the advantages of memory mapped registers, it can be said that they are accessible by load and store instructions and also the core is configurable via boot code.

The COFFEE RISC core is a 32-bit Harvard architecture by which it is possible to distinguish interfaces for data and instruction memory. Furthermore, large and slow memories can be linked directly due to multi-cycle access whose access times are software configurable by using CCB. As can be seen on the interface of the COFFEE RISC core in Figure 4.2, there can be up to four coprocessors whose addressing include 2 bits for coprocessor identification and 5 bits for register indexing. Coprocessors are able to interrupt the core by asserting an exception signal. Moreover, there is a capability for coprocessor interface which can be connected to dissimilar clock domains [40].

As one can observe in Figure 4.2, instead of access to data memory for peripheral devices, it is possible to have direct access to PCB by asserting its write and read signals, pcb_wr and pcb_rd, respectively. Also, COFFEE RISC core provides an internal interrupt controller which can support twelve external interrupt sources where four of them are for coprocessors when connected. Moreover, interrupt sources can be classified by means of CCB registers between 0 to 15 based on priority. The interrupt is activated by an interrupt signal which is a high pulse on one of the interrupt lines and then the controller performs priority resolving, switching to an interrupt service routine and returning from an interrupt service routine, respectively [5].

Referring to the interface diagram of the COFFEE RISC core shown in Figure 4.2, boot_sel is high for reading boot address from data bus for first executed address by COFFEE RISC core. In addition, in order to save power in battery powered system, stall signal can be enabled when the system is in idle mode and there is nothing to happen. By releasing stall signal, software execution resumes immediately [5].

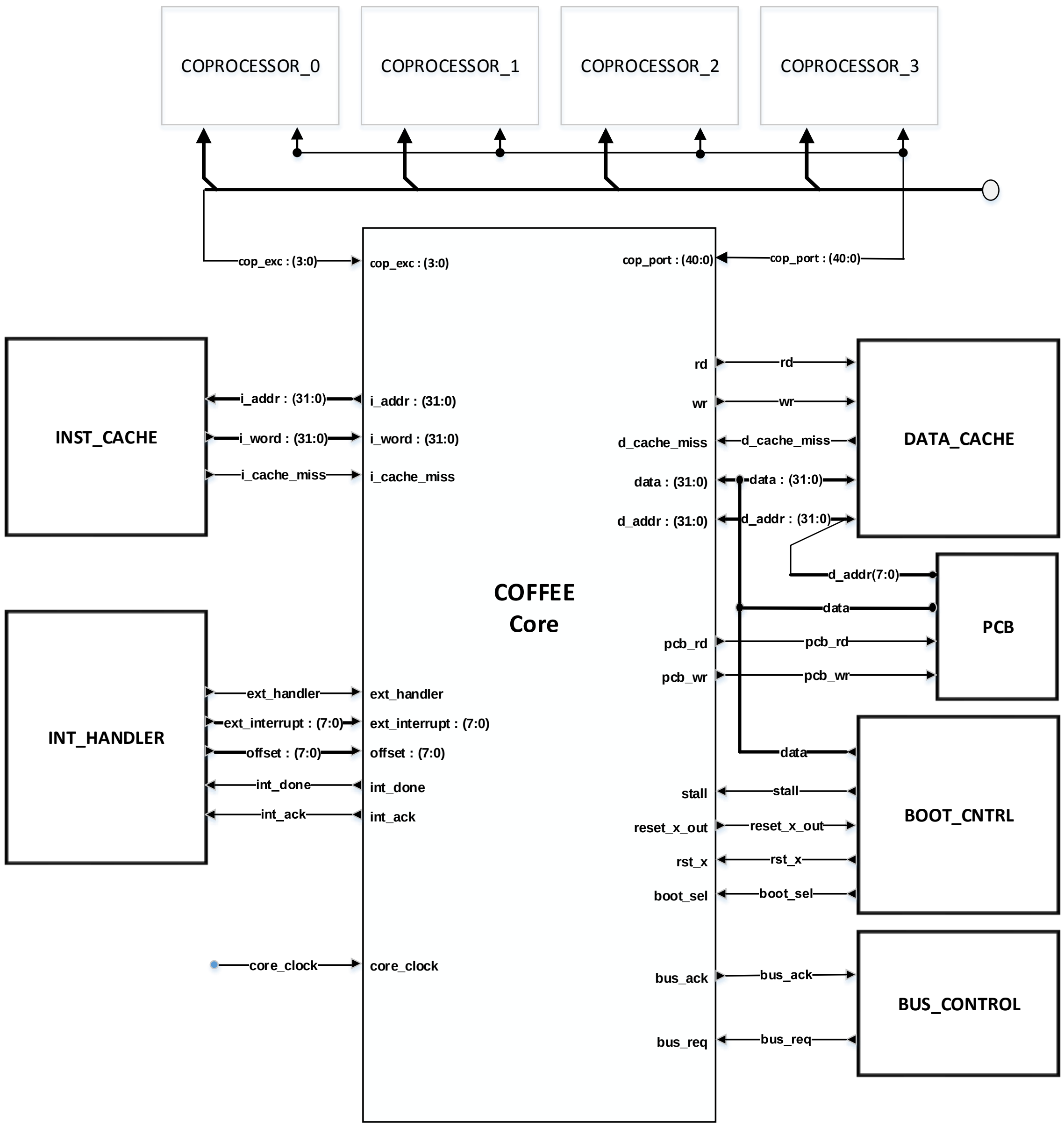
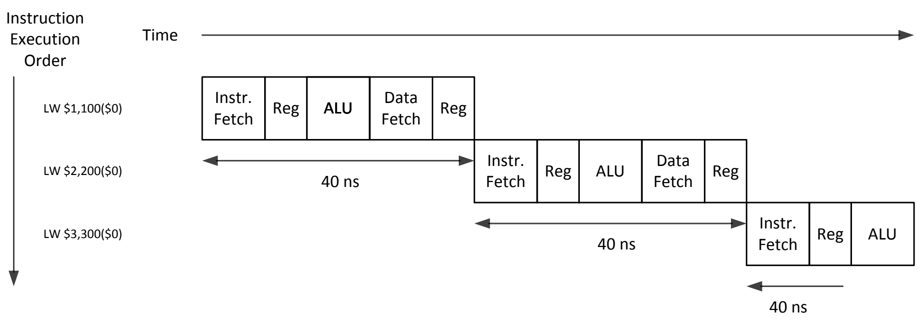


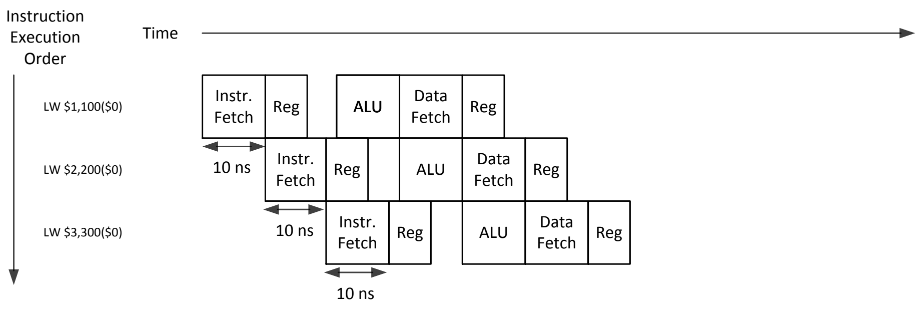
Figure 4.2 COFFEE RISC core interface ©IEEE, 2003 [5]

4.1.3 Core Pipeline Structure

The COFFEE RISC Core contains six stages pipeline, each of the stages can be done in a clock cycle, totally in six clock cycles for an instruction. At the end of each stage, intermediate or final results are clocked to the next stage from left to right. In an ideal case, we can have one instruction per cycle which means that new instruction enters the first stage and the other one completes the last stage without any stalls. Before talking about definition of each stage, let us have a short review about pipelining. As can be understood from its name, pipelining is a technique that multiple instructions can be implemented while overlapped in carrying out. So,



(a) Without Pipelining



(b) By using Pipelining technique

Figure 4.3 *Effect of Pipelining Technique on Execution of a Sequence of Load Instructions [42]*

there is no need to wait for executing each instruction before starting the next one [41].

Figure 4.3(a) shows the single-cycle design which is slowest one, so, it takes 3×40 ns or 120 ns for implementing these three instructions. On the other hand, Figure 4.3(b) depicts that by using pipelining it is possible to have a four-fold improvement over the single-cycle design.

The COFFEE RISC core pipeline stages are shown in Figure 4.4 and in the following, the task of each stage is summarized. In the first stage, Fetch, there are three operations. A new 32-bit instruction is read from the memory location specified by the program counter (PC) and placed in Fetch register. In some cases when the address is even in the 16-bit mode, a 32-bit double instruction is fetched. Furthermore, the address is checked in PC and an exception will be generated in case of a violation. After that, the PC address is increased by two or four and loaded back for next clock cycle.

During the Decode stage, which is the most important one based on control standpoint, it identifies an instruction and makes a decision about its performance for the next stages. In the case of 16-bit decoding mode, the instruction is extended to

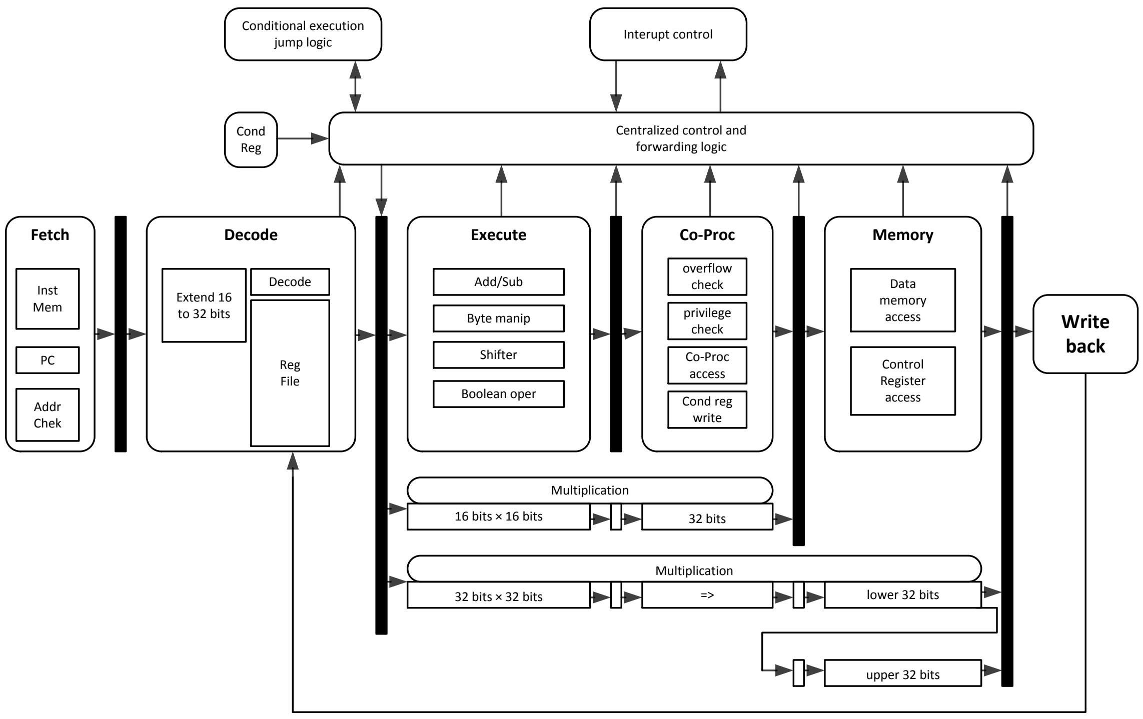


***Figure 4.4** Pipeline stages of COFFEE RISC core ©IEEE, 2003 [5]*

the 32-bit format. In order to make sure that execution condition is right, there is a particular field within the instruction word which specifies the execution condition. After evaluation of the condition, if there is any difference between pre-evaluated condition flags and evaluated execution condition, instruction gets flushed on the next rising edge of clock. Furthermore, there is a control section which checks data dependencies among signals assessed in decode stage and signals decoded from earlier instruction. All data dependencies could be resolved in COFFEE RISC core by forwarding the required data once it becomes available. In addition, fetch and decode stages are stalled in the case that source operand data cannot be forwarded (not available at the register file). All other operations like extending immediate operand and also all jump instructions and conditional branches are executed in Decode stage. Then register operands are clocked to the Execute register for further operations.

All kind of executions such as data manipulation, integer addition, shifting, Boolean and bit-field are performed inside the Execute stage. Multiplication instructions can be implemented at least in two or at most in three execution stages for 16-bit and 32-bit multiplications, respectively. During multiplication operations, intermediate results are produced for further processing in the second or third execution stages.

