DISSERTATION
In Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
from TELECOM ParisTech

Specialization: Communication and Electronics

Muhammad Najam ul Islam

Flexible Baseband Architecture Design & Implementation for Wireless Communication Systems

Defense scheduled on the 7th of October 2010 before a committee composed of:

Reporters	Prof. Emmanuel Boutillon	Université Bretagne Sud
	Prof. Christophe Moy	Supélec, Rennes
Examiners	Prof. Michel Auguin	Université de Nice - Sophia Antipolis
	Prof. Jean-Luc Danger	TELECOM ParisTech
Thesis supervisors	Prof. Raymond Knopp	EURECOM
	Prof. Renaud Pacalet	Lab SoC, TELECOM ParisTech

THESE

présentée pour obtenir le grade de

Docteur de TELECOM ParisTech

Spécialité: Communication et Electronique

Muhammad Najam ul Islam

Flexible Baseband Architecture Design & Implementation for Wireless Communication Systems

Thèse prévue le 7 Octobre 2010 devant le jury composé de :

Rapporteurs	Prof. Emmanuel Boutillon	Université Bretagne Sud
	Prof. Christophe Moy	Supélec, Rennes
Examinateurs	Prof. Michel Auguin	Université de Nice - Sophia Antipolis
	Prof. Jean-Luc Danger	TELECOM ParisTech
Directeur de thèse	Prof. Raymond Knopp	EURECOM
	Prof. Renaud Pacalet	Lab SoC, TELECOM ParisTech

Acknowledgments

I would like to express my sincere gratitude to Prof. Raymond Knopp and Prof. Renaud Pacalet for being my advisors and for providing me with the opportunity to conduct research in the field of my interest. Their continual support and guidance throughout my research work at Eurecom made this thesis a success. I thank Prof. Emmanuel Boutillon and Prof. Christophe Moy for their willingness to serve as the reporters for my research work.

I would also like to express my appreciation for the support of the Open Air Interface team at Eurecom for their technical support during my stay at the institute. I want to express special thanks to all my friends for their support and encouragement.

Lastly and most importantly, I wish to thank my family members for their endless support and blessings without which it wouldn't have been possible to complete my thesis. To them I dedicate this thesis.

Abstract

The past few decades have seen the rapidest growth of technology and diversification of services in the realm of wireless communications. This growth is because of the enhanced use of wireless devices with new functionalities in daily life, while diversity is indebted to the fact that new and advanced applications keep on pouring in. Today, there exists multiple standards for cellular networks (GSM, EDGE, WCDMA etc.), and wireless local area networks (IEEE 802.11a, b, g). Now each of these standards has different carrier frequencies, channel bandwidths and modulation schemes. The phenomenal growth of these modern standards and applications necessitate a flexible hardware platform that is capable to support these diverse standards in the entire wireless communication frequency range. Efficient and flexible baseband processors are imperative for endorsing true multi-standard radio platforms.

We present a generic baseband prototype architecture for Software Defined Radio (SDR) applications that anticipates not only to fulfill current UMTS processing requirements but is also proficient enough to handle 3GPP Long Term Evolution (LTE) processing requirements. The baseband architecture is adept in implementing 2G, 3G, 4G, broadcast communication and wireless LAN standards. The partitioning between HW and SW pursues a general cost-and-complexity versus speed trade-off. The control is in software part of design, which passes the relevant parameters to hardware for specific functionalities. The hardware is designed in such a manner that it would substantiate the most computation intensive task efficiently i.e. meeting the throughput and latency requirements. The hardware is also flexible enough to employ the same baseband processing resources for multiple standards. The presented configurable architecture takes advantage of the commonalities that exist among the different schemes to be implemented but in an efficient manner. The commonalities and disjoints are translated into hardware architecture to come up with a system that performs all the required operations by all the applications. The end product will enable user to perform desired scheme / standard by providing the parameters without going into any details of the architecture. The proposed scheme is far more efficient than having dedicated blocks for each application.

The multi-standard designs should have high performance to comply with the throughput and timing constraints of all the standards with the same HW/SW design. To explore the performance criteria in the baseband design, we present specification, design and implementation of hardware blocks using two approaches, Application Specific Integrated Circuit (ASIC) and Application Specific Instruction Processors (ASIP) designs. The ASIP design provides more flexibility and programmability at the cost of some loss in the performance. We also consider the other existing hardware technologies, take into account their specific advantages and drawbacks, and compare those on the basis of computation type categorization in the baseband design to come up with some guidelines for multi-standard baseband design.

Contents

List of Figures

List of Tables

Acronyms

Here are the main acronyms used in this document. The meaning of an acronym is usually indicated once, when it first occurs in the text.

3GPP	3rd Generation Partnership Project
A/D	Analog to Digital
ADL	Architecture Description Languages
AGU	Address Generation Unit
AHB	AMBA High Performance Bus
ALU	Arithmetic Logic Unit
AMBA	Advanced Microcontroller Bus Architecture
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction-set Processor
AVCI	Advanced Virtual Component Interface
BPSK	Binary Phase Shift Keying
CDMA	Code Division Multiple Access
CPO	Carrier Phase Offset
CWA	Component Wise Addition
CWP	Component Wise Product
D/A	Digital to Analog
DCS	Digital Cellular Service
DDR DRAM	Double Data Rate Dynamic Random Access Memory
DFT	Discrete Fourier transform
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor (ing)
DVB	Digital Video Broadcasting
EDGE	Enhanced Data Rates for GSM Evolution
FCC	Federal Communications Commission
FEP	Front End Processor

FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FT	Fourier Transform
GPP	General Purpose Processor
GSM	Global System for Mobile Communications:
H/W	Hardware
HDL	Hardware Description Language
I/Q	In-phase and Quadrature Components
IDFT	Inverse Discrete Fourier transform
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
IS-95	Interim Standard 95
ISA	Instruction Set Architecture
LAN	Local Area Network
LTE	Long Term Evolution
LISA	Language for Instruction Set Architecture.
LSB	Least Significant Bit
LUT	Look Up Table
MAC	Media Access Control
MAP	Maximum-a-Posteriori
MIMO	Multi-Input Multi-Output
MLE	Maximum Likelihood Estimation
MMSE	Minimum Mean Square Error
MSB	Most Significant Bit
MSS	Memory Sub-system
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
PCI	Peripheral Component Interconnect
PHY	Physical Layer
PP	Pre-Post Processing
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase-Shift Keying
RAM	Random Access Memory
RF	Radio Frequency
ROM	Read-only memory
RTL	Register Transfer Language
S/W	Software
SCR	Software Controlled Radio
SDMA	Space Division Multiple Access
SDR	Software Defined Radio

SIMD	Single Instruction Multiple Data
SISO	Single-Input Single-Output
SoC	System on Chip
UMTS	Universal Mobile Telecommunications System
VCI	Virtual Component Interface
VHDL	VHIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLIW	Very Long Instruction Word
VP	Vector Processor
WCDMA	Wideband Code Division Multiple Access

Chapter 1

Introduction

1.1 Motivation

The last three decades have seen wireless communications as the biggest engineering success, not only in research and development but also in market size and impact on the society. Wireless devices have become an integral part of our daily life, and are also deriving a big chunk of economics. Early on, only cellular phones were considered as wireless communication devices but now things have changed a lot. Wireless computer networks, wireless sensor networks, and wireless positioning systems are a few to name here; that have changed the composition of their applications and most importantly have created new directions of research in communications systems field.

Different types of applications and usages have led to the development of different standards being used in wireless communications systems. Though these systems have almost the same functional blocks, but the approach and hence the algorithms used differs a lot from standard to standard. In wireless communications systems, radio spectrum, radio access technologies and protocol stacks change the behavior among the different systems and networks. Today, there exist multiple standards for cellular networks (GSM, EDGE, WCDMA etc.), and wireless local area networks (IEEE 802.11a, b, g). Now each of these standards has different carrier frequencies, channel bandwidths and modulation schemes. As new standards and applications keep on growing, there is a need for a flexible hardware platform that is capable to support all the different standards in the entire wireless communication frequency range. Software defined radio (SDR), is a reconfigurable radio communication system that can be tuned to any frequency band, and can handle all the modulation schemes in a wide frequency range; thus serving multiple services and communication protocols [Mit95].

The use of flexible architecture that can serve multiple wireless communication standards provide many advantages including:

- Once a configurable architecture is in-place, it is expected to help rapid deployment of new standards. Moreover, it is quite possible that the existing design may compute the new algorithms without any modification. The maintenance cost is also expected to go down by a reasonable factor.

- The new market trends leading to new service-requirements and hence design and development of new systems will not start from scratch; instead these configurable systems will aid a faster development and deployments.

- The SDR design effort is to help the multi-radio handsets to adopt the changing surroundings. This leads to cognitive or opportunistic radio making it possible to use the spectrum more effectively and efficiently. The switching from one access protocol to another becomes seamless to the users.

- A scalable radio design would not only be useful for terminals but also for base stations.

1.2 Contributions

In this thesis report, we focus on the different design options for digital baseband architecture in the context of the SDR applications. The proposed digital baseband architecture is capable of implementing 2G, 3G, 4G, broadcast communication and wireless LAN standards using the same HW/SW architecture. In our design, the digital baseband processing is performed in a HW/SW co-design capable of supporting all the functional requirements at any air-interface and at each stage of SDR processing, from the lowest levels to higher ones. The partitioning between HW and SW follows a general cost-and-complexity versus speed trade-off. Hardware is designed in such a manner that it would support the most computationally intensive task efficiently i.e. meeting the throughput and latency requirements for all the standards in the design. The hardware is also flexible enough to use the same baseband processing resources for multiple standards. The control is in software part of design, which passes the relevant parameters to hardware for specific functionalities. The challenge in the design is to synchronize all the processing at air-interface in an efficient way with minimum resource utilization and high accuracy. The baseband hardware architecture is subdivided in two main parts: a high level control module and a Digital Signal Processing (DSP) engine. The control module is responsible to transfer MAC requests to the DSP engine and control data direction flow. The DSP Engine is responsible for all up-link/downlink signal processing. The baseband architecture along with its design approach is described in detail in this work.

To address the needs of the multi-standard wireless devices, the diverse tasks of the DSP engine of the baseband design range from sample rate matching to viterbi decoding. A thorough study of the target air interfaces led to the identification of a set of functional entities for the digital baseband processing. The identified operations are implemented as seven independent processing blocks, and can be called as hardware accelerators. These include: Pre-processor, Front End Processor, Mapper, Detector, Channel Encoder, Channel Decoder, and Interleaver / De-interleaver. The design approach of each of these blocks is to take advantage of the commonalities that exist among the different standards to be implemented. The commonalities and disjoints are translated into hardware architecture to come up with a system that performs all the required operation by all the applications. The end product will enable user to perform desired scheme / standard operation by providing the parameters without going into any details of architecture. The proposed scheme may not be as efficient as a single dedicated system for a particular standard but it would definitely be far more efficient than having dedicated blocks for each application

The Front End Processor (FEP) block inside the DSP engine is assigned to address all the requirements at the air-interface level that include Channel Estimation, Data Detection, Carrier Phase Offset (CPO) Estimation, and Synchronization. The mechanism of these functions differ from air-interface to air-interface and the different air-interfaces being used for wireless communication devices are Orthogonal frequency-division multiplexing/multiple-access (OFDM/A), Single Carrier FDMA (SC-FDMA), Wideband Code Division Multiple Access (W-CDMA), and Space-division multiple access (SDMA). A flexible FEP block of baseband design is capable to perform the above mentioned operations for all the standards in an efficient and unseen manner to external user i.e. without any requirement to change hardware configuration.

The Fast Fourier Transform (FFT) has been used as building block for air-interface specific architectures both at the transmitter and receiver. Over the last decade, different architectures have been proposed for OFDM receivers with the FFT as the key processing block [LL07] [CN06]. Similar to OFDM, different frequency domain computation architectures for WCDMA / HSDPA have been proposed with performances quite similar to classical time domain equalizers. Following the same methodology, FFT has also been utilized for MMSE turbo equalization in Global System for Mobile Communications(GSM)[LLBL05]. Based on these contribution for individual standards [YGC06], [LIMVS05], [LIZMP05], we analyze and propose a frequency domain processing block capable of catering all the air-interface operations. The implementations of these operations are typically tailored to the standard in question. The detailed design approach, functional description and the implementation of the hardwired, parameterizable FEP block is discussed in the thesis report.

The application specific integrated circuits (ASIC) designs are most optimized and efficient in terms of area, speed and power consumption in comparison to other design methodologies. On the other hand, the flexibility offered is little or even nothing in some cases. The Application Specific Instruction Set Processors (ASIPs) stand as one of the choices for a complex flexible platform design to handle the complex transceiver design tasks on the target software / hardware platform. The ASIPs also provide higher flexibility that is really helpful for hardware design to serve the ever changing wireless communication applications. For ASIP design, if high flexibility and customization for the instruction set architecture (ISA) is required then tools that focus on architecture level optimization may be used. These tools use Architecture Description Languages (ADL). There has been lot of research in this field, but so far the LISA language is the only one that gained commercial acceptance. We used this approach to design an ASIP core called vector processor (VP) to evaluate its usefulness in baseband architectures for SDR applications. The design details along with the results are presented in this document later. The two design approaches adopted for the FEP, i.e. hardwired and ASIP, are thoroughly analyzed in this thesis report as well.

In the context of the SDR baseband design, the set of algorithms to be implemented in the hardware come from diverse wireless standards and waveforms thus providing a greater design challenge. On the other hand, a very high digital processing power is required to implement the flexible and efficient solutions for the SDR applications. In this scenario, the digital hardware component performance in the software radio design becomes a really important aspect to measure the radio's capability. The different hardware components that can be used to carry out these digital processing are digital signal processors (DSPs), field programmable gate arrays (FPGAs), general-purpose processors (GPPs), and application specific integrated circuits (ASICs). The parameters such as cost, speed, flexibility, and power consumption are considered for each of these hardware technologies. Considering these hardware technology options against the set of algorithms in the SDR baseband design along with the performance requirements, some design guidelines are proposed at the end of the report.

1.3 Outline of the report

In the following chapter, the different hardware design approaches for the current and evolving wireless communications systems are described. It also provides a brief history of the architectures that were developed over the period of time for multi-standard communication systems. Chapter 3 focuses on the algorithms that need to be implemented in a flexible baseband transceiver design. Being a multiple standard baseband design, the algorithms for different wireless communication standards are listed, later on these algorithms are grouped together to identify different blocks inside the processing engine of the design.
In chapter 4, our baseband design is explained in detail. We give the design details, explain the reasons of different choices made and analyze the advantages that our approach provides. The detailed hardware design, functional specification and implementation details of the front end processor (FEP) make chapter 5 of this report. In chapter 6, a highly flexible ASIP design using LISATek as the design tool is presented. Then in chapter 7, we make a comparison of the two design approaches that we used for one of our hard-

ware block design. We extend the discussion to the other set of algorithms implemented in other hardware blocks in the SDR design and propose few guidelines to choose the hardware technology for different set of algorithms. In this chapter to strengthen our FEP design case, we present an example of LTE channel estimation methodologies using our hardware block. The last chapter is dedicated to concluding remarks of this work along with some future directions from this research contribution.

Chapter 2

Baseband Architectures for SDR Applications

The need for flexible platforms and their importance with the ever changing technology are the basis for motivation of this work, and is elaborated at the start of this chapter. This chapter also discusses the hardware design approaches for the current and evolving wireless communications systems. We describe the requirements for the different systems, and then highlight the need of flexible architectures to handle multiple applications and hence standards at the same time.

2.1 Software Defined Radio (SDR)

The current trend of convergence between communication and information systems contributes not only to the introduction of new telecommunication products with ever-increasing functionalities, but also to the integration of several means of communication in the same system. Even a standard PC, now-a-days, has facilities like Blue-tooth connection, several standards of high data rate local area network (IEEE 802.11a, b and g), and the possibility of integrating a 2G/3G card. The different types of applications and usages have led to the development of different standards being used in wireless communications systems. Although these systems have almost the same functional blocks, the way these blocks function differs greatly from standard to standard. In wireless communication systems, radio spectrum, radio access technologies and protocol stacks vary from system to system and network to network. Moreover, the evolution of new standards has not stopped and there are no signs of it in the near future, rather there exist incompatible network technologies. These issues give rise to the need of a global system design that can handle most of the existing communication devices (using different wireless communication standards) that exist, if not all. The obvious choice is to have a flexible hardware architecture, that will replace existing dedicated structures for each application. This sums up the concept of Software Defined Radio (SDR). By a broad definition, SDR is a reconfigurable radio communication system that can be tuned to any frequency band, and can handle all the modulation schemes in a wide frequency range; thus serving multiple services and communication protocols [Mit95].

In the beginning, the SDR emerged from military applications where communication between several forces was required without any intervention from the enemy forces. The projects like DARPA's (Defense Advanced Research Projects Agency) SPEAKeasy [LU95] and Joint Tactical Radio System (JTRS) [Mel02] can be viewed as the examples for development of SDR, where multiple air-interfaces with different signal processing techniques were integrated into one platform. Later on, the need and usage spread to civil and daily life applications. The existing cellular phones operating with different standards (UMTS, GSM, DCS-1800, IS-95 etc.) are a typical example of SDR applications. Software defined radio technology has rapidly evolved over the past years and is gaining more and more interest in the industry because of the enormous advantages it offers over conventional radio architectures [Tut99] [BCT99].

Software defined radio and reconfigurability can be defined in many ways, and it is always dependent on the functional support they are providing. Reconfigurability denotes the capability of a system that can dynamically change its behavior, usually in response to dynamic changes in its environment. However in the context of wireless communication, reconfigurability tackles the changeable behavior of the wireless networks and the associated equipment, specially in the fields of radio spectrum, radio access technologies, protocol stacks, and application services.

Though there exists no general consensus about the exact definition of SDR, the Federal Communications Commission (FCC) has proposed to define SDR as a radio that includes a transmitter in which the operating parameters of the transmitter, including the frequency range, modulation type, and maximum radiated or conducted output power can be altered by making a change in software without any hardware change [Kob01]. The Wireless Innovation Forum (previously known as the SDR forum) [WIF] has defined Software Defined Radio in different tiers based on the various capabilities. Each level in the tier definition corresponds to specific capabilities and level of flexibility, the five tiers are listed here :

- **Tier 0** Hardware Radio (HR): The radio is implemented using hardware components only and cannot be modified except through physical intervention.

- **Tier 1** Software Controlled Radio (SCR): Only the control functions are implemented in software. Typically this includes interconnects, power levels etc. but not to frequency bands and/or modulation types etc. Thus limited software controlled functionality.

- **Tier 2** Software Defined Radio (SDR): The software control for variety of modulation techniques, wide-band or narrow-band operations, communication security functions, and waveform requirements of current and evolving standards over a broad frequency range. The frequency bands covered may still be constrained at the front-end requiring a switch in the antenna system.

- **Tier 3** Ideal Software Radio (ISR): The programmability extends to the entire systems with analog conversion only at the antenna, speaker and microphones.

- **Tier 4** Ultimate Software Radio (USR): is defined by the Wireless Innovation Forum for comparison purposes only. Ultimate Software Radios could theoretically switch from one interface format to another in milliseconds.

Thus the simplest example of an SDR is Tier-1, where the baseband processing is performed with fixed hardware. Early dual-mode cell phones fall into this category, as they use two hardware radios to support two different communications standards. In addition to being the most commonly used SDRs, Tier 2 devices constitute the most popularly understood definition of software radios. These systems consist of various processing technologies, such as application specific integrated circuits (ASICs), field programmable gate arrays (FPGAs), and digital signal processors (DSPs). Tier-3 is the most advanced type of SDR that is achievable in the near future. The ideal software radio, has all the capabilities of a Tier-2 radio, except that digital processing starts at the RF range. It is supposed to perform all the demodulation (RF, IF and baseband) entirely in the digital domain.

In table 2.1 the features of few cellular and wireless standards are listed. This gives a glimpse of the diversity of parameters that a multi-standard architecture has to address. On one hand, multiple air-interfaces with their diverse features are to be catered which means a higher flexibility at the digital front end of design. On the other hand, the very high data rates for the evolving standards requires high computational power at high speed.

Standards	**Freq Band**	**Channel Bandwidth**	**Air-Interface**	**Data Throughput**
3 G	2 GHz	5 MHz	WCDMA	384 kbps
HSDPA	2 GHz	5 MHz	WCDMA	14 Mbps
LTE	2 GHz	1.4, 5, 10, 15, 20 MHz	OFDM	100 Mbps
802.11a	5 GHz	20 MHz	OFDM	54 Mbps
802.11b	2.4 GHz	20 MHz	DSSS	11 Mbps

Table 2.1: The features of cellular and wireless technologies

2.2 Baseband Processor Design

Once the common characteristics of the SDR has been listed, we look at the components of a typical wireless communication system to achieve the capabilities mentioned above. The wireless communication system normally is composed of radio front-end, baseband processor and media access control processing unit. The baseband processing takes place next to the radio interface, and from now onwards we only concentrate on the digital baseband part of SDR in this thesis report.
Since the wireless communication systems are designed to support both the transmitter and receiver streams, therefore there are two computational paths inside the baseband processor. In the transmitter mode, the baseband processor carries out channel coding, modulation and symbol shaping after receiving the data from the MAC (Media Access Control) processor. Then the data is forwarded to radio front-end via digital to analog converters. In the receiver mode, the radio signal is filtered and converted into digital baseband signal using the analog to digital converters. Then the

baseband processor carries out the filtering, synchronization, channel estimation and compensation, demodulation etc before transmitting the data to MAC protocol layer. Figure 2.1 [Pro95] shows block diagram of a wireless communication system with its conventional processing units as functional blocks.

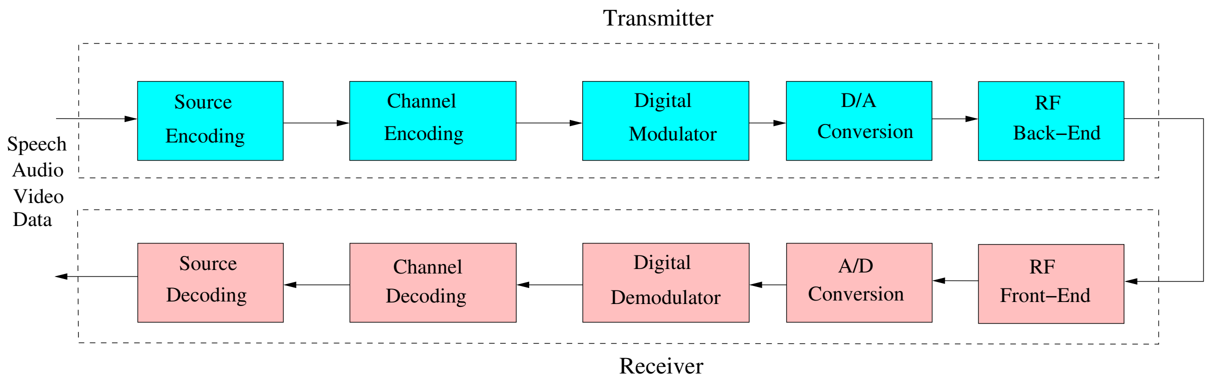

Figure 2.1: Functional Block Diagram of Wireless Communication System

Modern software radio systems require substantial digital processing power for implementing flexible and highly capable systems. With each new emerging wireless communication standard, the computational complexity requirements are increasing. On the other hand, the power budget is also limited for the wireless devices. The high performance hardware designs are required to meet the processing requirements with low power consumption but also need to achieve the required flexibility. Thus the programmable baseband processors are necessary to carry out efficiently multi-standard radio tasks. The flexible design for each block inside the communication system would meet the requirements of all the existing standards and also would anticipate the future requirements in the upcoming or future standards and provide a flexible yet efficient design.