In the fourth stage, the condition flags which are assessed through earlier stages are written to the specified condition register and are available for decode stage prior to written to the condition register bank. It can be performed by forwarding data inside condition register bank from input to output while the source register and target register are the same. Moreover, in this stage, coprocessor accesses are achieved and also address calculation overflow is checked.

Memory stage is the last one for load (**ld**) and store (**st**) instructions, where data memory is accessed. It should be mentioned that because of the importance of fast data memory or data cache, pipeline will be stalled in the case of multi-cycle till the instructions are accomplished in order.

Finally, in the last stage, *Write Back*, execution gets finished and result is written to the selected destination register and data become observable to the decode stage due to internal forwarding of register file.

In the COFFEE RISC core implementation, it should be noticed that it is a Register Transfer Level (RTL) soft core described in VHDL (Very high speed integrated circuit Hardware Description Language) and is able to be portable between different technologies. The COFFEE RISC core is a general-purpose processing element which is suitable for System-on-Chip (SoC) environment and embedded systems. Furthermore, it is possible to add application-specific processing power such as CGRA accelerators to a COFFEE RISC core which is described in the next chapter in detail.

4.2 CREMA

CGRAs normally require an area of a few million gates as they contain a large amount of computational and communication resources. CREMA was introduced as 32-bit CGRA template in order to simplify the instantiation of application-specific CGRAs and computation intensive tasks to work as accelerators with COFFEE RISC core. The main concept behind CREMA is that designers can instantiate resources which are required for a particular application. Since the run-time reconfigurability is the main feature of CREMA, there is an ability to switch the functionality of the PEs at run-time. With compile-time configurability each PE supports just required operations and interconnections which is called mapping adaptiveness and makes CREMA suitable for many application [4]. In addition, the final resource utilization would be reduced considerably. It should be noticed that mapping adaptiveness is a design-time option. The designers can modify the functionality and routing of the existing PEs in one cycle by switching the context used. It is performed by using the reconfiguration memory inside the PE which is composed of

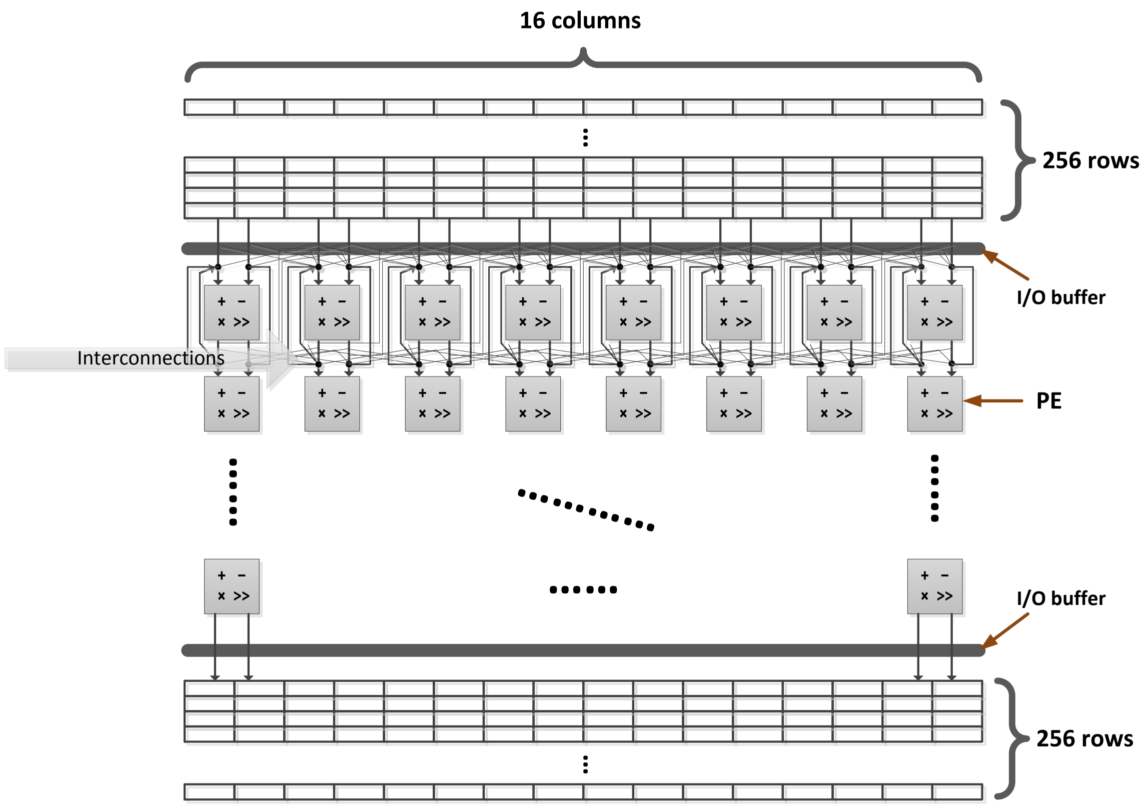


Figure 4.5 *CREMA Template ©2009 IEEE [43]*

reconfiguration words from different reconfiguration patterns. Each configuration word includes an operation field and address field for specifying the task of each PE and its address, respectively. For a new reconfiguration pattern, the configuration words are sequentially injected into the array of PEs and distributed along the horizontal and vertical direction in a pipelined way until they arrive to the correct PEs. The size of the reconfiguration memory depends to the size of the reconfiguration word and the number of contexts for fast switching [4]. Context means the pattern of interconnection among all PEs and the operations to be performed by the PEs which can be switched at run-time [31].

As it is shown in Figure 4.5, CREMA is equipped with 32 (4 rows×8 columns) PEs and (16 rows×256 columns) local memories for increasing the throughput of the CGRA. It can be seen that there are two I/O buffers made of registers and multiplexers for transferring data between local memories and PEs [44]. Each buffer has 16 I/Os and each of them is 32-bit wide. The first I/O buffer distributes the data from the first local memory to the PEs while the second I/O buffer redistributes the data from the PEs to the second local memory. The two local memories are working in "ping-pong" mechanism and they recieve the data to be processed from a Direct Memory Access (DMA) device [45]. In addition, the configuration words are loaded

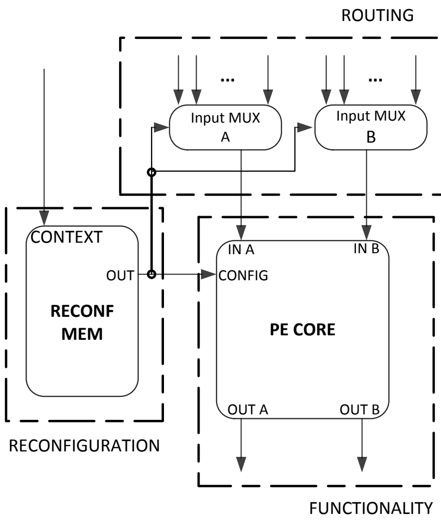


***Figure 4.6** Basic Components of each PE ©2009 IEEE [4]*

into the configuration memories of each PE with the help of same device and using a pipelined infrastructure [46]. The address of the destination PE is carried by a header in the configuration words.

Figure 4.6 depicts that the architecture of CREMA is composed of 3 main parts i.e., routing for operand selection by using two-input multiplexers, reconfiguration in order to define the functionality and routing of PE by using the configuration words and functionality in the PE core which are explained in detail in the following sections.

4.2.1 PE Core Parameters and Interconnections

All components of the PE core can be seen in Figure 4.7 which can be divided into functional blocks and configuration control blocks. Each PE receives two input operands and performs 32-bit integer and floating-point operations (IEEE-754 format) [31]. Furthermore, data can be exchanged between PEs through their ports and multiplexers are inside each PE [4]. A PE consists of a LUT, adder, multiplier, shifter, immediate register and floating-point logic. Unlike the two operand registers which are always active, each of these blocks (dashed borders) are instantiated only at design-time based on the processing requirements of the applications. Furthermore, there are two more blocks: decoder and output multiplexer. The size of these blocks has a direct relation with the number of operations that are going to be performed by each PE. For instance, if there is just one operation, there is no need to have decoder or output multiplexer, while in other cases, for n number of operations, the size of decoder and output multiplexer must be 1-to-n and n-to-1,

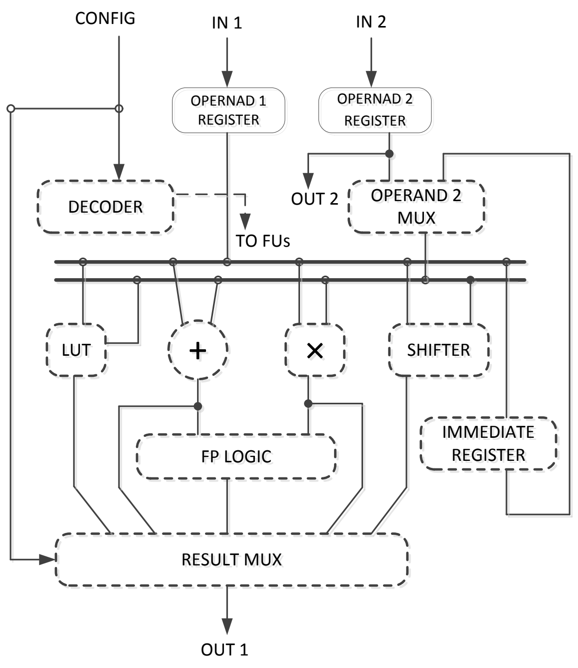


Figure 4.7 *PE core [4]*

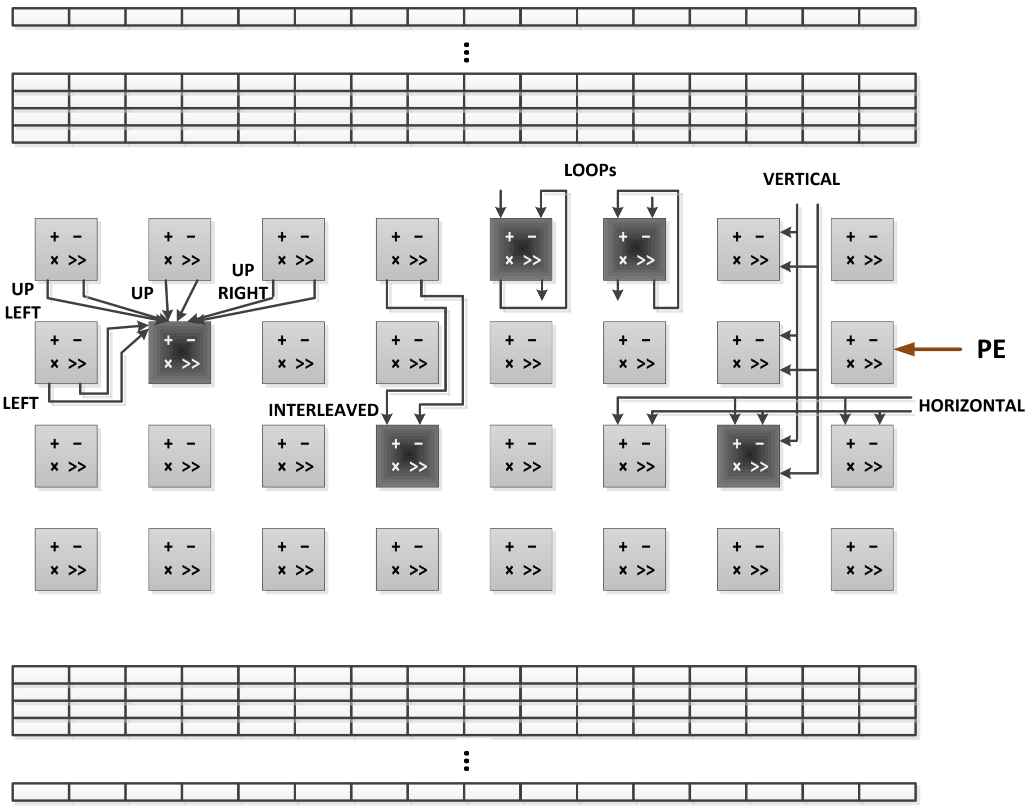


Figure 4.8 *Interconnections between PEs in CREMA [4]*

respectively. Moreover, there is one bit (config) belonging to functionality selection at run-time [4].