The main four design challenges in the baseband design of software radio are: flexibility, portability, scalability, and performance. These parameters are interrelated but have opposing effects on each other while making the design decisions. In the context of the software radio, flexibility can be defined as the ability to adapt to various current and evolving wireless communication standards and protocols. The flexible solution would help a faster integration of new features in the existing solutions. The flexibility will also make sure of dynamic switching between different standards [BHFN02]. The measure of ease of a system or its component with which it can be transferred from one hardware or software environment to another is termed as its portability [iee91]. With a portable software code to any other hardware architecture, the cost to produce new applications for standards is reduced. Also with a modular design, different levels of portability can be achieved for different implementations across the hardware platforms.

The scalability is defined as the desirable property of a system which indicates its ability to either handle growing amounts of work in a graceful manner or to be readily enlarged [Bon00]. A scalable baseband design of SDR system design would be capable not only to support the existing and evolving standards but will also be intuitive enough to meet the requirements of near future standards. The system should also have the capacity to be enlarged with respect to the new specification by a new standard.

The performance of a baseband design determines how successfully and efficiently it implements the radio functions. It is not only based on computational data throughput but also based a number of factors including energy efficiency, cost, and area requirements. So in turn, it is directly related to flexibility, portability, and scalability. Thus the trade offs must be made to provide a balanced implementation of baseband in the SDR system. Some of the design challenges are listed here:

- High computation performance with flexibility
- Low power consumption
- Address all the existing and evolving standards
- Low computational latency to meet timing constraints

This section provided an introduction to the baseband processing in the SDR design. We looked at the diverse design requirements across multiple standards. In the following section, we look at the existing baseband processors proposed by the industrial and research groups.

2.3 Existing Baseband Processors

There are number of existing processing architecture for SDR systems, which have been designed with different design strategies although the dream goal is to design true SDR system. These solutions include standard processors, DSP designs, configurable units based on FPGA technology and novel execution ideas. Some of the commercial solutions to be listed here do not provide the actual implementation and performance.

Hipersonic, application specific signal processor (ASSP) baseband design, was developed by Systemonic [KWD+02]. It has the following main sections:

- Hard-wired logic for computationally intensive processing
- A configurable DSP unit called OnDSP

The processor design also provides an option of protocol processor that can be used for scalar computations inside the baseband design. Hipersonic is designed to execute the baseband algorithms for IEEE802.11a and HIPERLAN/2 based 5 GHz wireless LAN applications. The configurable DSP unit, OnDSP, is a VLIW/SIMD (Very long instruction word / Single instruction multiple data) based DSP where one slot in the VLIW word controls a vector SIMD unit for complex computations. The OnDSP uses both "horizontal" and "vertical" (Tagged VLIW) [WF96] code compaction which compacts the VLIW word when there are unused fields and uses similarities between neighboring instructions in time.

Philips/NXP acquired Systemonic later on, and an evolved OnDSP processor EVP16 [vBHMKM05] was presented. The EVP16 also utilizes Tagged VLIW instructions to enable compact vector programs. Both the OnDSP and EVP16 are assumed to be incorporated into a larger system-on-chip design with a host processor, dedicated hardware etc. [vBHM+04]. The vector processors perform quite good generally, but their performance is quite low for the complex bit based operations in the digital baseband design. The interleaving and channel decoding in the transceiver design are examples of such computations, that become bottleneck in the system performance.

Icera [ice] came up with the idea that a predictable execution flow can tolerate a long pipeline and that the usage of a long pipeline is necessary to utilize silicon resources efficiently. The solution presented by Icera is named as Deep eXecution Pipeline (DXP) [Kno05]. In this design, the instructions are pointers to an entry in the configuration map which holds different configurations and constants for the 4-way deep execution data path. The configuration map is updated by control instructions. Only dynamic information is issued every clock cycle thus allowing a high operation count per clock cycle. Architecture details as well as information about the memory system has not been disclosed. Processors from Icera do not use any hardware accelerators for any functionality [Kno05].

The Sandblaster [GIL+03] is the name of the processor architecture proposed by Sandbridge Technologies [san]. The solution is based on two execution units: one RISC and the other SIMD based, and supports both communications and multimedia application processing. The compound instruction set uses an underlying VLIW/SIMD architecture. Each instruction is composed of four fields: Load/Store, ALU, Integer multiplier and vector multiplier (SIMD). Thus the architecture can issue up to four simultaneous instructions. The architecture also uses Token Triggered Threading (T^3) to hide the effects of memory access latency. The Sandblaster architecture currently supports 8 threads. In addition to this multi-threading scheme, cache memories are used in the memory system. However, the usage of cache memories results in the unpredictability induced by these memories. If a cache miss occurs, the core needs to be stalled, prohibiting the use of a statically scheduled processor.

The researchers at technical university (TU) Dresden optimized the concept of Task Triggered Architecture (TTA) to propose the the Synchronous Transfer Architecture (STA) [CRS+04]. In STA based architecture, each basic functional unit is connected to other functional units on each side making a connected network. The results in each clock cycle from each module are passed to the next module, and data moves in a synchronous network inside the entire system. The simplicity of the STA architecture is used to generate RTL (register transfer language) and simulation models automatically [MSL+06]. To demonstrate the STA architecture, a floating point baseband processor named SAMIRA [MSL+06] was designed. The SAMIRA processor combines both SIMD and VLIW parallelism by using several SIMD floating point data-paths in parallel. The SAMIRA processor utilizes VLIW instructions to reduce control complexity of the processor. On the other hand, code compaction is used in the SAMIRA processor to reduce the memory footprint associated with VLIW-based processors. The research group has also proposed an updated version of the said design that supports the multimedia applications for LTE / WiMAX terminals [LWB+08].

There have been some proposals from the FPGA vendors to use the reconfigurable logic for the multi-standard processing [xil] [alt]. Some academic research has also been carried out in this regard, however the resource requirement for FPGA configuration [BDBM$^+$04] is enormous. Therefore, the FPGAs should not be used for the whole system design for SDR applications. Also the high power consumption of the FPGAs is not practical in wireless devices.

The survey of the existing SDR baseband architecture reveals that typical implementations of SDR modems are based on the hardware technologies such as general purpose processors (GPPs), digital signal processors (DSPs), field programmable gate arrays (FPGAs) and application specific integrated circuits (ASICs). The computation requirements of the SDR are quite diverse in nature across different standards, and require not only high computational power but high flexibility and low power consumption. It can be safely stated that all the solutions provided so far do not achieve the required level of performance in one way or the other. The ASIC based solutions have limited or no flexibility although achieve high computational power. The DSP solutions are very specific to the internal units design, and the computational power is also limited. The FPGA solutions may be a good option for proof of concept with their low development cost and rapid development cost, but can not be assumed as a solution for mass scale productions.

To achieve the high performance along with flexibility and low power consumption, a hybrid platform solution that uses the different hardware technologies while using efficiently their pros and avoiding their corns might become an interesting option. The GPPs are the most flexible and can serve hugely variant tasks. The irregular operations should also be carried out by the GPPs, however it is worth mentioning again that the power consumption is high in GPPs thus only limited operations should be assigned to the GPPs in a hybrid design. The control operations in the baseband design may typically be assigned to the GPPs. Both GPPs and the DSPs have low performance for the complex bit based operations as is the case in the interleaving and channel decoding in a transceiver design. The FPGAs are power hungry devices compared to their competitors, making it hard to choose them as the solution for baseband design. ASICs offer the most optimized, powerful and computationally efficient digital hardware implementation for the signal processing applications at the cost of flexibility. The ASIC implementations tend to be better suited for highly complex problems or high volume applications or high data throughput requirements, such as cellular phones. In this thesis, we present a hybrid solution that uses general purpose processors, use the optimized DSP solutions and also parameterizable ASICs. The basic approach is to assign one set of tasks to specific hardware technology e.g. control tasks to GPP and highly regular computation intensive tasks to ASICs. We carry on this discussion in chapter 7 as well, where we extend the discussion to Application Specific Instruction Processors (ASIPs) as well.

Summary

In this chapter, we have presented the introduction to the multi-standard architectures. We also discussed the variant nature of the computation requirements by different wireless communication standards. This thesis presents only the digital baseband part of the multi-standard platform [ope], and in the following the focus will be only at the baseband design. In the next chapter, we present the functional requirements of the two main air interfaces OFDM and WCDMA and map those on the functional blocks required in the hardware design.

Chapter 3

Transceiver Tasks for Multiple Standards

This thesis report mainly focuses on digital front end processing inside the baseband design, therefore we discuss the operations required at the air-interface by different standards. The two main interfaces used in the wireless communication devices are $OFDM$ and $WCDMA$. We enlist the algorithms to carry out the transceiver tasks for these interfaces and try to come up with hardware functional blocks that are required for the algorithms. In our baseband design, the hardware block or the Intellectual Property (IP) that is supposed to carry out these functions is named as Front End Processor (FEP). The block, in fact, takes the advantage of the commonalities that exist among the standards. The FEP block is implemented as independent hardware accelerator in the baseband design, and is connected to other blocks via a common interface to carry out the whole transceiver functionality.

First of all, we define the terminology used in this document and also list the operations that are presented in this chapter. Then we list the algorithms or set of operations to carry out the air interface operations.

3.1 Function Definitions

The operations covered in this document are of the following types:

- Discrete Fourier Transform
- Vector Operations
- Energy and Maximum calculations

In the following we list the conventions and notations used in the document along with the definition of the functions.

- Vectors names are uppercase. Scalar names are lowercase.
- If a is a complex number, $x_{min} \leq x_{max}$ two real numbers and p a natural number:
  - $j = \sqrt{-1}$
  - $a = \Re(a) + j \times \Im(a)$: $\Re(a)$ is the real part of a, while $\Im(a)$ is the imaginary part of a
  - $\overline{a}$ denotes the conjugate of a: $\overline{a} = \Re(a) - j \times \Im(a)$
  - $|a|$ denotes the modulus of a: $|a|^2 = \Re(a)^2 + \Im(a)^2 = a \times \overline{a}$
- In a n-components vector X, $X[0]$ is the first component and $X[n-1]$ is the last
- If U and V are two n-components complex vectors, W is a real vector, y a scalar complex number, x_{min} and x_{max} two scalar real numbers and p a natural, $\forall\ 0 \leq i < n$:
  - $\overline{U}[i] = \overline{U[i]}$: conjugate, n-components complex vector
  - $(U \odot V)[i] = U[i] \times V[i]$: component-wise product, n-components complex vector
  - $(U \times y)[i] = U[i] \times y$: component-wise product with a scalar, n-components complex vector
  - $(U \oplus V)[i] = U[i] + V[i]$: component-wise addition, n-components complex vector (note: $\oplus$ is also frequently used to represent the bit-wise exclusive or between bits or words of bits)
  - $(U \ominus V)[i] = U[i] - V[i]$: component-wise subtraction, n-components complex vector

  - $(U \oslash W)[i] = U[i]/W[i]$: component-wise division, n-components complex vector
  - $e(U) = \sum_{i=0}^{n-1} |U[i]|^2$: energy, positive real number
  - $\tilde{e}(U) = \frac{e(U)}{2^{\lfloor \log_2(n) \rfloor}}$: approximate average energy, positive real number
  - $\max(U) = \max_{0 \le i < n} |U[i]|^2$: maximum square of modulus, positive real number
  - $\text{argmax}(U) = \min\{0 \le i < n, |U[i]|^2 = \max(U)\}$, smallest index of maximum square of modulus, natural number
  - $U.V = \sum_{i=0}^{n-1} U[i] \times V[i]$: dot product, complex scalar number

- 1 is used to indicate an all-ones vector if the arithmetic context clearly indicates this.

The Discrete Fourier Transform (DFT) of a complex vector of size n is defined as:

$$DFT_n(X)[u] = \frac{1}{\sqrt{n}} \sum_{v=0}^{n-1} X[v].e^{-\frac{2\pi \times j \times u \times v}{n}}, u \in [0, n-1] \tag{3.1}$$

The Inverse Discrete Fourier Transform (IDFT) of a complex vector of size n is defined as:

$$IDFT_n(Y)[v] = \frac{1}{\sqrt{n}} \sum_{u=0}^{n-1} Y[u].e^{\frac{2\pi \times j \times u \times v}{n}}, v \in [0, n-1] \tag{3.2}$$

Thanks to the $\frac{1}{\sqrt{n}}$ normalization term the property $X = IDFT(DFT(X))$ holds. The IDFT is computed from $IDFT(Y) = \overline{DFT(\overline{Y})}$.