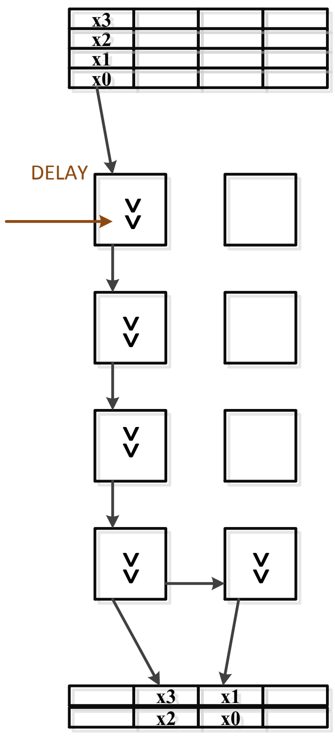


Figure 4.9 *Data reordering by using delay chains*

The Figure 4.8 shows all the possible interconnections between PEs in order to process data which are coming from the first local memory to the second local memory and selected by each multiplexer at run-time. By using this kind of interconnections, it is possible to process data in optimal way. For example, in order to calculate the sum of values in an array, designer can use LOOPs while the length of iteration should be equal to the length of the array and then, transfer the final result to the second local memory. Every PE can accept and process two input operands (32-bit wide). Totally there are 15 different routing possibilities in three categories i.e., local, interleaved and global. Local connections are only with the nearest-neighbor PEs while the global connections are divided into vertical and horizontal connections for distributing data between PEs. Furthermore, vertical connections are used for loading the immediate values to the PEs which are required for shift operation. This placement and routing for an application is performed using an in-house designed Graphical User Interface (GUI) tool.

4.2.2 CREMA Control Unit

CREMA control unit is a user defined technique to exploit different number of latencies (varied from one to thirty-two) which can be implemented using delay chains. CREMA control unit is based on two Finite State Machines (FSMs) which are read state machine and the write state machine. Both of them are working in parallel to read the data from one local memory and write in the other in a time-multiplexed way. The write state machine supports both write cycles and write stalls which mean writing for a given number of cycles and then disable the writing for a defined number of stalls, respectively [4].

For instance, as it is shown in Figure 4.9, the data stored in a single bank needs to be processed in parallel. Therefore, an additional step is required for reordering the data. It can be performed by using a 5-element delay chain and write latency for 5 cycles until the first input ($x0$) attains the last delay element.

4.2.3 Process of Application-Specific Accelerator Design on CREMA Platform

CREMA can be used for developing accelerators to COFFEE RISC core for Software Defined Radio (SDR) purposes. Application-specific CGRAs are generated from CREMA template based on user-specific input. The computational kernels which are required to be accelerated using CREMA must be identified by application developer. All application-specific accelerators should be designed based on the algebraic expressions with suitable placement and routing. A given application can be accelerated by using several configuration pattern. A configuration pattern specifies the operations to be performed by each PE and the interconnection between PEs. It should be noticed that CREMA generated CGRA accelerator is programmable in C. The design, program and execution flow of CREMA is almost same for all applications. Mapping of the kernels on CREMA can be performed manually by using a graphical interface and text description, respectively.

In order to design application-specific accelerators, the application designer has to manually defines the functionalities of the PEs and routing between them by using a dedicated tool, called Firetool. Firetool is a GUI which has been written in JAVA (language) and is used for reconfiguration management and field-programming of CREMA [4]. In other words, a set of configuration patterns for each application can be specified with Firetool. Each pattern is composed of information related to the functionality of each PE and the routing between PEs. The GUI of the Firetool for the context description is composed of 32 PEs in form of a drop down menu. Each PE has two source operands where they are composed of all possible interconnections between PEs. Another important part of designing specific accelerators by using CREMA, is defining I/O buffers which make connections between local memories and PEs. Once all these steps are performed, the tool creates a set of configuration package that includes VHDL package and a set of C header files for the run-time reconfiguration.

The configuration stream is loaded in the configuration memories of CREMA accelerator by the user-specified program flow entirely written in C. The program flow contains custom function calls which are supported for COFFEE RISC core. After configuring the array with a set of contexts, the data must be loaded into the local memories of CREMA for processing. Then a context must be activated in order to process data which are in local memory. These steps may be repeated for processing the previous results or loading new data into the local memories and switching the contexts until the algorithm completes its execution [3].

5. DESIGN IMPLEMENTATIONS AND ALGORITHMS MAPPING

In last two chapters, the basic structure of OFDM systems and platform architecture including analyzing the general characteristics of run-time reconfigurable devices is explained. Furthermore, it is described in detail how an application-specific accelerator could be designed by utilizing CREMA. In this chapter, design and execution of four blocks of OFDM receiver using CREMA are studied in detail and fully described. As it was explained earlier, a baseband receiver performs digital signal processing algorithms in order to exploit the received data bits with highest accuracy. First of all, the whole transceiver algorithms have been implemented using `MATLAB` to test the functionality of baseband algorithms and also, the random data symbols using a chosen constellation based on IEEE 802.11a specifications is generated. Afterward, each block is written in `C` in order to map the applications platforms. Then accelerators are designed and implemented for each block and their output is compared with the result of `MATLAB`.

In this experimental work, OFDM data symbols are generated based on 16-QAM and the channel is modeled with Additive White Gaussian Noise (AWGN) where SNR is equal to 20 dB. Moreover, it is assumed that the amount of CFO is equal to 40 kHz. In the following, it can be seen that how an application specific accelerator for each receiver blocks can be designed and implemented in CGRA. Furthermore, ModelSim is used for simulation purposes and testing the functionality of each accelerator along with computing the number of clock cycles in terms of COFFEE and execution time for each. Furthermore, the designed accelerators will be synthesized on Stratix FPGAs (Field Programmable Gate Arrays).

5.1 Time Synchronization

The first block of OFDM receiver is time synchronization which is used in order to find the right position of FFT window. Basically, synchronization in different wireless standards requires the calculation of the correlation for determining similarity between received signal and its delayed version. As it was mentioned before, there

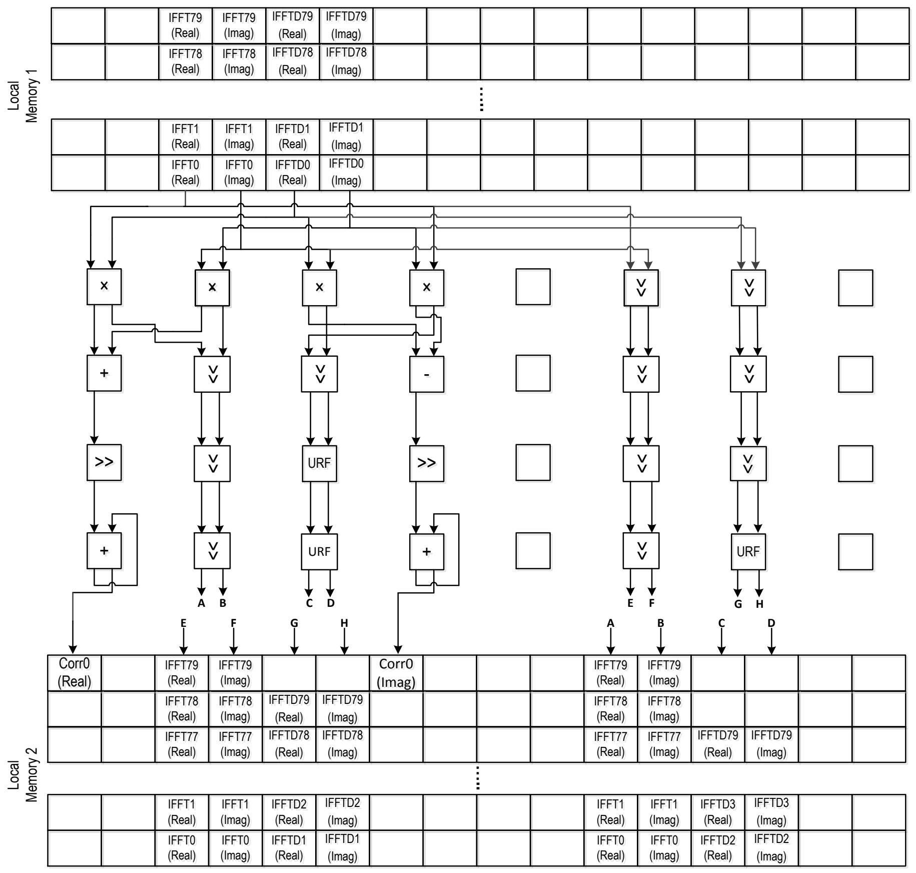


Figure 5.1 *First context for the calculation of the correlations*

are two ways to perform time synchronization: using special symbols or cyclic prefix correlation method. In this experimental work, the second method is used. Thus, a correlation is implemented between the received signal and a delayed version of itself where the amount of delay z^{-D} is equal to the length of cyclic prefix that is $D = 16$ according to the IEEE 802.11a specifications. The mapping of an 80-point correlation algorithm on CREMA is depicted in Figure 5.1 and Figure 5.2. It should be noticed that these two contexts must be used consecutively. Moreover, they are completely same in terms of the interconnections among PEs and the operations to be performed by PEs. However, the only difference between them is their I/O buffers. First of all, the received data symbols (the output of IFFT) must be loaded into the first local memory. Based on Equation 2.16, the received data symbol should be multiplied by complex conjugation of its delayed version. The multiplication of a complex number and its complex conjugate could be written as

$$(x_i + iy_i)(x_{i+D} + iy_{i+D})^* = \underbrace{(x_i x_{i+D} + y_i y_{i+D})}_{RealPart} + i\underbrace{(x_i y_{i+D} - y_i x_{i+D})}_{ImaginaryPart}, \tag{5.1}$$

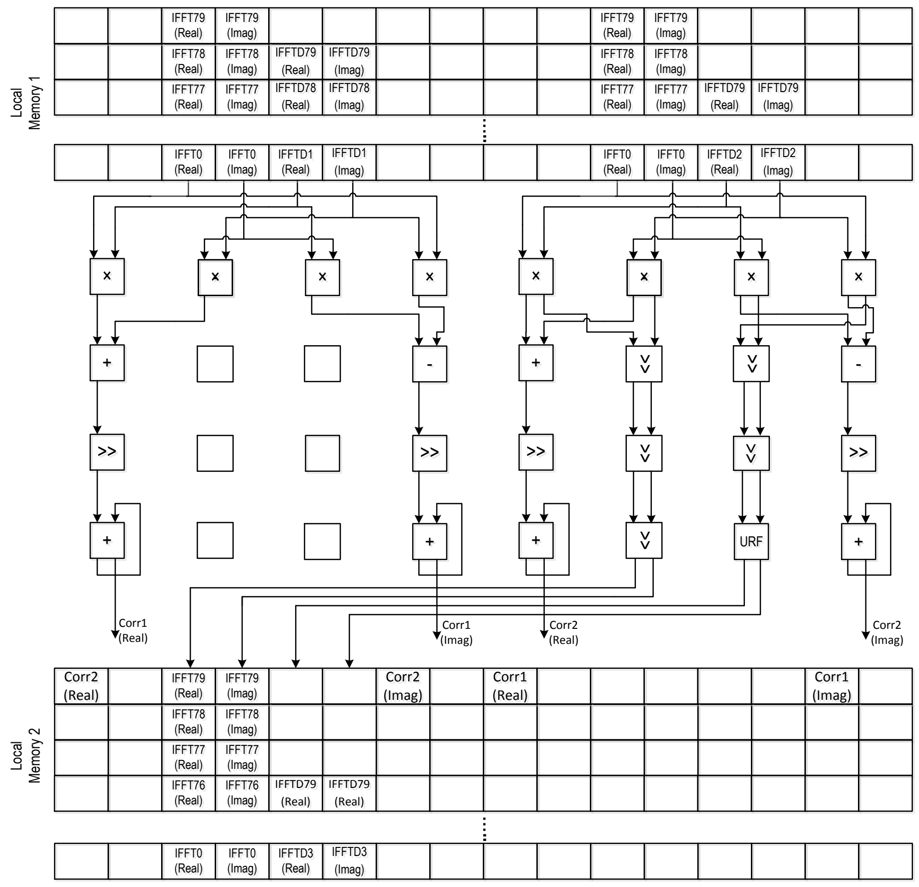


Figure 5.2 *Second context for the calculation of the correlations*

where x_i, y_i, x_{i+D} and y_{i+D} are equivalent to $IFFT_i$ (real), $IFFT_i$ (imag), $IFFTD_i$ (real) and $IFFTD_i$ (imag), respectively. In order to perform time synchronization for 80 data samples in one data symbol, 80 correlations are required. As it is mentioned earlier, a shift operation is needed after each multiplication (12 bits in this case). For each correlation, a sum-of-product of the results of complex multiplications must be calculated which could be done by utilizing LOOPs, noting that the length of an iteration should be equal to the length of the array. Otherwise, the delayed version of received data symbol is shifted by one and the correlation is repeated. This issue can be solved by using Unregistered-Feed Through (URF) operation. There is a possibility for us to perform two correlations in parallel in just one context. As can be observed in Figure 5.1, URF is used in a context in order to shift the delayed version of received data symbol by one- and two-cycle delay which allows accomplishment of the next step. Subsequent to each iteration, the final result of a sum-of-product is transferred to the main memory for further processing. In addition, a shifted version of data is transmitted to the second local memory to execute further correlations utilizing a same context. Hence, there is no need to transfer back all data samples between two subsequent correlations and the inputs could be recycled. Once corre-

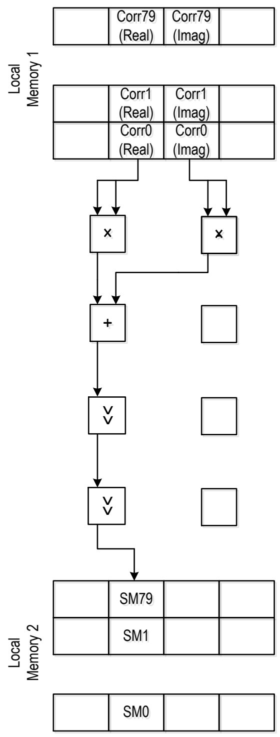


Figure 5.3 *Two columns of the context for the square modulus*

lation is done, another kernel used in the time synchronization should be mapped on CREMA which is shown in Figure 5.3 and $Corr_i$ and SM_i are representing the complex result of each correlation and the result of square modulus, respectively.

The square modulus of the complex results is required in order to acquire the maximum value (time offset) after the correlation. Thus, the complex result of each iteration is transferred back from main memory to the first local memory and the sum of the square values of the real and imaginary part could be written in the second local memory for comparison purposes. It should be mentioned that since the calculation of the square modulus and the pure modulus is equally beneficial for comparison, we just mapped square modulus on CREMA for this purpose. For instance, if $z = a + bi$ is a complex number, square modulus and pure modulus of z can be defined as $(a^2 + b^2)$ and $(\sqrt{a^2 + b^2})$, respectively. For each data symbol, only one context of square modulus is enough which can be implemented in two columns. It means that we can operate square modulus for four data symbols by using all eight columns of one context at once. In addition, it should be considered that since CREMA is efficient only for a large amount of data, in some cases is better to perform square modulus by using the processor instead mapping on CREMA. With the completion of the implementation of this code, the index of the largest peak is specified which is equivalent to the start point of FFT window.

```c
int  maximum, x, location = 1;
maximum = Result[80];

for (x = 1; x < 80; x++)
{
  if (Result[x] > maximum)
  {
       maximum  = Result[x];
       location = x+1;
       }
}
```

Program 5.1 *C code for the search of maximum value inside an array*

Once the calculation of square modulus is finished, results are transmitted back to the main memory in order to detect the largest peak and compute the index of time offset which is the edge of the first FFT window. The search of the largest peak for each data symbol is executed by the C code in Program 5.1 which is performed in COFFEE RISC processor software. Subsequent to finding the time offset, the data symbol is transferred to the next block which is Frequency offset estimation that is discussed in the next section.

Simulation Results

Out of the simulations that are implemented for time synchronization, mapping of correlations and square modulus use altogether Three contexts and four I/O buffers. The first context is used for loading the immediate values into the PEs which is required for shift operation. The rest three contexts are utilized for correlations and square modulus, respectively. The 80-point correlation is performed in only 50 CC (clock cycles), totally, in 4017 CC for the whole 80 correlations. It should be noticed that there is an overhead between each correlation due to the control operations which is composed of context switching, reloading of I/O buffers, loading another configuration and transferring data from main memory to local memory or vice versa and also, it is implemented in COFFEE RISC core. The whole performance of correlations can be performed in COFFEE processor software with 85681 CC by using nested loops or 12800 CC by using a loop for each correlation separately. It should be mentioned that in an ideal case, the time offset could be observed after only 879 CC. Execution of square modulus will take 225 CC in CREMA, versus 1680 CC in COFFEE RISC core. Then, at the last part of time synchronization, maximum value along with its location could be found in COFFEE RISC core software (Program 4.1) and requires 835 CC. Finally, it can be observed that the whole process of time synchronization can be implemented in 5077 CC by using both CREMA and processor software.

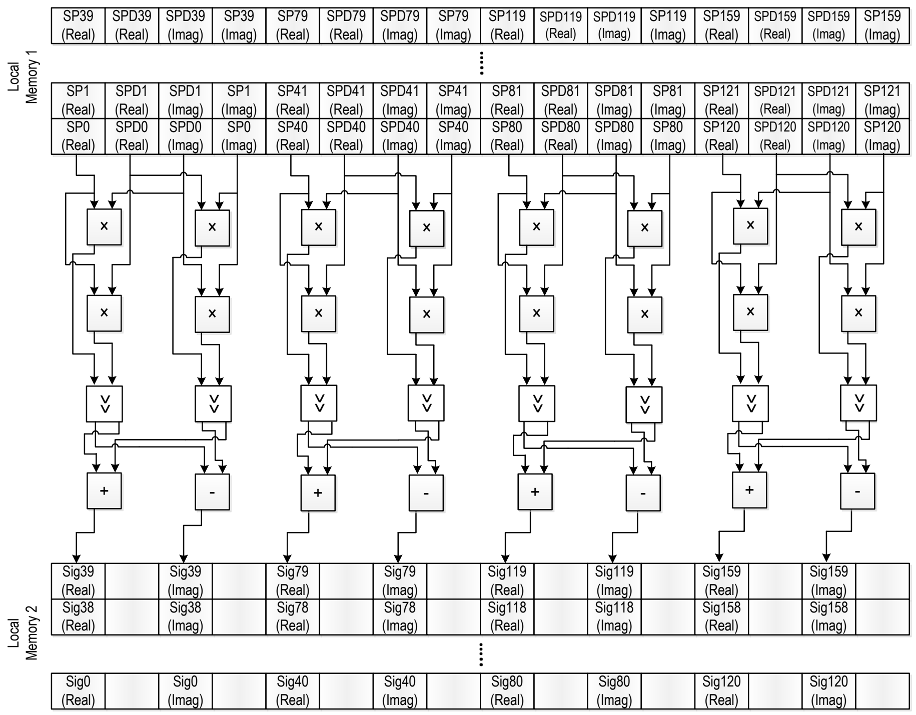


Figure 5.4 *Context for the multiplication between a signal and its complex conjugation*

5.2 Frequency Offset Estimation

As it was mentioned before, carrier frequency offset occurs in OFDM systems because of mismatch between the oscillators of the transmitter and receiver that could be estimated and corrected by using several methods. Based on IEEE 802.11a specifications, short training symbols might be used in order to estimate the amount of carrier frequency offset. For this purpose, according to Equation 2.22, delay and correlation method is utilized where the amount of delay is equal to 16. In other words, received training symbols must be multiplied by complex conjugation of its delayed version in order to acquire the phase difference between them. Thus, there is need to 160 complex multiplications (equal to the length of short training sequences) that can be performed via CREMA in just one context.

The first context is depicted in Figure 5.4 where short training symbols are loaded into the first local memory along with its delayed version by utilizing DMA from the main memory of the system. Here *SP* and *SPD* stand for short preamble and its delayed version, respectively. It is considered that this context is the most optimal implementation of multiplication between complex numbers as all sixteen PEs are used. Subsequent to multiplication between received signal and its delayed version, the amount of phase difference must be computed in the next step. Hence, the result of first context is transferred back from the second local memory to the main memory for further processing.

There are several methods in literature for computing the phase angle of a complex value. One of these methods is using `ATAN` function, expressed in radians. Let us assume that we are going to find the phase angle of a complex number like $x + iy$ where x and y are real and imaginary parts, respectively. The required equation could be expressed as

$$z = atan(\frac{y}{x}), \tag{5.2}$$

where z is representing the phase angle of a complex number. Thus, at first, an imaginary part must be divided by the real part and then, phase angle to be computed by utilizing `ATAN` function. It is feasible to emulate division in software using CORDIC algorithm ([48], [49], [50]).

CORDIC Algorithms
COordinate Rotation DIgital Computer (CORDIC) is an efficient algorithm for the calculation of trigonometric and hyperbolic functions (functions of an angle), including exponential and logarithmic. In addition, it might be used for other purposes containing complex number multiplication, division, matrix inversion, eigenvalue computation, conversion between binary and mixed-radix systems and general scientific computation. CORDIC algorithms could be vital when there is no predefined hardware multiplier since it is using only addition, subtraction, bit-shift and lookup table. The reason behind the popularity of CORDIC algorithms is due to simplicity of its hardware implementation which could be performed using the basic shift-add operation of the form $a \pm b.2^{-i}$.

In this case, we just used CORDIC algorithm in order to execute division operation in COFFEE RISC processor software. Let us assume that the imaginary part (y) should be divided by the real part (x). The division (z) could be found using the following C code which is composed of shifted version of x.

```
for (i = 0; i < MaxBits; i++)
{
        if (y < 0 || z >= 0)
        {
                y = y + x*t;
                z = z - t;
        }
        else
        {
                y = y - x*t;
                z = z + t;
        }
        t = t >> 1;
}
```

Program 5.2 *C code for CORDIC division algorithm*

Here the initial value of z is equal to zero. Moreover, since all numbers are represented in 12-bits format, the initial value of t must be assumed $1 * 2^{12} = 4096$. Also, by increasing the number of iterations (*MaxBits*), the results increase in accuracy accordingly. The CORDIC division algorithm is based on rewriting the equation $z = \frac{y}{x}$ into the form $y - x * z = 0$. The value of z is computed by driving t to zero 1 bit at a time (right shift).

Once the division is performed, based on Equation 5.2, the phase angle of a complex number must be calculated by using the result of division from the previous part. As it was discussed earlier, there are no predefined functions in COFFEE RISC processor, thus we should implement `ATAN` function by utilizing another method such as Taylor series [51]. Taylor series is an expansion of a particular function into an infinite sum of terms about a point. A Taylor series of a real or complex-valued function *f(x)* could be written as Equation 5.3.

$$\begin{aligned} f(x) &= f(a) + f^{'}(a)(x-a) + \frac{f^{''}(a)}{2!}(x-a)^2 + ... \\ &= \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \end{aligned} \tag{5.3}$$

Moreover, Taylor series could be expanded particularly for different functions like

`ATAN` which is expressed as Equation 5.4.

$$\begin{aligned} arctan\ x &= x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + ...\ for - 1 < x < 1 \\ &= \sum_{n=0}^{\infty} \frac{-1^n}{2n+1} x^{2n+1} \end{aligned} \tag{5.4}$$

Thus, the phase angle of a complex number can be computed using above equation. In addition, since all numbers are represented in 12-bits format, the precision is not lost and we are confident on the accuracy of the Taylor series.

Once the phase angles of the received data symbols are found, carrier frequency offset must be estimated using Equation 2.23 which was explained earlier. Then, data symbols should be corrected separately based on estimated frequency offset using Equation 2.24 where the exponential function is required. Accordingly, Taylor series expansion of exponential function is needed that could be written as

$$\begin{aligned} e^x &= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + ...\ for\ \ - inf < x < inf \\ &= \sum_{n=0}^{\infty} \frac{x^n}{n!} \end{aligned} \tag{5.5}$$

Considering to the above equation, it is clear that it is just working for integer numbers, not the complex ones. Hence, Equation 5.5 can be rewritten for a complex number z as

$$e^z = e^x(cos(y) + isin(y)), \tag{5.6}$$

where z is composed of real part (x) and imaginary part (y). Meanwhile, Taylor series is still required in order to expand `COS` and `SIN` functions as a sum series which are depicted in Equation 5.7 and Equation 5.8, respectively.

$$\begin{aligned} cos\ y &= 1 - \frac{y^2}{2!} + \frac{y^4}{4!} - \frac{y^6}{6!} + ...\ for\ \ - inf < y < inf \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} y^{2n} \end{aligned} \tag{5.7}$$

$$\begin{aligned} sin\ y &= y - \frac{y^3}{3!} + \frac{y^5}{5!} - \frac{y^7}{7!} + ...\ for\ \ - inf < y < inf \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} y^{2n+1} \end{aligned} \tag{5.8}$$

Finally, the received signal must be multiplied by the correction factor that is calculated above. This multiplication can be mapped in another context which is shown