At the air interface, the time and frequency synchronization procedures are similar in OFDM and WCDMA and are explained next.

3.2 Time and Frequency Synchronization

Time Synchronization

Timing synchronization is typically achieved using a sliding window correlation with a known pilot waveform transmitted from a synchronization source. For example, in 802.11a/g/p systems, the so-called *short training*

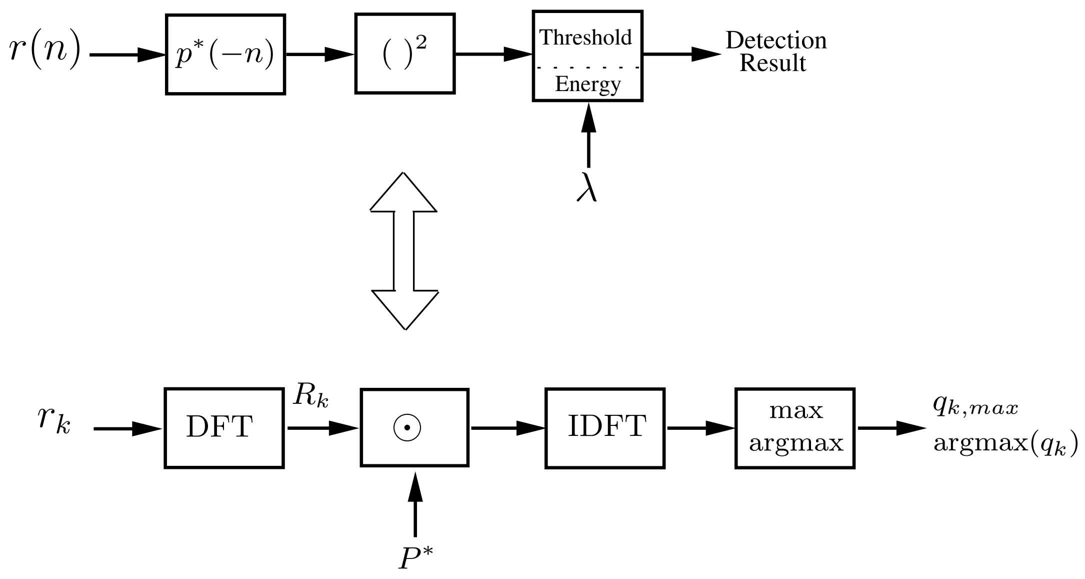

Figure 3.1: Time Synchronization in Time and Frequency Domain

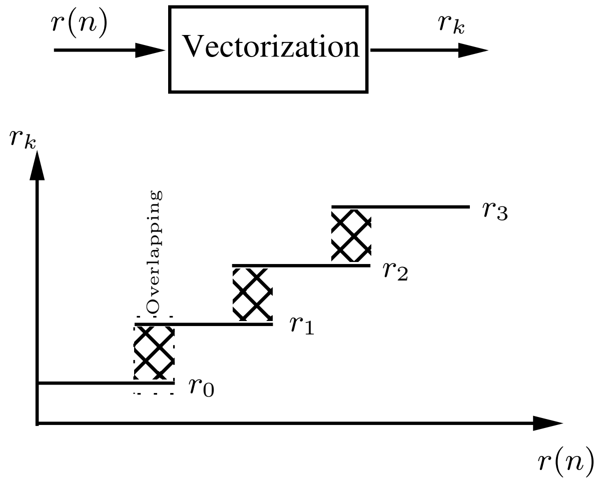

Figure 3.2: Vectorization Process

sequence (STS) is used. The structure of this signal is chosen to offer a compromise between efficient time synchronization and carrier frequency offset estimation. In LTE and WCDMA, the so-called *primary synchronization signal (PSS)* is used for the same purpose.

In the FEP, the time offset estimation process can be efficiently implemented using an overlapping FFT-based correlator, followed by non-coherent peak detection. The overlapping procedure along with the vector / block formation from the signal is depicted in figure 3.2. Let $p(n), 0 \leq n < L_p$ be the reference pilot signal used for time synchronization, and L_p is its length in samples. For 802.11a/g/p L_p is 160 samples, while in LTE L_p is 128 samples (assuming down sampling to 6 resource blocks during initial acquisition). Let $r(n)$ be the complex-baseband received signal and define the detection problem as

$$r(n) = e^{j2\pi\Delta f n} p(n) * h(n - \Delta n) + z(n) \tag{3.3}$$

where Δf is an unknown frequency offset to be estimated subsequently to Δn. Here the effect of the channel is unknown, which infers a non-coherent detection rule. Typically, even the statistical characterization of $h[n]$ is assumed unknown, aside from an underlying assumption on its effective time duration (time delay spread). Define signal segments of dimension N_p samples

$$r_k = [r(kO_p), r(1 + kO_p), \cdots, r(kO_p + N_p - 1)] \tag{3.4}$$

where N_p can be chosen to be $2^{1+\lfloor \log_2 L_p \rfloor}$, that is 256 for both 802.11a/g/p and LTE. O_p would be the overlap factor $N_p - L_p$. This choice of overlap guarantees that the signal component at delay Δn falls entirely in only one segment r_k. Now define

$$R_k = DFT_{N_p}(r_k)$$

to be the DFT of r_k. The correlation operation would be performed as

$$q_k = IDFT_{N_p}(R_k \odot P^*) \tag{3.5}$$

where $P = \mathrm{DFT}(p)$ and

$$p = \left[p(O), p(1), \cdots, p(L_p - 1), \underbrace{0, \cdots, 0}_{O_p \text{ times}} \right].$$

The statistics for detection of Δn are

$$q_{k,\max} = \max(q_k)$$

and

$$\Delta\tilde{n} = \text{argmax}(q_k)$$

It is to be noted that the max and argmax are computed using the $|q_k|^2$ values as defined in section 3.1. Detection of Δn would typically be controlled by a threshold on $q_{k,\max}$ derived from the average received energy, $\lambda(E_k)$, where E_k could be computed as $E_k = \alpha E_{k-1} + (1-\alpha)E(r_k)$, where α controls the memory of the energy computation. One possible rule (not necessarily optimal) would be to choose $\tilde{\Delta(n)}$ when $q_{k,\max} > \lambda(E_k)$ and $q_{k,\max} > q_{k-1,\max}$ and $q_{k,\max} > q_{k+1,\max}$.

Coarse Frequency Offset Estimation

Once time synchronization is achieved (hypothetically) in segment k, estimation of Δf can be attempted using R_k. One possible method for this estimation would be to compute the following statistics based on R_k,

$$q_{k+} = \frac{1}{\sqrt{N_p}}(R_k \odot P_+^*) \cdot \psi(\Delta\tilde{n}) \tag{3.6}$$

and

$$q_{k-} = \frac{1}{\sqrt{N_p}}(R_k \odot P_-^*) \cdot \psi(\Delta\tilde{n}) \tag{3.7}$$

where $\psi(x)[n] = e^{j\frac{2\pi}{N_p}nx}$. The P_+ and P_- are computed as

$$P_+ = DFT_{N_p}(p_+)$$

and

$$P_- = DFT_{N_p}(p_-)$$

where $p_+(n) = p(n)e^{j2\pi\Delta f_{\max}n}$ and $p_-(n) = p(n)e^{-j2\pi\Delta f_{\max}n}$. P_+, P_- and $\psi(x)$ can be computed offline. $\Delta f_{\max}$ is a parameter which represents the maximal frequency offset that a receiver should expect (it can be derived from the standards specification of the system). The statistic $|q_{k+}|^2$ roughly represents the amount of signal energy at a frequency offset of $\Delta f_{\max}$ from the carrier frequency in the direction of the pilot waveform. Based on the three statistics $q_{k,\max}$,q_{k+} and q_{k-} a simple binomial interpolation function can be used to estimate the carrier frequency offset $\Delta\tilde{f}$. The maximum of the binomial is used as an estimate of the frequency offset. After a bit of algebra, this estimate is given by

$$\Delta\tilde{f} = \frac{|q_{k-}|^2 - |q_{k+}|^2}{2(|q_{k-}|^2 + |q_{k+}|^2 - 2q_{k,\max})}\Delta f_{\max} \tag{3.8}$$

This is motivated by the fact that

$$q(u) = p(n)e^{j2\pi\Delta f n} * p^*(-n)e^{-j2\pi un}\Big|_{n=\Delta\tilde{n}}$$

is a convex-$\bigcap$ function. The procedure adopted is depicted in figure 3.3. The point $true_{max}$ represents the actual maximum value of $q(\Delta f)$, while $q_{k,max}$ is the maximum value based on binomial approximation using the parameters q_{k-}, q_k, and q_{k+}.

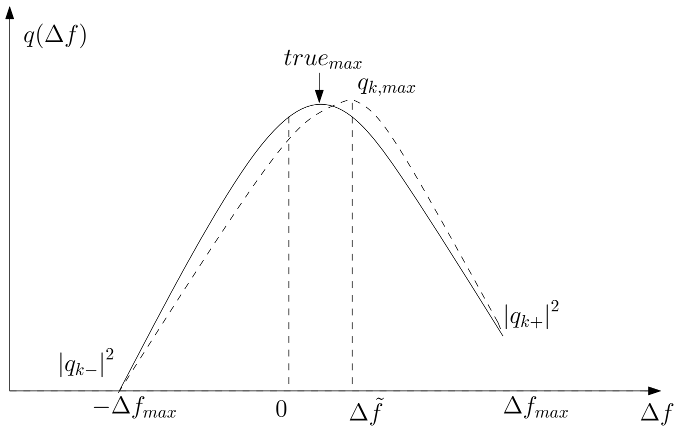

Figure 3.3: Coarse Frequency Offset Estimation

Thus to have time and frequency synchronization in OFDM and WCDMA air-interfaces, the following functional blocks are required:

- DFT, IDFT
- Component-wise-product
- Maximum, arg-maximum computation
- Dot Product

In the following, we describe the procedures for OFDM air-interface and list the required functional blocks that are required to carry out these procedures.

3.3 OFDM : Operations

To transmit the data simultaneously over several sub-carrier frequencies, the Orthogonal Frequency Division Multiplexing (OFDM) method is used by various standards. All the sub-carrier frequencies are mutually orthogonal to each other, thereby signaling on one frequency is not visible on any other sub-carrier frequency. This orthogonality can be implemented by collecting the symbols to be transmitted on each sub-carrier in the frequency domain, and then simultaneously translating all of them into one time domain symbol using an Inverse Fast Fourier Transform (IFFT).

Since each sub-carrier only occupies a narrow frequency band in OFDM, it can be considered to be subject to flat fading and hence avoids a complex channel equalizer. On the receiver side, the signal is transferred back in frequency domain using the fast Fourier transform. Then the impact of the channel on each sub-carrier can be compensated by a simple multiplication in order to scale and rotate the constellation points to the correct position.

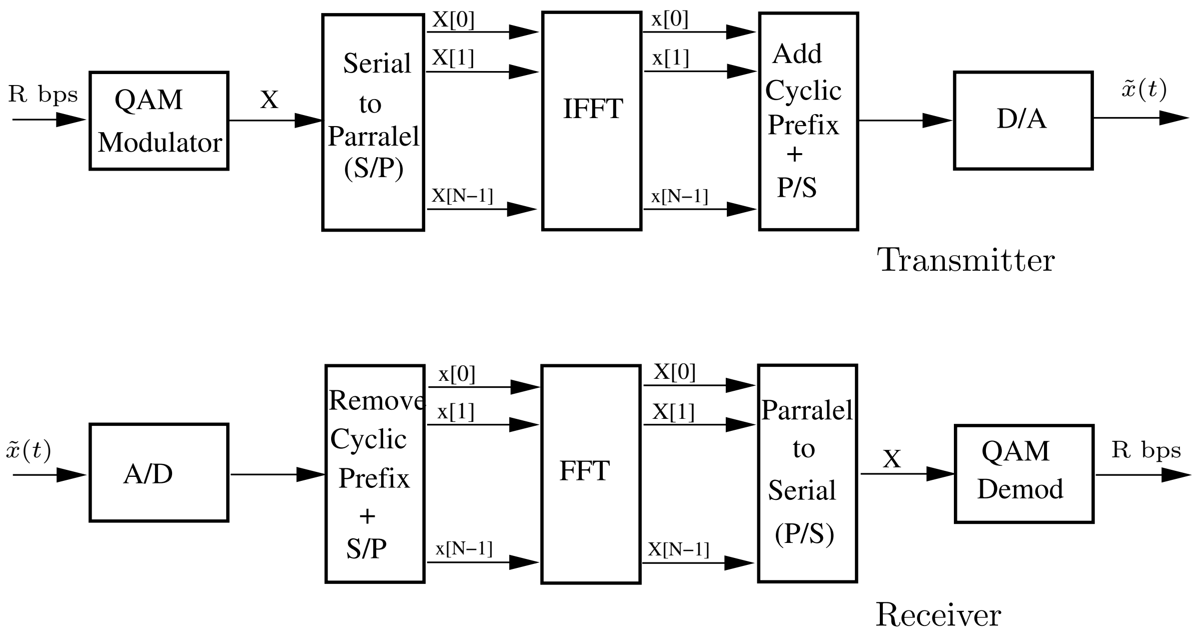

Figure 3.4: OFDM Transmission Chain

In OFDM schemes, a Cyclic Prefix (CP) is added at the beginning of each symbol by copying end portion of symbol and adding it in front of the symbol. This added guard period between the OFDM symbols helps to reduce the impact of multi-path propagation and Inter Symbol Interference (ISI). Thus the effects of ISI are mitigated as long as the channel delay spread is shorter than the cyclic prefix. The OFDM transmission scheme is shown in figure 3.4 [Gol05].

For all the standards using OFDM as the air interface, the DFT and IDFT are integral part of any baseband implementation. In the transceiver implementation, to achieve high performance and efficiency, the DFTs are implemented for the vector sizes that are power of 2. There are numerous efficient algorithms that use the symmetry for specific sizes of input vectors. We discuss the algorithms in the following chapter, when we describe the design of our DFT implementation. The survey study of the different standards shows that the required size can be as small as 64 in case of 802.11 and can be as large as 2048 as is the case of LTE. For the channel synchronization procedure where the samples are used along with the cyclic prefix, or some other operation requiring DFTs of over sampled vectors; even larger vector size DFT than 2048 is required. To make the design more generic, we decide to have the input vector size range for DFT operations to be $8 - 4096$ and input sizes are power of 2.

3.3.1 Pilot Structures in Different Standards

The channel estimation procedure in the receiver is dependent on pilot information that is transmitted along with the data in the transmission schemes. The two basic pilot arrangement schemes used in OFDM systems are illustrated in Figure 3.5. The first one, block-type pilot scheme, is performed by inserting pilot tones into all sub-carriers of OFDM symbols within a specific period. All the other symbols have the pilots at specific location and these pilots are sparse. This structure is used in systems using *802.11 a/g/n/p* and *802.16e*. The second one, diffuse pilot symbols or comb-type pilot channel estimation, is performed by inserting pilot tones into certain sub-carriers of each OFDM symbol, where the interpolation is needed to estimate the conditions of data sub-carriers. This structure is used for the systems like *LTE* and *802.11* for carrier phase estimations. The strategies of the channel estimation procedure for these two basic types are described here.

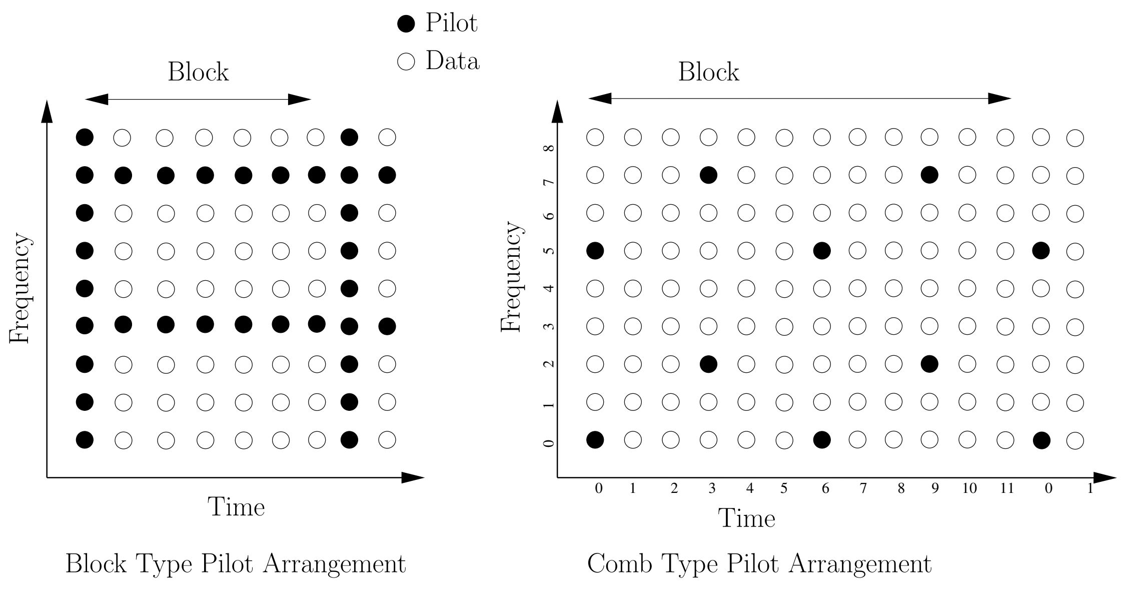

Figure 3.5: Two Basic Types of Pilot Arrangement in OFDM Systems

3.3.2 Channel Estimation for Block Type Pilot Schemes

Channel estimation is achieved using the pilot OFDM symbols (long training symbol or LTS) that are transmitted at the start of each block 3.5 or signal/data burst. For simplicity, consider the case of using a single OFDM symbol for channel estimation, r_{LTS} which is transformed as $R_{\text{LTS}} = DFT(r_{\text{LTS}})$ and given by

$$R_{\text{LTS}} = H \odot P_{\text{LTS}} + Z \tag{3.9}$$

where Z is additive noise, H is an unknown channel to be estimated and P_{LTS} is the frequency-domain representation of the long training symbol. The least-squares estimate of H under no assumption of prior knowledge on H (i.e. even no knowledge of the time delay spread) is

$$\hat{H} = R_{\text{LTS}} \odot P^{*}_{\text{LTS}} \tag{3.10}$$

This estimate can be improved (if required) through additional smoothing/interpolation functions, by taking into account the typical duration (and shape) of the channel response.

3.3.3 Channel Estimation for Diffuse Type Pilot Schemes

In case of the diffuse pilot scheme, the least square estimate can be used as well. The channel estimate is achieved using the symbols that contain pilots inside each data block transmitted. Here pilot resource elements (carriers) are sparsely distributed in time and frequency, the channel estimation procedure requires some form of interpolation. The exact location of pilots for a specific standard and for a specific structure is known e.g. the pilot positions for 2-antennas MIMO in LTE. Let us consider a received symbol r_s which is transformed in frequency domain $R_s = DFT(r_s)$ and is given by:

$$R_s = H \odot X_s + Z \tag{3.11}$$

where Z is additive noise, H is an unknown channel to be estimated and X_s represents the transformed transmitted symbol. However, the known information at the receiver is only at the pilot positions. The least squares estimate at the pilot positions:

$$\hat{H}_p = R_{s,p} \odot P^* \tag{3.12}$$

where $\hat{H}_p$ represents the channel estimate at the pilot positions only, $R_{s,p}$ is the transformed received signal at pilot positions and P^* is the conjugated known pilot information. To get a channel estimate over all the sub-carriers inside the symbol, interpolation schemes are to be utilized. There are multiple interpolation schemes that can be used:

- Linear Interpolation in Frequency Domain
- Pre-defined Filter Interpolation
- Time Domain Interpolation

Linear Interpolation in Frequency Domain

The linear interpolation is the simplest one, which computes the channel estimate at sub-carriers with the help of pilot position channel estimate computed above. A fractional sum of two pilot position estimate is summed up to get the channel estimate at the non-pilot position sub-carriers. To illustrate, a simple example is presented here. Consider the diffuse pilot arrangement shown in figure 3.5, there are 12 symbols transmitted per block and are numbered $\{0, 1, \ldots, 11\}$, while there are 9 sub-carriers inside each symbol and are numbered $\{0, 1, \ldots, 8\}$. Let us consider the symbols 0 to start with. Using the least squares methodology explained, we have the channel estimate at the pilot positions $\{0, 5\}$. The linear interpolation scheme uses the estimate at sub-carrier positions $\{0, 5\}$ to compute the estimate at sub-carrier positions $\{1, 2, 3, 4\}$ and is given by:

$$\begin{aligned}
\hat{H}_{0,1} &= \frac{4}{5}\hat{H}_{0,0} + \frac{1}{5}\hat{H}_{0,5} \\
\hat{H}_{0,2} &= \frac{3}{5}\hat{H}_{0,0} + \frac{2}{5}\hat{H}_{0,5} \\
\hat{H}_{0,3} &= \frac{2}{5}\hat{H}_{0,0} + \frac{3}{5}\hat{H}_{0,5} \\
\hat{H}_{0,3} &= \frac{1}{5}\hat{H}_{0,0} + \frac{4}{5}\hat{H}_{0,5} \qquad (3.13)
\end{aligned}$$

Here, $\hat{H}_{i,j}$ represents the channel estimate in symbol i at sub-carrier j.

Thus linear interpolation takes into account the distance between the position to be estimated and pilot location and then assigns the fractional co-efficients. The sub-carrier positions at the end of sub-carrier that don't have the pilot or reference signals on both sides, use the extrapolation procedure. In extrapolation procedure, the last two pilot positions are taken into account and only the fractional co-efficients change. A detailed example of *LTE* channel estimation is presented at the end of this document in chapter 7. Similarly, the channel estimate for the other positions can be computed inside the symbol and the procedure is followed by the other symbols with the pilots, symbols $\{3, 6, 9\}$ in the block in figure 3.5. Next is to estimate the channel for the symbols that are with out pilots i.e. symbols $\{1, 2, 4, 5, 7, 8, 10, 11\}$ in this case. The temporal interpolation procedure is similar to what we have for the frequency interpolation. The estimate for symbols $1, 2$ is given by:

$$\begin{aligned}
\hat{H}_{1,j} &= \frac{2}{3}\hat{H}_{0,j} + \frac{1}{3}\hat{H}_{3,j} \\
\hat{H}_{2,j} &= \frac{1}{3}\hat{H}_{0,j} + \frac{2}{3}\hat{H}_{3,j} \qquad (3.14)
\end{aligned}$$

$\hat{H}_{1,j}$ represents the channel estimate for symbol 1 in the block at all sub-carrier positions j.

Thus, linear interpolation requires complex vector addition, and vector multiplication with scalars (fractional coefficients). The fractional coefficients are dependent on the distance between the two consecutive pilot positions in the symbol and vary for different pilot structures in the standards.

Time Domain Interpolation

The time domain interpolation, as the name suggests, would require the estimate over pilot position to be transferred in time domain. It follows the fundamental principles of discrete signals that "zero padding" in one domain results in an increased sampling rate in the other domain. For example, the most common form of zero padding is to append a string of zero-valued samples to the end of some time-domain sequence, and if we go to frequency domain the signal will be sampled over string length plus the original length. For the channel estimation of diffuse pilot scheme, the estimated channel response at pilot positions is transferred in time domain, zeros are appended to increase the sampling rate as per symbol length and then again the estimate is transferred in frequency domain. This would require Discrete Fourier Transform (DFT), and Inverse Discrete Fourier Transform (IDFT) functions in the block. For the symbols without the pilot information, the temporal interpolation procedure is followed.