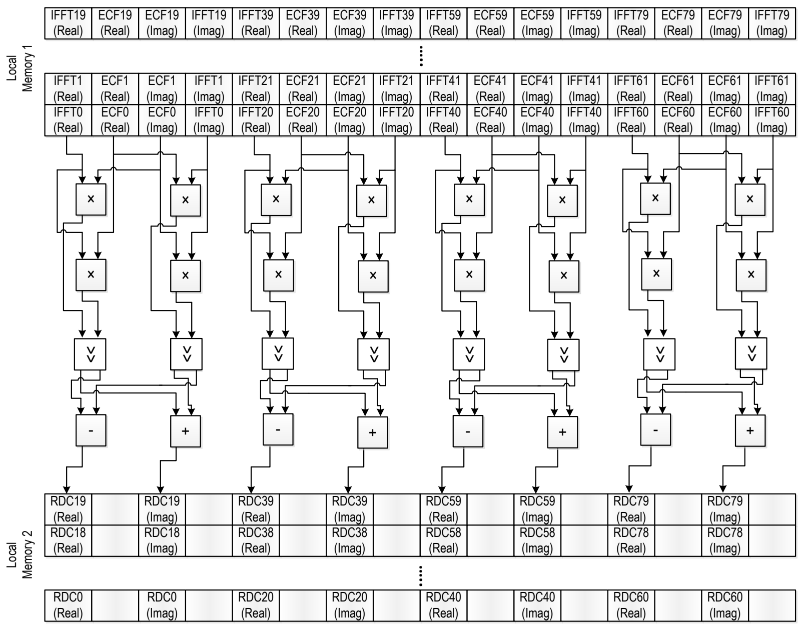


Figure 5.5 *Context for the complex multiplication*

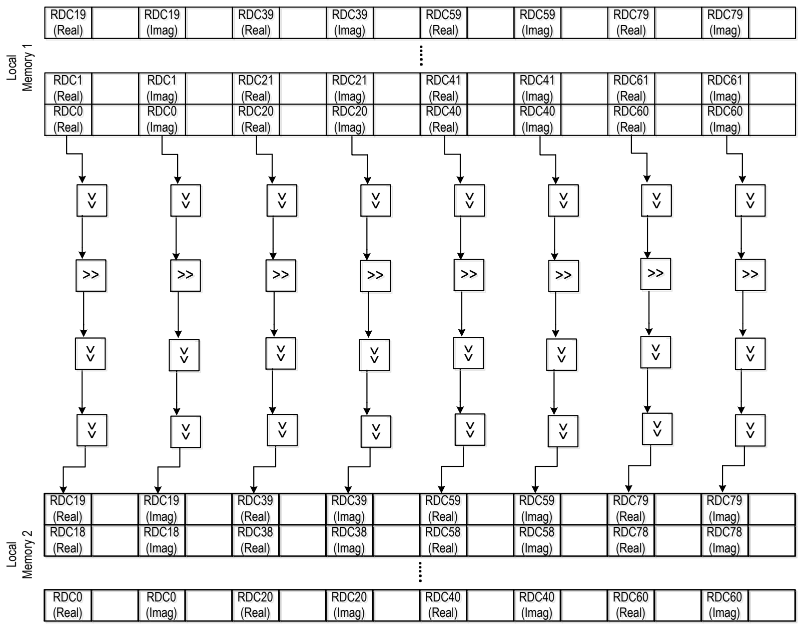


Figure 5.6 *Third context for the frequency offset estimation*

in Figure 5.5. It should be noticed that the previous context of this block (complex conjugate multiplication) is not suitable for complex multiplication due to the sign of the imaginary part.

As can be seen in Figure 5.5, the output of IFFT is loaded into the first local memory using DMA along with estimated correction factor (ECF) in order to be corrected prior to demodulator block (FFT). Once complex multiplication is performed, results

(RDC standing for Received Data Corrected) are stored in the second local memory. As it was mentioned before, the shift amount is required after each multiplication where in this case the shift amount is equal to 12 (since all numbers are represented in 12 bits). Thus, in the last part, the final result could be shifted in a separate context which is depicted in Figure 5.6.

After executing the operation of frequency offset estimation, the local memories already have the data symbols which can be demodulated directly. It should be noticed that cyclic prefix must be removed before demodulating the data symbols. Based on IEEE 802.11a demodulation, 64-point FFT operation is executed to retrieve the transmitted data symbols in the frequency domain.

Simulation Results

The simulation of the whole frequency offset estimation block is executed using both CREMA and processor software. Totally, mapping on CREMA consume altogether four contexts and three I/O buffers. The processing of 160-point multiplication between a complex number and its complex conjugate is performed in only 26 CC using CREMA, against 4338 CC for the COFFEE RISC core. However, due to the communication overhead, the number of clock cycles is increased to 485 CC. The second part of frequency offset estimation, division using CORDIC algorithm, requires 163 CC for each division. Thus, we need 29357 CC in order to execute 144-point division (the first 16 points are ignored due to their zero values) which is done in software of COFFEE RISC core. Then, the correction factor could be implemented by using Taylor series in 5868 CC in software. It should be mentioned that the amount of n is equal to 8 and 9 based on Equation 5.7 and Equation 5.8 respectively in order to keep precision. The last part of this block is 80-point complex multiplication along with shift operation that require 30 CC (842 CC with communication overhead) in CREMA, while, it will take 2338 CC in processor software.

5.3 Channel Estimation

Once data symbols are recovered in demodulator block, the channel frequency response must be estimated in the next step. As it was discussed earlier, the transmitted symbols may be distorted in the wireless channel due to different impairments. Channel estimation can be performed using the long training sequences or the pilots which are already known to the receiver. At the beginning part of this block, channel impulse response must be calculated based on Equation 2.29 which is the complex multiplication between the received pilots and inverse of the transmitted ones. Based on IEEE 802.11a specifications, the number of pilots is equal to four for

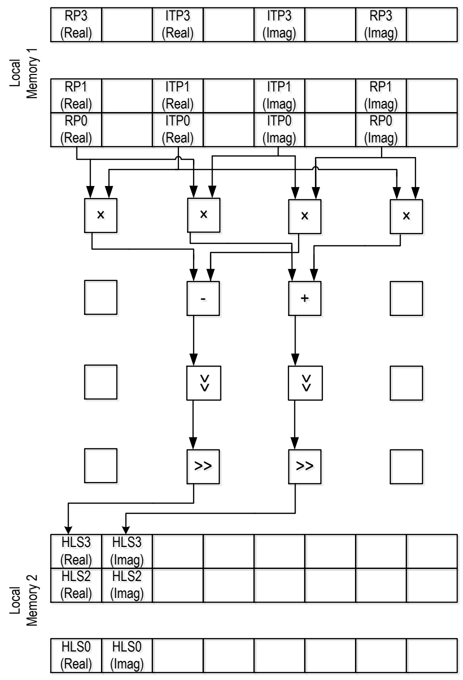


Figure 5.7 *First context for the channel estimation*

each data symbol that is inserted between subcarriers in the transmitter.

As can be observed from Figure 5.7, only four columns of PEs are utilized for the first context of channel estimation while six PEs are unused. Here, RP and ITP are representing the Received Pilots and Inverse of Transmitted Pilots which are loaded into the first local memory for further processing. This context is the same as the context of complex multiplication for the frequency offset estimation, but the difference is that the shift operation is performed in the same context, not in the separate one. Thus, the channel response (HLS) could be computed and stored in the second local memory using only one context.

As it was discussed previously, subsequent to computation of the channel impulse response of pilots, it should be expanded for the rest of the subcarriers by using linear interpolation algorithm (Equation 2.30) that refers to adding samples between each two pilots. Linear interpolation could be mapped on CREMA using only one context. However, it must be executed for real and imaginary parts of the channel impulse response separately.

First of all, as it is shown in Figure 5.8, the real part is loaded into the first local memory along with the step size which can take sixteen different fixed values specifically for this case (64-point OFDM). The last two columns of PEs which are not active in the first context are used in order to transfer the result of the real part of

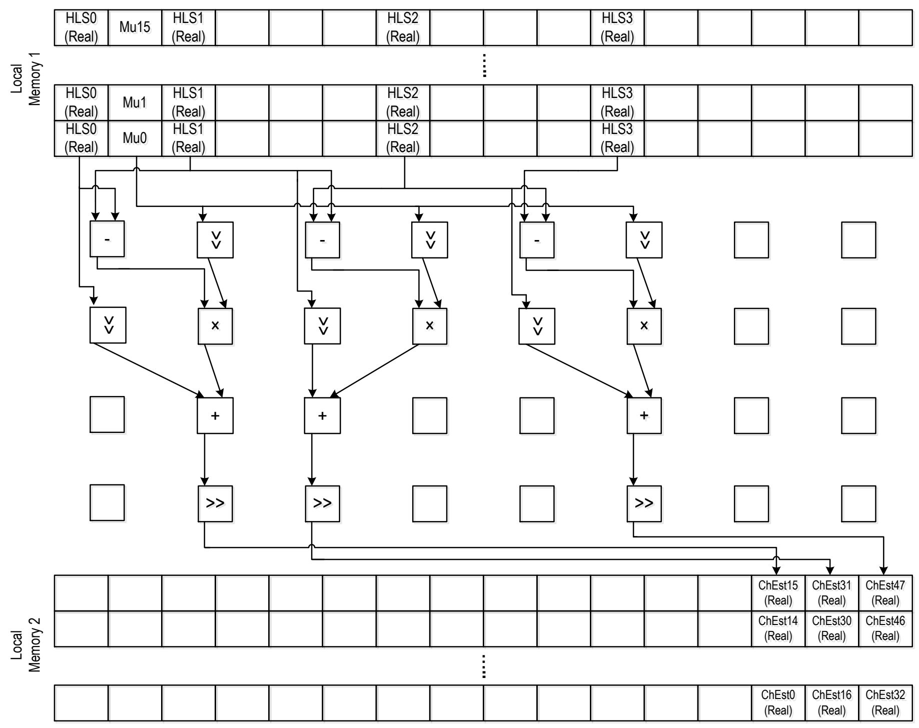


Figure 5.8 *First context for the Linear Interpolation*

linear interpolation to the second context through delay operation. Once linear interpolation is completed in Figure 5.9 for both real and imaginary part, the channel frequency response for all of the subcarriers is transmitted back to the main memory since it will be required in the last step for implementing channel equalization. After computing the channel estimates, the demodulated data symbols must be equalized with respect to Equation 2.31. Channel equalization can be performed by dividing the received data symbols and their channel response one after another. As it was mentioned earlier, division operation is not available in COFFEE RISC core and CREMA. Accordingly, we should use another algorithm in order to perform division operation which is essential for channel equalization. During the previous section, frequency offset estimation, we described and implemented CORDIC algorithm for division. CORDIC algorithm could be one of the best and most efficient methods if there is no knowledge about the values of the numerator and denominator. However, we made a decision to use another algorithm for two reasons:

- Designing CGRA for CORDIC algorithm is not efficient due to using iteration in that.
- Denominator could be assumed a fixed value in this case that is explained in the following.

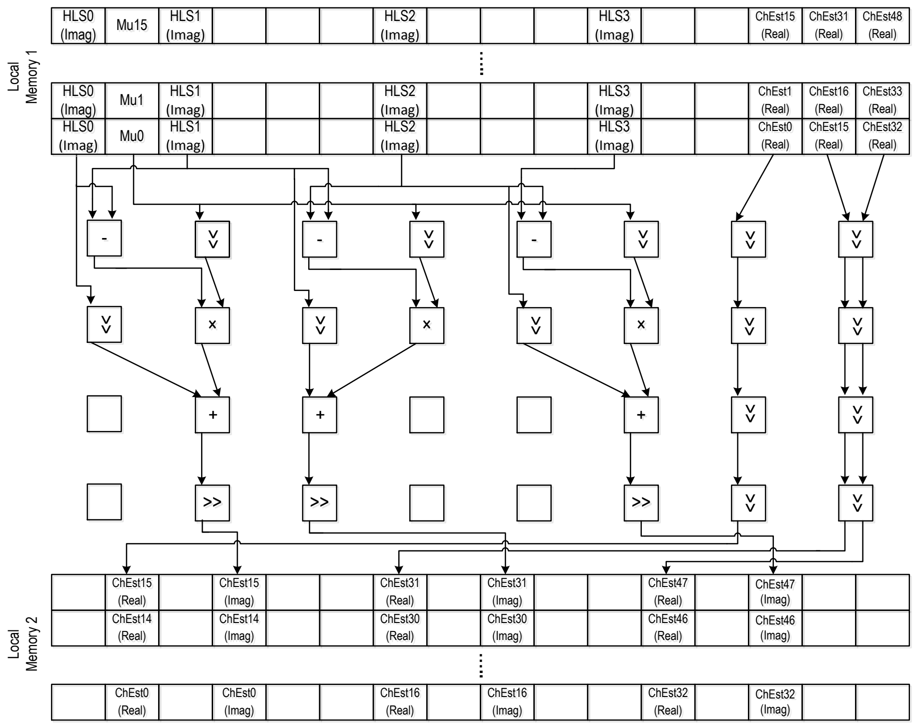


Figure 5.9 *Second context for the Linear Interpolation*

Newton-Raphson method [52] is an algorithm for finding the root of an equation. If there is a given function *f(x)*, the algorithm can be applied to obtain the first approximation of its root which is expressed as

$$x_{n+1} = x_n + \frac{f(x_n)}{f'(x_n)}, \tag{5.9}$$

where n = *0, 1, 2, 3, ...* is the number of iteration, x_0 is the initial guess for the root of the function f and f' is derivative of a function. Also, there is a possibility for Newton-Raphson method to be simplified for the division operation purpose. For example, let us assume that we are going to find $\frac{1}{D}$. For this purpose, a function *f(x)* should be found which has a zero at $x = \frac{1}{D}$. Thus, a function could be written as $f(x) = \frac{1}{x} - D$ and expressed specifically for division based on the above equation as

$$\begin{aligned}
x_{n+1} &= x_n + \frac{\frac{1}{x_n} - D}{-\frac{1}{x_n^2}} \\
&= x_n + \left(\frac{\frac{1}{x_n} - D}{-\frac{1}{x_n^2}} \times \frac{-x_n^2}{-x_n^2}\right) \\
&= x_n - (-x_n + Dx_n^2) \\
&= 2x_n - Dx_n^2 \\
&= x_n.(2 - Dx_n)
\end{aligned} \tag{5.10}$$