For the case shown in figure 3.5, there is one pilot symbol per 5 sub-carriers. Therefore, the transform is to be spread over 5 times vector length. The procedure followed is given by:

$$\hat{H}_{i,5N_p} = DFT_{5N_p}(IDFT(\hat{H}_{i,N_p})||ZE_{4N_p}) \tag{3.15}$$

The $\hat{H_{x,N_p}}$ represents the channel estimate vector for symbol i at the pilot position and has length equal to number of pilots N_p. First the $IDFT$ takes the channel estimate into the time domain and then zeros are appended. The symbol $||$ is used to represent the appending operation while ZE_{4N_p} describes the zero stream with length $4N_p$. Once the zeros are appended then the resultant vector is converted back in to frequency domain by taking the DFT with vector length $5N_p$.

Pre-defined Filter Interpolation

The pre-defined filter e.g. "sinc" function can be used to spread the channel estimate at pilot positions over the whole symbol. To map the calculated channel response at pilot positions to all sub-carriers:

$$\hat{H} = \sum_{i=0}^{f_l-1} H_{p,i}F_i \tag{3.16}$$

F is a predefined filter (a sinc function in this case), and its length f_l is dependent on coherence bandwidth of the channel. The filter length is dependent on coherent bandwidth. The filter is moved over all the sub-carriers in the symbol and the procedure is repeated for all symbols. These operations require complex multiplication between vectors and scalars and also addition over complex vectors. The temporal interpolation procedure is applied to the symbols without the pilot sub-carriers.

With a detailed discussion on LTE channel estimation (which uses diffuse pilot structure) and performance analysis [STB09], the linear interpolation in frequency domain is better choice. It is simpler as well and performs better.

3.3.4 Significance of Subband Operations

The diffuse pilot structure shown in figure 3.5 is similar to used by OFDMA, the air-interface for the LTE. The pilots are located at a specific distance from each other in pilot symbols. The starting position or the location of the first pilot inside the pilot symbol is set by the upper layers by passing the parameters such as cell ID. The first position may vary for different structures, however the distance between two consecutive pilot sub-carriers is constant for a specific scheme.

To carry out different operations e.g. the component wise product between received pilots and reference signals, the information required by the computing unit includes the first pilot index and the distance between the two pilots. There are other operations, like computing average energy of the sparse pilot symbols in the block type pilot structure to compute the carrier phase offset, where the skipped index computations are carried out instead of consecutive index computations.

We define the subband operation along with offset and skip parameters in the FEP. The subband corresponds to an array of elements in which the offset and sub-sampling factor are constant. The subband span indicates the total length over which the sub-band is formed. The size of sub-band can be less than or equal to its span. The size of subband corresponds to the number of samples / complex numbers to be used in the FEP computations.The offset means the distance between the start of sub-band span and the first sample to be used in the FEP computations. If the value is 0, then it means that we start from the very first sample i.e. start of the sub-band span. The sub-sampling factor gives the value of the skip between the two consecutive samples over the sub-band span. If the sub-sampling parameter is set to 1, then it means that consecutive samples are read out and used in the computations.

3.3.5 Single Antenna: Received Signal

The transformed received signal for symbol n in case of single antenna is given by

$$R_n = e^{j\phi_n} H \odot X_n + Z_n \tag{3.17}$$

where ϕ_n is the phase offset induced by the residual frequency offset and phase noise, H_n, X_n, and Z_n represent Channel Response, Data-transmitted, and noise respectively. The data transmitted is assumed to be QAM-constellation, with variance equal to 1.

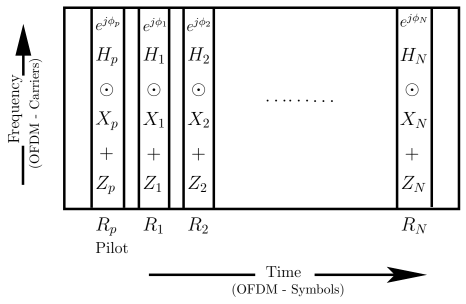

Figure 3.6: OFDM receiver parameters

As per the specifications of our baseband design, the signal fed to the detector (another IP block in the design) at the end of FEP operations should look like:

$$R = X_n + Z_n^{out} \tag{3.18}$$

i.e. FEP operations should extract the transmitted signal from the convolution with the channel response and noise. The FEP block needs to pass the channel estimate and $|\hat{H}|^2$ to the next block if the final step of the detection (log likelihood ratio (LLR) generation) is performed in another block of the baseband design.

3.3.6 Carrier Phase Offset Estimation

Case: Block Type Pilot Structures

In case of block type pilot structures, each symbol apart from the pilot symbols has sparse pilot sub-carriers as is shown in figure 3.5. On the receiver side before data detection, estimation of e^{ϕ_n} must be achieved using additional pilot symbols interleaved in transmitted signal i.e. X_n. We assume that the channel H is unchanged with respect to its state during the preamble period. Let us say that n_p pilot carriers are transmitted out of total n_{sc} sub-carriers, and p defines the pilot positions in the sub-carriers $p \subseteq \{0, 1, \ldots, n_{sc} - 1\}$. The phase offset with respect to the symbol with which channel estimation was performed can be estimated as:

$$A(H)e^{-j\hat{\phi_n}} = \frac{1}{n_p} \sum_{i \in p} (P^*_{\text{LTS}} \odot X_n)[i](R^*_{\text{LTS}} \odot R_n)[i] \tag{3.19}$$

where $A(H)$ is a scalar representing the amplitude of the product, its value is dependent on H and given by:

$$A(H) = \frac{1}{n_p} \sum_{i \in p} \left(|H[i]|^2 + H^*[i]Z_n[i] + H^*[i]Z[i] + Z^*[i]Z_n[i]\right) \tag{3.20}$$

This unknown term can be approximated by

$$\hat{A}(H) = \frac{1}{n_p} \sum_{i \in p} |\hat{H}[i]|^2 \tag{3.21}$$

Thus computing $\hat{A}(H)$ is like average energy calculation over a vector length n_{sc} but selecting n_p elements in the vector with pre-defined format using subband definitions.

The computation of $A(H)e^{-j\hat{\phi_n}}$ can be viewed as averaged dot product of two vectors with subband definitions.

$$A(H)e^{-j\hat{\phi_n}} = \frac{1}{n_p} U.V \tag{3.22}$$

where

$$U = (P^*_{\text{LTS}} \odot X_n)[i]$$

and

$$V = (R^*_{\text{LTS}} \odot R_n)[i]$$

Case: Diffuse Pilot Structures

In case of diffuse pilot structure, the linear interpolation procedure is used for the channel estimate. The channel response including the residual phase offset after coarse frequency estimation and correction for transmit antenna i, symbol l and reference sub-carrier k can be closely approximated as

$$H_{l,i,k} = e^{j2\pi l \Delta f_{\text{fine}}} H'_{l,i,k} \approx (1 + 2\pi l \Delta f_{\text{fine}}) H'_{l,i,k} \tag{3.23}$$

under the important assumption that the residual frequency offset after coarse frequency correction and any Doppler frequency shift due to mobile objects, Δf_{fine}, is small compared to the OFDM carrier spacing (15 kHz in LTE). This linear variation of the channel response between pilot-bearing symbols can again be easily be estimated by linear interpolation between the two channel estimates which include the phase offsets and can be achieved using the temporal interpolation procedures defined in channel estimation subsection 3.3.3. Thus a separate carrier phase offset estimation procedure is not required in case of diffuse pilot structures.

However, these interpolation methods will yield unsatisfactory performance if Δf_{fine} is too significant, especially for high spectral-efficiency transmission (high modulation and coding scheme (MCS)). More sophisticated techniques would have to be considered. Δf_{fine} on the order of 50 Hz should be sufficient even for high spectral-efficiency transmission.

3.3.7 Typical Measurement Procedures in Wireless Systems

The measurements performed by the user equipment (UE) in wireless communication systems (e.g. LTE) make use of the channel estimates and raw signal samples. These operations essentially require

- Wideband energy computation from channel estimates. This is required to compute wideband received signal strength indicators (RSSI) and wideband signal-to-noise ratios (SNR).
- Subband energy computations from channel estimates. This is required to compute subband SNRs for channel-quality indicators (CQI)
- Subband energy computations from received signal vectors during known blank periods to estimate noise levels
- Component-wise-products and dot-products for pre-coding-matrix indicator (PMI) computation

3.3.8 Timing drift adjustment

Timing adjustment is required in order to track the drift induced by slight offsets in sampling frequency at the UE with respect to that used to generate the incoming signal. This can be performed periodically based on the time-domain representation of the channel estimates. Tracking loops could make use of the statistic

$$\Delta\tilde{t} = \text{argmax}\left(IDFT_{N_{\text{RE}}}(\hat{H}_{l,i,j})\right) \tag{3.24}$$

which is the peak location of the dominant path in the impulse response. N_{RE} is the number of carriers or resource elements of the OFDM symbols. For robustness through transmit and/or receive diversity, this could be done using the channel response which is strongest among the set of transmit/receive antenna pairs.

These procedures require DFT, subband energy computations, maximum, arg-maximum computations, component-wise products and dot products. Both the measurement procedures are common in OFDM and CDMA based systems, and hence the computational blocks defined can be used for both the air-interfaces.

3.3.9 Channel Compensation (Equalization)

This procedure is the first step in the data detection procedure in the sense that it prepares (filters) the received signal using the estimated channel response. The outputs of this step are sufficient statistics for detection i.e. R^* and $f(\hat{H})$ and explained in the following.

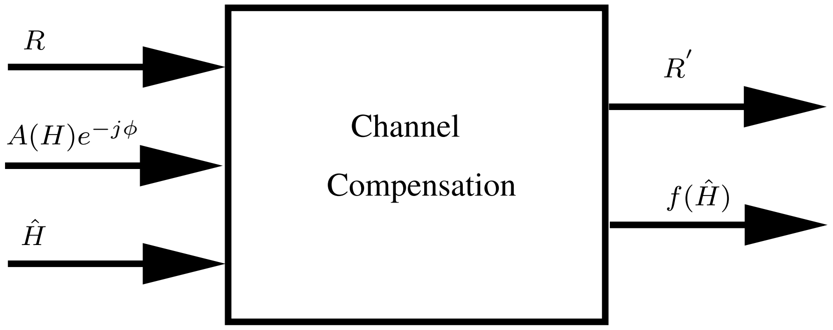

Figure 3.7: Channel Compensation Procedure

Case: Block Type Pilot Structures

Both $A(H)e^{-j\hat{\phi_n}}$ and $\hat{A}(H)$ will be used in conjunction with the channel estimate $\hat{H}$ to provide sufficient statistics for data detection (in the FEP or in another dedicated co-processor). One target for data detection is to produce a "equalized" signal

$$R_{d,n} = (1+\epsilon) \odot X_n + Z_n^{out} \tag{3.25}$$

where $\epsilon[i] \ll 1$ represents the error due to phase and channel estimation, and should be almost negligible. This is approximately equal to what we defined in equation 3.18. 1 is an all ones vector.

The estimated channel and phase offset along with received signal R_n are used in Matched filter receiver technique:

$$\begin{aligned} R' &= A(H)e^{-j\hat{\phi_n}}\hat{H}^* \odot R_n \\ &= A(H)e^{-j(\phi-\hat{\phi_n})}|H|^2 \odot X_n + A(H)e^{-j\hat{\phi_n}}\hat{H}^* \odot Z_n \end{aligned} \tag{3.26}$$

The two operations used are component-wise product between vectors for $\hat{H}^* \odot R_n$, and then resultant is multiplied by $A(H)e^{-j\hat{\phi_n}}$ using component-wise product (vector by scalar).

$e^{-j(\phi-\hat{\phi_n})}$ is termed as phase-error and it is approximately equal to one. The above equation reveals, to get back the desired signal of equation 3.18, we need to divide the above resultant R' by $A(H)$ and $|H|^2$. The operation performed is as follows:

$$R_{d,n} = \frac{R'}{A(H)} \oslash |\hat{H}|^2 \tag{3.27}$$

$$R_{d,n} = e^{-j\hat{\hat{\phi}}}X_n + Z_n^{out} \tag{3.28}$$

where $e^{-j\hat{\hat{\phi}}}$ represents the phase-error, and its effect should be almost negligible. The noise factor is given by:

$$Z_n^{out} = \frac{A(H)e^{-j\hat{\phi_n}}\hat{H}^* \odot Z_i}{A(H)} \oslash |\hat{H}|^2 \tag{3.29}$$

The amplitude of the final resultant in the noise element is of the order of $\frac{1}{H}$. The amplitude of $A(H)$ is canceled out with the division by itself, while a division by $|\hat{H}|^2$ further reduces the amplitude of noise.

Case: Diffuse Pilot Structures

Since the channel estimate includes the carrier phase offset estimation, therefore the compensation procedure is simplified. The above explained procedure is used with out the division by $A(H)$.

3.3.10 Data Detection for Block Type Pilot Structures

The FEP can be used for the final step of data detection (LLR generation) in the case of Gray-coded QAM modulation (all known systems!). Consider first the case of QPSK modulation. Gray coding ensures that the real and imaginary components can be completely decoupled in the detection process [MSM05]. Here we use the statistic

$$R_{d,n} = (A(H)e^{-j\hat{\phi_n}} \cdot ((\hat{H}^* \odot R_n)))/\hat{A}(H) = (1+\epsilon) \odot |H|^2 \odot X_n + Z_n^{out} \quad (3.30)$$

which is sufficient for detection (channel decoding). In the case of 16QAM modulation $R_{d,n}$ is sufficient for two out of four bits, the exact bits depend on the mapping used. The remaining two bits are obtained as

$$R_{d2,n} = \mathrm{abs}(R_{d,n}) \ominus \frac{2}{\sqrt{10}} \cdot (\hat{H} \odot \hat{H}^*) \quad (3.31)$$

which is approximately sufficient for detection [GK10]. The function *abs* represents the absolute value. In the case of 64QAM modulation $R_{d,n}$ is sufficient for two out of four bits, and again the exact bits depend on the mapping used. The next two bits are obtained as

$$R_{d2,n} = \mathrm{abs}(R_{d,n}) \ominus \frac{4}{\sqrt{42}} \cdot (\hat{H} \odot \hat{H}^*) \quad (3.32)$$

and the remaining two bits as

$$R_{d3,n} = \mathrm{abs}(R_{d2,n}) \ominus \frac{2}{\sqrt{42}} \cdot (\hat{H} \odot \hat{H}^*) \quad (3.33)$$

In this section, we listed the procedure to be used for channel estimation, phase estimation and data detection for the OFDM schemes in case of single antenna. The hardware macro blocks required to implement these operations were also identified. Now we move to multiple antenna case, and repeat the procedure.

3.3.11 Multiple Antenna Case

The procedures for the multiple antennas are similar to what we described for single antenna. Here we consider the case of single stream. The received signal for a symbol n can be represented in generic format:

$$R_{i,j,n} = e^{j\phi_{i,j,n}} H_m \odot X_{i,n} + Z_{i,j,n} \quad (3.34)$$

where i represents the transmit antenna index while j is receive antennas index. X, H, Z, and R are the conventional notations for data symbols transmitted, channel response, noise added, and the received signal.

Channel estimation, carrier phase offset correction, and data detection are computed in the same fashion as for single-antenna case and are elaborated in the following.

Channel Estimation

The channel estimation procedure is generalization of the single-antenna case. The pilots are conceived to allow receiver to estimate multiple channels. First the received signals are converted in the frequency domain by using the DFT function. The component-wise-product of pilot symbols and received signal for these pilot symbols results in channel estimate for the respective receive antenna:

$$\hat{H}_j = R_{pp} \odot P^* \tag{3.35}$$

Here *pp* represents the pilot positions. In case of block type pilot structure, the training sequences are used for received signals. On the other hand, for diffuse pilot structures, the pilot positions refer to exact sub-carrier positions inside the symbols containing the reference signals (as explained in 3.3.3). For diffuse or sparse pilot arrangements, the interpolation schemes are applied. The subband operations are applied to estimate the channel. As the schemes work on one stream and one antenna at a time, therefore the procedure and the required functions don't change from single antenna case. The procedure is repeated on each receive antenna for all transmitted bocks.

The phase offset estimation and compensation procedures for block type pilot structures procedure are the same for multiple antennas case as explained for single antenna case (section 3.3.6). The procedure is adopted for all receive antennas, and the same computational blocks are required.

Here we explain the channel compensation and data detection procedure for diffuse pilot scheme.

3.3.12 SIMO Channel Compensation and Data Detection : Diffuse Pilot Structures

Consider now the channel compensation (equalization) process for single-antenna transmission, or the so-called *LTE - Transmission Mode 1.* This procedure could be applied to all downlink physical channels (PBCH, PC-FICH/PDCCH and PDSCH). We assume a receiver with M antennas and received signals

$$R_{n,\mathrm{j}} = (H_j \odot X_n) \oplus Z_{n,j}, j = 0, 1, \cdots, M-1 \tag{3.36}$$

The following statistic is sufficient for data detection in the case of QPSK transmission since LTE employs Gray-coding

$$R_{d,n} = \bigoplus_{j=0}^{M-1} \hat{H}_j^* \odot R_{n,j} = \left((1 \oplus \epsilon) \odot \bigoplus_{j=0}^{M-1} |H_j|^2 \right) \odot X_n \oplus Z_n^{out} \tag{3.37}$$

The real and imaginary components of $R_{d,n}$ can be passed through a de-interleaver procedure prior to channel decoding. Since the FEP is part of a baseband transceiver design, and the units such as interleaving and de-interleaving exist. Also, the baseband design units are supposed connected to each other via a common bus and thus have mutual communication. These units can be used for the said interleaving or de-interleaving procedures, and then the data can be fed back to the FEP.

Similarly to the case of block type pilot schemes (section 3.38), 16QAM data can be detected first using $R_{d,n}$ for two of the bits and using

$$R_{d2,n} = \mathrm{abs}(R_{d,n}) \ominus \frac{2}{\sqrt{10}} \left(\bigoplus_{j=0}^{M-1} \hat{H}_j^* \odot \hat{H}_j^* \right) \tag{3.38}$$

for the remaining two bits, using the abs(·) operator. For 64QAM data the two statistics in addition to $R_{d,n}$ would be

$$R_{d2,n} = \mathrm{abs}(R_{d,n}) \ominus \frac{4}{\sqrt{42}} \left(\bigoplus_{j=0}^{M-1} \hat{H}_j \odot \hat{H}_j^* \right) \tag{3.39}$$

and

$$R_{d3,n} = \mathrm{abs}(R_{d2,n}) \ominus \frac{2}{\sqrt{42}} \left(\bigoplus_{j=0}^{M-1} \hat{H}_j \odot \hat{H}_j^* \right) \tag{3.40}$$

Note that these operations require the presence of an abs(·) operator.

In the event that a generic QAM detector is used outside the FEP, then the statistic

$$R_{d,n} = \left(\bigoplus_{j=0}^{M-1} \hat{H}_j^* \odot R_{n,j} \right) \oslash \left(\bigoplus_{j=0}^{M-1} \hat{H}_j \odot \hat{H}_j^* \right) = ((1 \oplus \epsilon) \odot X_n) \oplus Z_n^{out} \tag{3.41}$$

along with $\left(\bigoplus_{j=0}^{M-1} \hat{H}_j \odot \hat{H}_j^*\right)$ would be passed to the detector.

3.3.13 MISO and MIMO Channel Compensation- Transmit Diversity : Diffuse Pilot Structure

We consider the case of LTE where dual antenna transmit diversity makes use of a space-frequency code based on Alamouti's original space-time code [Ala98]. It is used in all transmission modes for control information (PBCH,PDCCH), and for PDSCH in *Transmission Mode 2*. It is the fall back solution for all transmission modes on the PDSCH when performance enhancements of more complex transmission techniques do not prove to be beneficial because of channel conditions or type of traffic. The receiver in the UE must employ a linear combination of resource elements of the received OFDM symbol. For OFDM symbols not containing the reference signals these are always adjacent elements, whereas in the symbols containing reference signals, the reference resource elements must be skipped, and the linear combination sometimes straddles the reference element. The latter significantly complicates matters in the FEP. For the first case, we first compute the statistics for transmit antenna i and receive antenna j

$$R_{d,n,i} = \bigoplus_{j=0}^{M-1} \hat{H_{i,j}^*} \odot R_{n,j}, i = 0, 1 \tag{3.42}$$

The FEP memory organization should be capable to to allow the following:

$$\begin{aligned} R_{d,n}[2k] &= R_{d,n,0}[2k] + R_{d,n,1}^*[2k+1] \\ R_{d,n}[2k+1] &= R_{d,n,0}[2k+1] - R_{d,n,1}^*[2k] \end{aligned} \tag{3.43}$$

Data detection is then performed in a similar manner to the process described in Section 3.3.12.

The data can be de-interleaved using another block of the baseband design as explained in the previous section. Once FEP has the data in order, it would perform the above combining procedure on adjacent samples.

3.3.14 MISO and MIMO Channel Compensation- Single-layer Precoding

In case of LTE, this channel compensation is used by the UE when configured in *Transmission Mode 6*. We compute the statistics $R_{d,n,i}$ as in Section

3.3.13. For contiguous groups of resource blocks known to the UE, the UE performs the linear combination

$$R_{d,n,g} = R_{d,n,0,g} \oplus (R_{d,n,1,g} \times q_g) \tag{3.44}$$

where g indicates the group of contiguous resource blocks, and $q_g \in \{1, -1, j, -j\}$ is the known precoding constant for group g. In more general precoding strategies q_g could be arbitrary complex numbers (e.g. 4-antennas LTE precoding).

The functions or the macro blocks required in case of multiple antenna case are more or less the same as of single antenna case. In the next section, we describe the CDMA scheme at air interface and also list the operations / macro blocks for the standards that use CDMA scheme.

3.4 CDMA : Operations

Code Division Multiple Access (CDMA) is a multiple access scheme which allows concurrent transmission in the same spectrum by using orthogonal spreading codes for each communication channel.

In a CDMA transmitter, binary data are mapped on to complex valued symbols which then are multiplied (spread) with a code from a set of orthogonal codes. The length of the spreading code is called the spreading factor (SF). In the receiver, data are recovered by calculating a dot-product (despread) between the received data and the assigned spreading code. Since the spreading codes are selected from an orthogonal set of codes, the dot-product will be zero for all other codes except the assigned code. By varying the spreading factor, the system can trade data rate against SNR as a higher SF increases the energy per symbol.

The CDMA analysis can be divided into two sections: low-rate Wideband CDMA (WCDMA) and high-rate WCDMA. We explain the FEP procedures for both the variants of CDMA. First of all, channel estimation procedure is explained which is common in both cases.

3.4.1 Channel Estimation Procedure

The channel estimation can be achieved in a similar fashion to the OFDM case. The overlapped FFT technique is used to compute the channel response. The received signal is transformed in frequency domain and then component-wise multiplication with reference signals (pilots) is carried out.

The method is approximate unless cyclic prefix is used as is the case in TD-SCDMA (Time Division Synchronous Code Division Multiple Access).

3.4.2 Low-rate WCDMA (Long Spreading)

We consider the case of a RAKE receiver, which uses several baseband correlators to individually process several signal multipath components. The correlator outputs are combined to achieve improved communications reliability and performance. The receiver block diagram is shown in figure 3.8.