Here D is representing the denominator. In this case, the denominator is a complex number due to the noisy channel. Hence, it would be simplified to an integer for further processing using Newton-Raphson method and mapping on CGRA according to the Equation 5.11 where $x + iy$, $a + ib$ and $a - ib$ stand for demodulated data symbols after FFT block, estimated channel response and complex conjugation of the channel response, respectively.

$$\frac{x + iy}{a + ib} \times \frac{a - ib}{a - ib} = \frac{(x + iy) \times (a - ib)}{a^2 + b^2} \tag{5.11}$$

First of all, in order to perform channel equalization, we should map Newton-Raphson method on CREMA for computing the value of $\frac{1}{a^2+b^2}$ where D is equivalent to $(a^2 + b^2)$ based on Equation 5.10. Here a and b are representing the real and imaginary part of the channel frequency response which are already computed and stored in the local memory. Mapping of Newton-Raphson method follows the same methods demonstrated before for CREMA. The mapping is depicted in Figure 5.10 and Figure 5.11 which means the whole algorithm could be done in two different contexts.

The mapping occupies seven and six columns of the first and second context, respectively. During the first context, the first row of PEs performs multiplication in order to calculate the square values of the real and imaginary parts of the channel frequency response which are added to each other on the second row. Then, obtained results must be multiplied by the initial guess (Dx_n). In the next stage, referring to Equation 5.10, *2* is loaded into the local memory along with the results of the previous context. It is to be noticed that since local memories are only line readable in CGRAs (to become simpler and faster), the values of X and *2* are loading along with every column for correct performance. From the second context, it is apparent that the first row of PEs executes preprocessing of data using latency which is described earlier. Finally, within the last three rows of PEs, the required shift operations, subtractions and multiplications are implemented and the ultimate

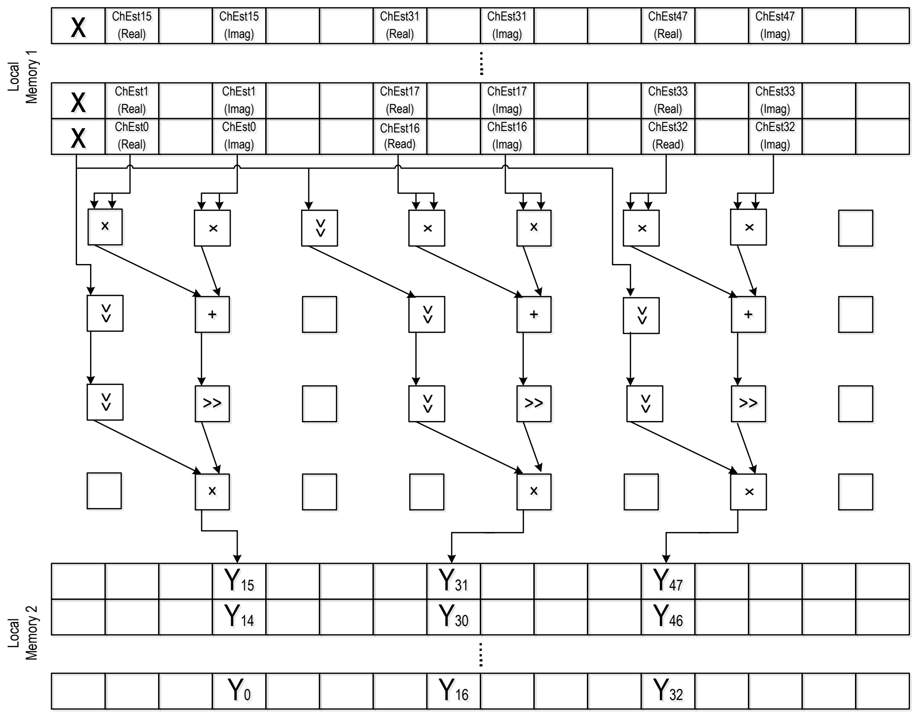


Figure 5.10 *First context for the Newton-Raphson method*

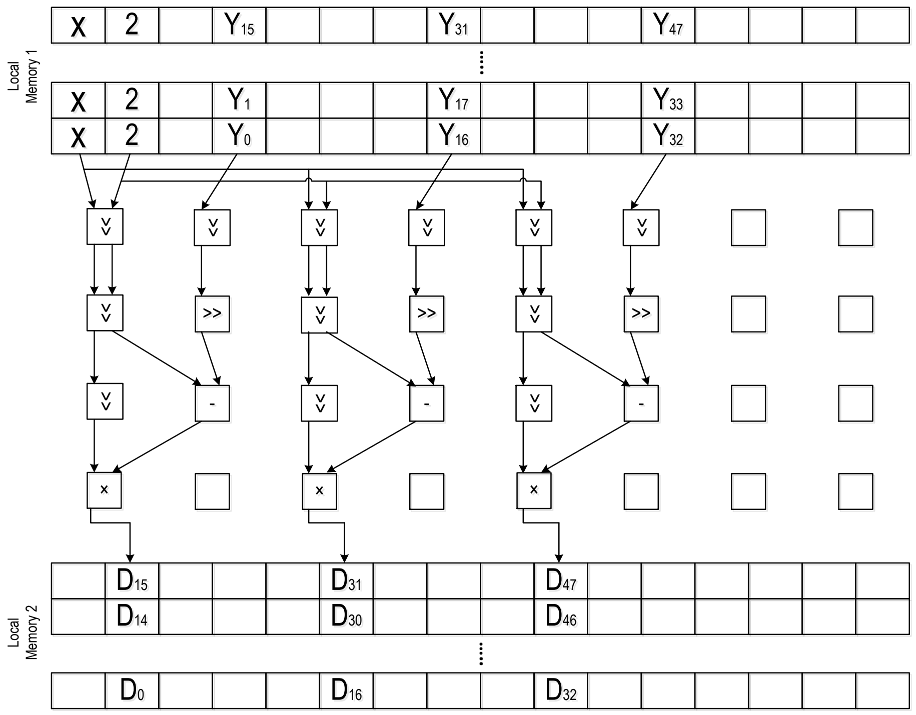


Figure 5.11 *Second context for the Newton-Raphson method*

result of division $(\frac{1}{a^2+b^2})$ is transferred back to the main memory utilizing special DMA operations.

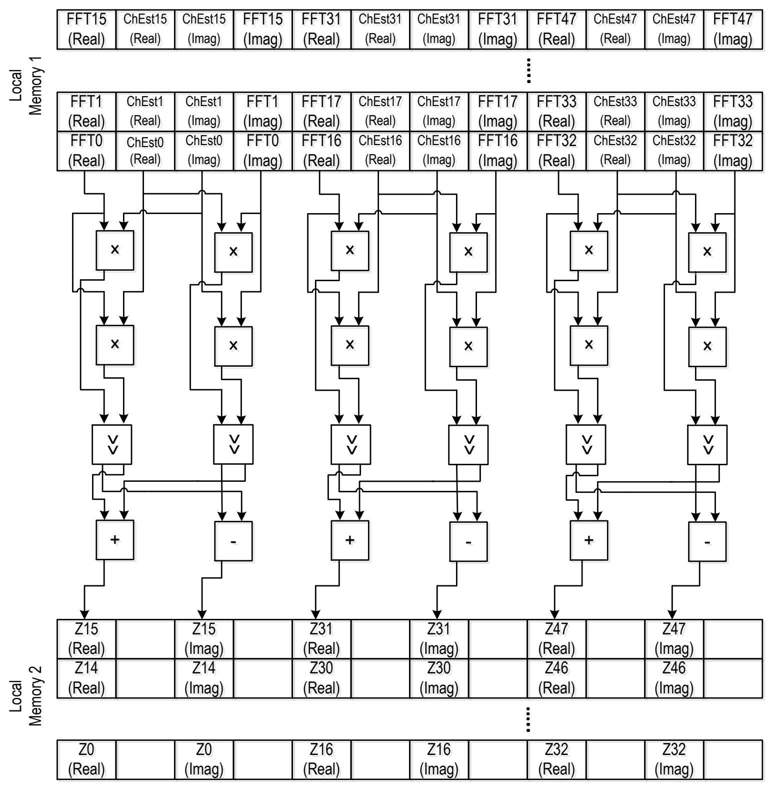


Figure 5.12 *Sixth context for the channel estimation*

In the next stage, the numerator of Equation 5.11 must be computed which is a complex multiplication between demodulated data symbols and complex conjugation of estimated channel frequency response. As it is shown in Figure 5.12, data is transferred back from the main memory to the first local memory for performing complex multiplication in a context where only six columns of its PEs are used.

Once the complex multiplication is implemented, results of division (which is done using Newton-Raphson method) are transmitted back from the main memory to the local memory of CREMA for performing channel equalization. To do so, two more contexts are employed as depicted in Figure 5.13 and Figure 5.14. Within these two contexts, multiplication and shift operation are used to produce the final product. It should be considered that the shift operation that is required after each multiplication is executed only during last two stages for the whole results of channel equalization instead of having a separate context to perform it. Here Res_i stands for equalized received data which is ready for the extracting data bits from it within the next block.

Simulation Results

Channel estimation could be executed completely with eight contexts and seven I/O

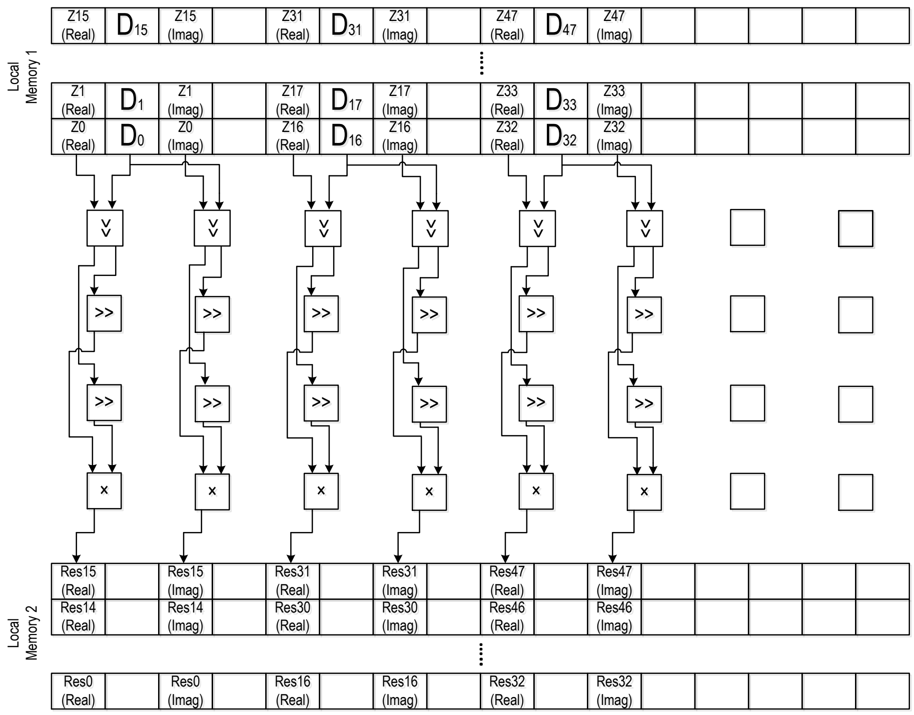


Figure 5.13 *Seventh context for the channel estimation*

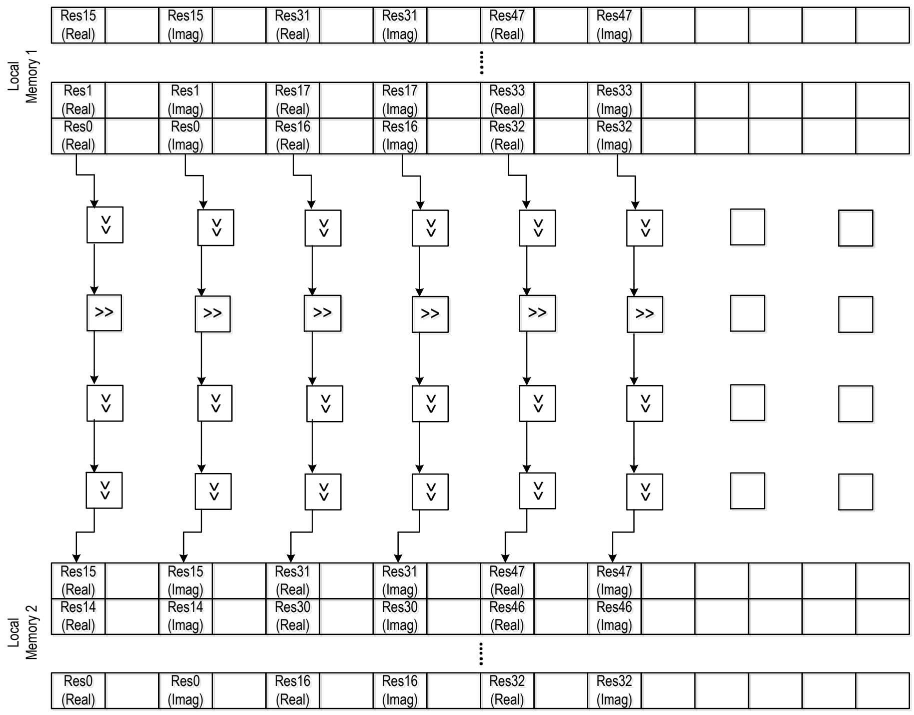


Figure 5.14 *Eighth context for the channel estimation*

buffers. The contexts are composed of one context for loading immediate values in order to perform shift operation, one context for complex multiplication between the received and the transmitted pilots, one context for linear interpolation, two contexts for Newton-Raphson method and three contexts for the last part that is explained above.

Table 5.1 *Cost for different step of Channel Estimation*

S/No	Execution Steps	Clock Cycles (CC)
1	overhead	31 CC
2	stage-1 complex multiplication	5 CC
3	overhead	352 CC
4	stage-2 linear interpolation	30 CC
5	overhead	15 CC
6	stage-3 Newton-Raphson method	30 CC
7	overhead	175 CC
8	stage-4	14 CC
9	overhead	16 CC
10	stage-5	14 CC
11	overhead	12 CC
12	stage-6	14 CC
	Total	708 CC

From Table 5.1, it can be observed that the overall processing of channel estimation requires 708 CC, versus 6274 CC for the COFFEE RISC core.

5.4 Symbols Demapping

Subsequent to performing channel estimation and equalization, we should make decisions about the received data symbols by using decision boundaries. In other words, we should specify the most likely transmitted data bits for each received data symbol. As it was mentioned earlier, there are two ways for making decisions about the received data bits, namely hard decision and soft decision. Here, the first method is used in order to perform symbols demapping for 16-QAM modulation scheme with Gray coded bit mapping which is described in the following.

Transmitted data symbols are composed of two independent real baseband signals (I/Q modulation). Accordingly, the complex plane could be divided into decision regions where each of them consist of the set of points that are closest to a certain symbol (Maximum-likelihood detection). In addition, we are using Gray coding which means all adjoining constellation symbols differ by only one bit. As can be observed from Figure 5.15, the complex plane is divided into the In-phase and Quadrature parts which are equivalent to real and imaginary parts of received data symbols, respectively. Each data symbol is composed of four data bits that might be utilized separately for symbols demapping purpose. Based on the `C` code written

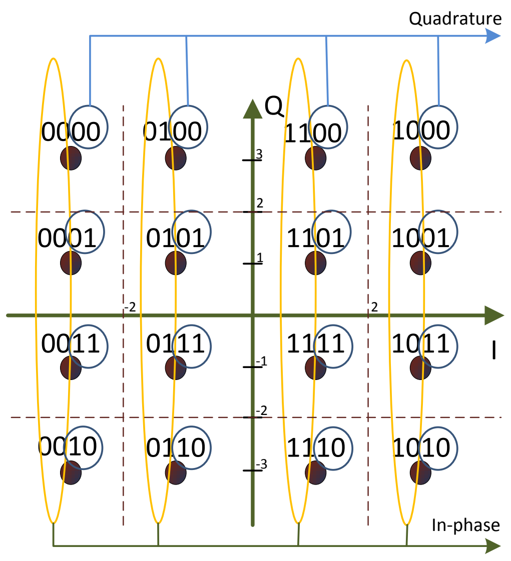


Figure 5.15 *Decision regions for 16-QAM constellation points*

for symbols demapping block using hard decision that is shown in Program 4.3, the two leftmost bits could be detected in the beginning. Subsequent to dividing the complex plane into In-phase and Quadrature areas, it can be seen that there are four zones for each two bits (leftmost and rightmost). Moreover, the first two bits are repeated regularly for each zone of I-axis while the last two bits are same for each region of Q-axis. As it is depicted in Table 5.2, the four bits in each constellation point consist of two bits on I-axis and Q-axis, respectively.

As an example, let's suppose that '$2.8 + i0.8$' is received as a demodulated data symbol in the receiver (instead of '$3 + i1$' due to the AWGN channel). First of all, a decision could be made for the real part and then, imaginary part. Totally, four states (≥ 2, <2 AND ≥ 0, <0 AND ≥ -2 and $<$-2) might occur for each real and imaginary parts. Regarding this example, '2.8' is bigger than '2', thus, '10' are assigned to the first two data bits. At this moment, based on Figure 5.15, received data bits are just varied between four different values ('1000', '1001', '1011' and '1010') which are located in the same area of I-axis. In the next step, an imaginary part could be detected in the same way which is equal to '01' in this case.

Table 5.2 *Gray coded constellation mapping for 16-QAM*

BIT1 BIT2	I	BIT3 BIT4	Q
00	-3	00	+3
01	-1	01	+1
11	+1	11	-1
10	+3	10	-3

Therefore, the actual transmitted constellation points for each data symbol could be found by using at least two comparisons in the best case or at most six comparisons in the worst case.

Simulation Results

The overall processing of symbols demapping block could be executed in at least 1824 CC using COFFEE RISC software for 48 random generated data symbols. However, it should be noticed that we can not generalize the number of clock cycles since it depends on the values of data symbols and the number of comparisons for each of them. Based on our simulations, symbols demapping could be performed for each data symbol in at least 38 CC (ideal case with just two comparisons) or in at most 57 CC (worst case with six comparisons).

```
for(i = 0 ; i < NUMBER_OF_DATA_SYMBOLS ; i++)
        {
        // REAL PART
                if ( RES_REAL >= 2 ){
                        BIT1[i] = 1;
                        BIT2[i] = 0;
                }
                else if ( RES_REAL >= 0 ){
                if( RES_REAL < 2 ){
                        BIT1[i] = 1;
                        BIT2[i] = 1;
                        }
                }
                else if ( RES_REAL >= -2 ){
                if( RES_REAL < 0 ){
                        BIT1[i] = 0;
                        BIT2[i] = 1;
                        }
                }
                else{
                        BIT1[i] = 0;
                        BIT2[i] = 0;
                }
        // IMAGINARY PART
                if ( RES_IMAG >= 2 ){
                        BIT3[i] = 0;
                        BIT4[i] = 0;
                }
                else if ( RES_IMAG >= 0 ){
                if( RES_IMAG < 2 ){
                        BIT3[i] = 0;
                        BIT4[i] = 1;
                        }
                }
                else if ( RES_IMAG >= -2 ){
                if( RES_IMAG < 0 ){
                        BIT3[i] = 1;
                        BIT4[i] = 1;
                        }
                }
                else{
                        BIT3[i] = 1;
                        BIT4[i] = 0;
                }
        }
```

Program 5.3 *C code for Symbols Demapping*

5.5 Synthesis Results

Prior to this section, we described the mapping of three blocks of OFDM receiver using CREMA. In order to evaluate the resource utilization and maximum frequency of each accelerator, FPGA synthesis results need to be analyzed. Table 5.3 collects all useful information about the area utilization of each accelerator on an Altera Stratix-IV EP4SE360H29C2 FPGA device. Moreover, the comparison beteen designed accelerators is shown in Table 5.4 in terms of synthesis frequency in two different temperature and three different categories (clock frequency of CREMA, memory and system). It is clear that the resource utilization and maximum operating frequency are not similar for different application-specific accelerators.

Table 5.3 *Synthesis results of the proposed accelerators on Altera Stratix-IV EP4SE360H29C2 FPGA device*

Accelerator	**Resource Utilization**				
	ALUT	Dedicated logic Regs.	DSP Block	Logic Utilization	Memory Bits
Time Synchronization	17,149 (6%)	11,872 (4%)	48 (5%)	9%	6,930,040 (37%)
Frequency Offset Estimation	19,259 (7%)	13,460 (5%)	80 (8%)	11%	6,930,040 (37%)
Channel Estimation	22,446 (8%)	15,129 (5%)	96 (9%)	12%	6,930,488 (37%)

Table 5.4 *Synthesis frequencies of accelerators generated on Altera Stratix-IV EP4SE360H29C2 FPGA device*

Accelerator	**Max Frequency Slow 85C Model [MHz]**			**Max Frequency Slow 0C Model [MHz]**		
	clk_crema	clk_mem	clk_sys	clk_crema	clk_mem	clk_sys
Time Synchronization	250	164	110	261	168	115
Frequency Offset Estimation	173	178	118	181	183	124
Channel Estimation	151	185	114	159	190	119

The amount of speed-up for each accelerator should be calculated based on the execution time which is the total number of clock cycles divided by the clock frequency. Accordingly, the overall speed-up is the ratio of old execution time to the new execution time for a system or the ratio of execution times for two different systems. As it can be observed from Table 5.5, the execution on CREMA gives a significant speed-up in comparison with COFFEE RISC software. On the other hand, as it was discussed earlier, the amount of speed-up would be reduced because of the communication overhead. It should be mentioned that the clock frequency of COFFEE RISC core is equal to 100 MHz [5].

Table 5.5 *Performance comparison in clock cycles between COFFEE RISC core software and CREMA*

Accelerator	**Application**	**COFFEE Software** (*clk cycles*)	**CREMA** (*clk cycles*)	**Speed-up**
Time Synchronization	Correlation	12800	4017	8×
Time Synchronization	Square Modulus	1680	225	18.6×
Time Synchronization	Overall	15315	5077	7.5×
Frequency Offset Estimation	160-point complex multiplication	4338	485	15.5×
Frequency Offset Estimation	80-point complex multiplication along with shift operation (two contexts)	2338	842	4.8×
Channel Estimation	Overall	6274	708	13.4×

6. CONCLUSION

In this thesis work, application-specific accelerators were designed and implemented for OFDM WLAN baseband receiver while following the IEEE 802.11a standard specifications. In this case, the algorithms implemented are time synchronization, frequency offset estimation, channel estimation and symbols demapping. The mapping of time synchronization block including the calculation of correlations and square modulus on CREMA shows a speed-up of 8× and 18.6×, respectively in comparison with its implementation in COFFEE RISC software. The last part of time synchronization, the maximum value detection, is performed using software in 835 CC. From the implementation of frequency offset estimation block, it could be observed that there is no possibility for CREMA to perform the whole procedure without cooperating with processor software. Thus, the first part (160-point complex multiplication) and the last part (80-point complex multiplication) of this block could be mapped on CREMA which gives us 15.5× and 4.8× speed-ups respectively in comparison with processor software of COFFEE RISC core. On the other hand, since there are no predefined functions like division operation or `ATAN`, there is a need of using other algorithms. Hence, the second part of frequency offset estimation, division operation via CORDIC algorithm, requires 163 CC for each division which is executed using processor software. In addition, the correction factor is implemented by using Taylor series in 5868 CC in processor software. The mapping of channel estimation block on CREMA is executed by utilizing eight contexts and seven I/O buffers in total. Considering the speed-ups achieved by designing the application-specific accelerator for channel estimation block, it gives us an overall speed-up 13.4× versus the COFFEE RISC core. The overall processing of symbols demapping block could be executed at minimum in 38 CC or in at most 57 CC using COFFEE RISC software. Considering the synthesis on FPGA, the maximum operating frequencies for the first three implemented blocks of OFDM receiver are approximately 250, 173 and 151 MHz at $85°C$ and 261, 181 and 159 MHz at $0°C$ and 900mV. Furthermore, their resource utilizations are equal to 9%, 11% and 12%, respectively.

From the simulation results, it can be concluded that the accelerated implementation of wireless communication algorithms gives speed-up by exploiting CREMA architecture's inherent parallelism. Taking into account the implemented application-specific accelerators for OFDM receiver, it is clear that other applications especially ones belonging to digital signal processing could also be mapped using the CREMA platform. Presently, many issues in the design of application-specific accelerators using template-based CGRA remain to be resolved. In future, the algorithms mentioned in this thesis could be implemented on a scaled-up version of CREMA like AVATAR or SCREMA which will give more speed-up and performance improvement. Moreover, a higher number of FFT size for OFDM-based communication systems can be assumed for the designed accelerators. The design of application-specific accelerators can be extended for other intensive kernels related to SDR.

BIBLIOGRAPHY

[1] M-H. Lee, H. Singh, L. Guangming, N. Bagherzadeh, F. J. Kurdahi, E. M. C. Filho, and C. A. Vladimir, ”Design and Implementation of the MorphoSys Reconfigurable Computing Processor”, The Journal of VLSI Signal Processing-Systems for Signal, Image, and Video Technology, Kluwer Academic Publishers, pp. 147-164, Vol. 24, March 2000.

[2] N. S. Voros, M. Hübner, J. Becker, M. Kühnle, F. Thomaitiv, A. Grasset, P. Brelet, P. Bonnot, F. Campi, E. Schler, H. Sahlbach, S. Whitty, R. Ernst, E. Billich, C. Tischendorf, U. Heinkel, F. Ieromnimon, D. Kritharidis, A. Schneider, J. Knaeblein, and W. Putzke-Rming, ”MORPHEUS: A Heterogeneous Dynamically Reconfigurable Platform for Designing Highly Complex Embedded Systems”, ACM Trans. Embed. Comput. Syst. 12, 3, Article 70 (April 2013), 33 pages.

[3] W. Hussain, F. Garzia, and J. Nurmi, ”Evaluation of Radix-2 and Radix-4 FFT Processing on a Reconfigurable Platform”, in Proc. IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems. Vienna, Austria: IEEE, April 2010.

[4] F. Garzia, W. Hussain, and J. Nurmi, ”CREMA, A Coarse-Grain Reconfigurable Array with Mapping Adaptiveness”, in Proc. 19th International Conference on Field Programmable Logic and Applications (FPL 2009). Prague, Czech Republic: IEEE, September 2009.

[5] J. Kylliäinen, T. Ahonen, and J. Nurmi, ”General-purpose embedded processor cores - the COFFEE RISC example”, In Processor Design: System-on-Chip Computing for ASICs and FPGAs, J. Nurmi, Ed. Kluwer Academic Publishers / Springer Publishers, ch. 5, pp. 83-100, ISBN-10: 1402055293, ISBN-13: 978-1-4020-5529-4, June 2007.

[6] R. Airoldi, F. Grazia, O. Anjum, and J. Nurmi, ”Homogeneous MPSOC as Baseband Signal Processing Engine for OFDM Systems”, ©2010 IEEE, 2010.

[7] Man-On Pun, M. Morelli, and C-C Jay Kuo, ”Multi-carrier techniques for broadband wireless communications : a signal processing perspective”, copyright ©2007 by Imperial College Press, December 2007.

[8] J. Heiskala and J. Terry, ”OFDM Wireless LANs: A Theoretical and Practical Guide”, Copyright ©2002 by Sams Publishing, SAMS, 201 West 103rd St., Indianapolis, Indiana, 46290 USA.

[9] S. Afrasiabi Gorgani. Peak Power Reduction In Multicarrier Waveforms. Master Thesis. Tampere 2013. Tampee University of Technology, Faculty of Computing and Electrical. 77 pages.

[10] Supplement to IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: High-Speed Physical Layer in the 5 GHz Band,"IEEE Std 802.11a -1999, 1999.

[11] E. Perahia and R. Stacy, "Next Generation Wireless LANs 802.11n and 802.11ac", 2nd edition, 2013, Cambridge University Press, 452 p.

[12] A. F. Molisch, "Wireless Communication", 2nd Edition, John Wiley and Sons Ltd., 816 p, 2011.

[13] Peled, Abraham, and A. Ruiz, "Frequency domain data transmission using reduced computational complexity algorithms."Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'80. Vol.5, IEEE, 1980.

[14] M. Valkama and M. Renfors, COMMUNICATION THEORY,`http://www.cs.tut.fi/kurssit/TLT-5206/general.html`

[15] L. Liang, J. Shi, L. Chen, and S. Xu, "Implementation of Automatic Gain Control in OFDM digital receiver on FPGA", 2010 International Conference on Computer Design and Applications (ICCDA), vol.4, pp.V4-446,V4-449, 25-27 June, 2010.

[16] T. Hwang, C. Yang, G. Wu, S. Li, and G.Y. Li, "OFDM and Its Wireless Applications: A Survey", IEEE Transactions on Vehicular Technology, vol.58, no.4, pp.1673-1694, May 2009.

[17] L. Harju and J. Nurmi, "Hardware platform for software-defined WCDMA/OFDM baseband receiver implementation,"Computers & Digital Techniques, IET , vol.1, no.5, pp.640-652, Sept. 2007.

[18] J. Rinne, Multicarrier Techniques, `http://www.cs.tut.fi/kurssit/TLT-5706/`

[19] A. Mueen, A. Nath, and J. Liu, ”Fast approximate correlation for massive time-series data”, Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp.171-182, Indianapolis, Indiana, USA, June 6-11, 2010.

[20] J.-J. van de Beek, P.O. Borjesson, M.-L. Boucheret, D. Landstrom, J.M. Arenas, P. Odling, C. Ostberg, M. Wahlqvist, and S.K. Wilson, ”A time and frequency synchronization scheme for multiuser OFDM”, IEEE Journal on Selected Areas in Communications, vol.17, no.11, pp.1900-1914, Nov. 1999.

[21] R. G. Lyons, Understanding Digital Signal Processing. Boston, MA, USA: Addison-Wesley, 1999.

[22] J. W. Cooley and J. W. Tukey, ”An Algorithm for the Machine Calculation of Complex Fourier Series”, Mathematics of Computation, vol. 19, pp. 297-301, 965.

[23] S. Bouguezel, M. Ahmad, and M. Swamy, ”Improved radix-4 and radix-8 FFT algorithms” in Proc. of International Symposium on Circuits and System-sISCAS, vol. 3, pp. 561-564, 2004.

[24] W. Hussain, F. Grazia, T. Ahonen, and J. Nurmi, ”Designing Fast Fourier Transform Accelerators for Orthogonal Frequency-Division Multiplexing Systems”, Journal of Signal Processing Systems for Signal, Image and Video Technology, Springer, ISSN 1939-80, Vol. 69, pp. 161-171, December, 2012.

[25] S. Takaoka and F. Adachi, "Pilot-assisted adaptive interpolation channel estimation for OFDM signal reception", IEEE 59th Vehicular Technology Conference (VTC 2004-Spring), vol.3, pp.1777-1781, 17-19 May 2004.

[26] R. Hajizadeh, K. Mohamedpor, and M.R. Tarihi, ”Channel Estimation in OFDM system Based on the Linear Interpolation, FFT and Decision Feedback”, 18th Telecommunication forum TELFOR , Serbia, Belgrade, November 23-25, 2010.

[27] `http://www.mathworks.com`

[28] M. Renfors and M. Valkama , DIGITAL TRANSMISSION, `http://www.cs.tut.fi/kurssit/TLT-5406/`

[29] Filipo Tosato and Paola Bisaglia, ”Simplified Soft-Output Demapper for Binary Interleaved COFDM with Application to HIPERLAN/2”, HPL-2001-246, October 10th, 2001.

[30] R.V. Nee and R. Prasad, ”OFDM for Wireless Multimedia Communications”, Copyright ©2000 by Artech House, Inc. Norwood, MA, USA.

[31] W. Hussain, T. Ahonen F. Garzia, and J. Nurmi, ”Energy and power estimation of Coarse-Grain Reconfigurable Array based Fast Fourier Transform accelerators”, in Proc. 7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), pp.1-4, 9-11 July 2012.

[32] W. Hussain, T. Ahonen, and J. Nurmi, ”Effects of Scaling a Coarse-Grain Reconfigurable Array on Power and Energy Consumption”, International Symposium on System-on-Chip, Tampere, Finland, pp. 1-5, October 2012.

[33] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins, ”ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix”, Field-Programmable Logic and Applications, vol. 2778, pp. 61-70, ISBN 978-3-540- 40822-2, September 2003,

[34] F. Bouwens, M. Berekovic, A. Kanstein, G. Gaydadjiev, P. Diniz, E. Marques, K. Bertels, M. Fernandes, and J. Cardoso, ”Architectural Exploration of the ADRES Coarse-Grained Reconfigurable Array”, Reconfigurable Computing: Architectures, Tools and Applications, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, vol. 4419, pp. 1- 13, ISBN: 978-3-540-71430-9, 2007.

[35] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho, ”Morphosys: An integrated reconfigurable system for data-parallel and computation-intensive applications”, IEEE Trans. Computers, vol. 49, no. 5, pp. 465-481, 2000.

[36] V. Baumgarte, G. Ehlers, F. May, A. Nuckel, M. Vorbach, and M. Wein- hardt, ”PACT-XPP A Self-Reconfigurable Data Processing Architecture”, The Journal of Supercomputing, vol. 26, no. 2, pp. 167-184, September 2003.

[37] J. M. P. Cardoso, and M. Weinhardt, ”XPP-VC: A C Compiler with Temporal Partitioning for the PACT-XPP Architecture”, in Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream, Editors: M. Glesner and P. Zipf and M. Renovell, Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 864-874, Vol. 2438, ISBN: 978-3-540-44108-3, 2002.

[38] F. Thoma, M. Kuhnle, P. Bonnot, E. M. Panainte, K. Bertels, S. Goller, A. Schneider, S. Guyetant, E. Schuler, K. D. Muller-Glaser, and J. Becker, ”MORPHEUS: Heterogeneous Reconfigurable Computing”, International Conference on Field Programmable Logic and Applications, FPL 2007, pp. 409-414, 27-29 Aug. 2007.

[39] http://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc

[40] http://coffee.tut.fi/docs/COFFEE_Core_USER_MANUAL.pdf

[41] COFFEE RISC Core, http://coffee.tut.fi/docs/COFFEE_pipeline.pdf

[42] D. A. Patterson and J. L. Hennessy, Computer Organization & Design, The Hardware/Software Interface, 1993., pp.307-308.

[43] M. W. Hussain, Design and Development from Single Core Reconfigurable Accelerators to a Heterogeneous Accelerator-Rich Platform. Thesis for the degree of Doctor of Science in Technology, Tampere University of Technology, Tampere, Finland, 2014.

[44] W. Hussain, T. Ahonen, F. Grazia, and J. Nurmi, ”Application-Driven Dimensioning of a Coarse-Grain Reconfigurable Array”, in Proc. NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2011), pp. 234-239, San Diego, California, USA, 2011.

[45] C. Brunelli, F. Garzia, C. Giliberto, and J. Nurmi, ”A Dedicated DMA Logic Addressing a Time Multiplexed Memory to Reduce the Effects of the System Buss Bottleneck”, in Proc. 18th International Conference on Field Programmable Logic and Applications, (FPL 2008), Heidelberg, Germany, pp. 487-490, 8-10 September 2008.

[46] F. Garzia, C. Brunelli, and J. Nurmi, ”A pipelined infrastructure for the distribution of the configuration bitstream in a coarse-grain reconfigurable array”, in Proceedings of the 4th International Workshop on Reconfigurable Communication-centric System-on-Chip (ReCoSoC'08). Univ Montpellier II, July 2008, pp. 188-191, ISBN:978-84-691-3603-4.

[47] W. Hussain, X. Chen, G. Ascheid, and J. Nurmi, ”A Reconfigurable Application-specific Instruction-set Processor for Fast Fourier Transform processing”, 2013 IEEE 24th International Conference on Application- Specific Systems, Architectures and Processors (ASAP), pp. 339-345, 5-7 June 2013, Washington, D.C., USA.

[48] J. E. Volder, ”The CORDIC Trigonometric Computing Technique”, IRE Transactions on Electronic Computers, pp. 330-334, September 1959.

[49] P. K. Meher, J. Valls, T-B Juang, K. Sridharan, and K. Maharatna, ”50 Years of CORDIC: Algorithms, Architectures and Applications”, IEEE Transactions on

Circuits & Systems-I: Regular Papers, vol.56, no.9, pp. 1893-1907, September 2009.

[50] J. E. Meggitt, ”Pseudo Division and Pseudo Multiplication Processes”, IBM Journal, April 1962.

[51] M. D. Greenberg, ”Foundations of Applied Mathematics”, copyright ©1978 by Micheal D. Greenberg, 1978.

[52] V. S. Ryaben'kii and S. V. Tsynkov, ”A Theoretical Introduction to Numerical Analysis”, CRC Press, p. 243, 2006.