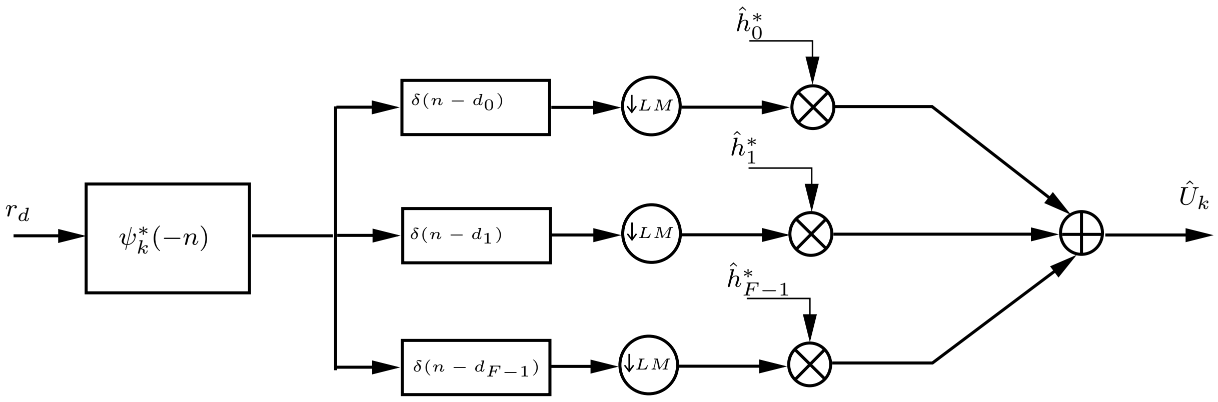

Figure 3.8: RAKE Receiver Block Diagram

The received signal for symbol k and finger offset d_0 is given by:

$$r_{k,d_0} = r(d_0 + kLM, d_0 + 1 + kLM, \ldots, d_0 + (LM-1) + kLM) \quad (3.45)$$

where L is the spreading factor and M is the oversampling factor (2 or 4 in case of RAKE receiver). To retrieve the transmitted signal, the following procedure can be applied:

$$\hat{U}_k = \sum_{f=0}^{F-1} \hat{h}_f^* \sum_{l=0}^{LM-1} r(d_f + kLM + l)\psi_k^*[l] \quad (3.46)$$

i.e. the received signal is first used in a dot-product by the long sequence spreading sequence. A finger f in RAKE receiver is termed as pair of amplitude and delay, and d_f represents the finger offset. Each correlator in the RAKE receiver detects a time-shifted version of the original transmission, and each finger correlates to a portion of the signal, which is delayed by at least one sample in time from the other fingers. The long sequence ψ_k is computed by another block, and BPSK/QPSK modulation scheme is used in WCDMA case.

We split the receiver operation into two elementary operations: convolving the received signal and spread sequence, and channel compensation. The convolution of received signal and spreading sequence shown in figure 3.8 is equivalent to dot product and is given by:

$$R.\psi = \sum_{l=0}^{LM-1} r(d_f + kLM + l)\psi_k^*[l] \tag{3.47}$$

The procedure is repeated for each finger. Therefore, there will be F dot-product units utilized each of length LM.

The second elementary operation is the channel compensation and its is carried out by using the channel estimate for corresponding finger.

$$\hat{U}_k = \sum_{i=0}^{F-1} (R_{k,d_i}.\psi_k)\hat{h}_i^* \tag{3.48}$$

This can be accomplished as a dot product with size F.

The control software manages h and $d_0, d_1, \ldots, d_{F-1}$. The procedure is called finger tracking on h. The initial acquisition would make use of $argmax(h)$, however later on the software will take care of the selection of samples for the set of fingers.

3.4.3 High-rate WCDMA

The procedure for high-rate WCDMA or high speed packet access (HSPA) is shown in figure 3.9. The procedure consists of two main parts: frequency domain equalization (FDE) and despreading or descrambling process.

The FDE procedure [LIZMP05] uses the MMSE process which in fact normalizes the channel estimate computed in the previous step and is given by:

$$H_2 = \hat{H}^* \oslash (\sigma^2 \oplus \hat{H}^*.\hat{H}) \tag{3.49}$$

where σ^2 is the noise variance.

The normalized channel estimate is used with the received signal in frequency domain in component-wise operation to complete the equalization process.

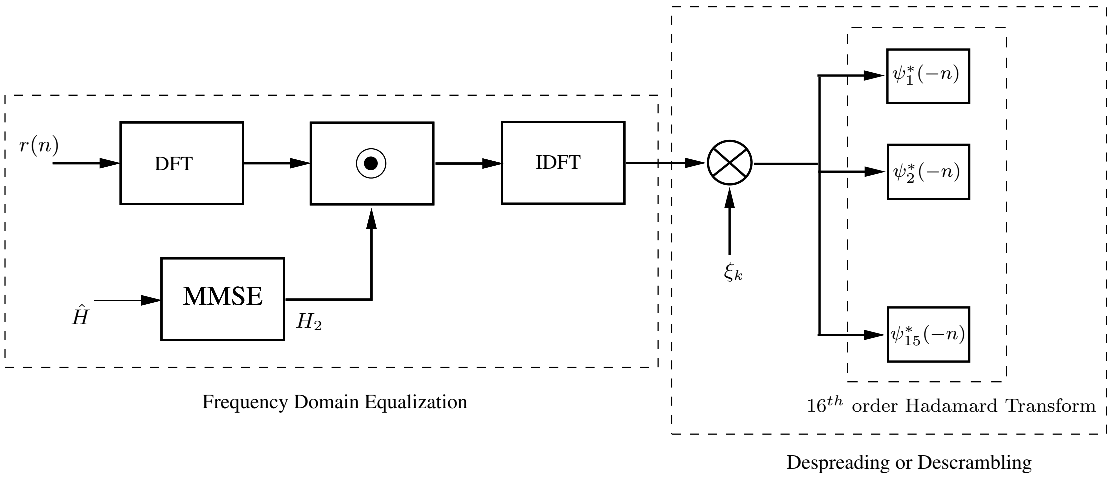

Figure 3.9: High Rate CDMA Receiver

The next step is to despread or de-scramble the equalized signal and retrieve the blocks transmitted. The FDE resultant is multiplied component-wise with the de-scrambling sequence. In figure 3.9, ξ represents the de-scrambling QPSK sequence, and it is fed to FEP from outside (system level or by another block). The next step is to pass the data through the orthogonal spreading sequences. In figure 3.9, $\hat{\psi}_k^*(-n)$ is the orthogonal variable spreading factor (OVSF) which is equivalent to 16^{th} order fast Hadamard transform (FHT). The FHT can be implemented in the FEP or in another block as a separate computational unit. A variable length DFT/IDFT would be an integral part of FEP since it is used by many other operations at the air-interface. Thus DFT unit may also be used to compute the FHT using different set of twiddle factors. The twiddle factors are roots of unit and are multiplicative factors in the DFT/IDFT unit. The last step of the receiver is given by:

$$FHT(r_k \odot \xi_k)$$

There are multiple methodologies used for CDMA in the transceiver design, however our effort was to choose the procedures that are similar to OFDM unit to facilitate the hardware design.

3.5 Extensions: SC-FDMA

Single-carrier FDMA (SC-FDMA) is a frequency-division multiple access scheme and its implementation is not considered in first version of the FEP design. However with slight modifications and/or addition of new modes in the functionality, it can easily be realized with our baseband design.

In figure 3.10, a block diagram of the SC-FDMA is depicted. First of all, the received signal is transformed into frequency domain using a 2^M size DFT block, where M is a natural number. Since the DFT block that we already defined serves all vector sizes that are power of 2 in the range $8 - 4096$, therefore FEP can carry out this operation. In the following step, $N^{'}$ samples are extracted to form a sub-vector or subband, and the size of the subband needs not to be power of 2. In case of LTE, it is defined as

$$N^{'} = 2^{(2+\alpha)}3^{(1+\beta)}5^{\gamma}$$

where α, β, γ are natural numbers. Thus the size of subbands can be multiple of $2, 3$ or 5. In case of LTE, the subband size is always multiple of 12.

The extraction of subbands is followed by the channel compensation procedure using channel estimate. The channel estimate H_2 is computed using the frequency domain equalization (FDE) procedure, explained in section 3.4.3. Then the subbands are transformed into time-domain using the IDFT. Since the size of the subbands or sub-vectors is not power of 2, thus the current formation of the DFT is unable to support this sub-procedure in the receiver. Therefore to carry out SC-FDMA operations, radix-3 and radix-5 need to be implemented along with the radix-2 and radix-4 modes of the DFT block in the FEP. Similarly, for the SC-FDMA transmitter, a DFT block implementing radix-2, radix-3, radix-4 and radix-5 is required. All the other operations needed by SC-FDMA transmitter and receiver are already present in the FEP design.

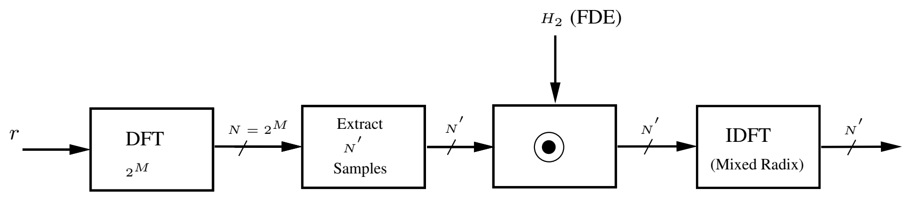

Figure 3.10: SC-FDMA Receiver

3.6 Summary

In this chapter, we have listed the air interface operations for OFDM and CDMA. We also identified the hardware macro units that are required for these inside the processing block FEP, which is supposed to carry out the functionality. The set of hardware macros required are:

- Variable Size Discrete Fourier Transform (DFT) and Inverse DFT
- Component wise addition, multiplication between complex vectors
- Multiplication between a complex vector and scalar
- Division between a complex and a real vector
- Energy and Maximum calculation over vector

It is also worth mentioning that the functions need to have conjugation and subband options for their implementation.

In the next chapter, we present our baseband architecture in detail.

Chapter 4

Flexible Baseband Hardware Design

In this chapter, the baseband design of our multi-standard transceiver platform is presented. The architecture basis for our design are explained in detail along with the design choices made. We also explain the methodology adopted to carry out different set of algorithms, and give a brief description of each of the hardware unit designed to carry out these operations. To start with, the baseband design queries are addressed.

4.1 Baseband Design Choices

In a digital communication system design, the most important phase is to choose target technology, hardware software partitioning level, identifying sub-blocks inside, and the interface among the sub-blocks. So first of all, the choices made for the baseband design are listed along with the reasons of making these choices.

- The SDR baseband design should be portable to different technologies. The aim of the baseband architecture is to first come up with a research tool or an experimental prototype platform, and is not meant for mass scale production. Therefore the selected target technology is FPGAs and not ASICs. This choice is based on reduced design cycle, flexibility, ease of use and lower costs of FPGAs. For the same reasons the higher layers are implemented in software only and run on a host PC. Once validated this architecture will be reworked and adapted to a System on Chip target technology.

- The choice of a specific target technology, i.e. FPGA in this case, constraints the design slightly. The designer is supposed to take into account the specific memory blocks and the DSP slices that come with the specified technology. Thus the design is sub-optimal to some extent in the global context, and the synthesis results must be used with a cautiously approach. Given these facts, all the modules inside the design are perfectly synthesizable with all the existing technologies though optimized for a specific one in some cases.

- The proposed hardware architecture is subdivided in two main parts: a high level control module and a Digital Signal Processing engine. The separation of control and processing not only facilitates simpler design but also makes the system scalable for arrival of new standards or functionalities. The two modules, control and processing, are implemented in high end Virtex-V FPGAs from Xilinx. Figure 4.1 depicts the architectural overview of the system.

- The *interface and control FPGA* in figure 4.1 needs a general purpose processor to handle all the processing inside and the communication through external interfaces. The main processor is also supposed to handle the in-order scheduling of the tasks to the processing blocks. Since the Open Air Interface is an open source design, we chose SPARC - LEON3 processor. Any other general purpose 32-bit processor would have been as useful as is LEON3.

- To interconnect the processing units inside the processing engine requires a generic, standardized and point to point interface. This can be achieved by using any of the standard bus / methodology, we chose Advanced VCI compliant bar which serves the purpose. More details are listed later in this chapter.

- The processing blocks inside the *processing engine FPGA* have a local micro controller to control the data transfers and processing commands inside the block. This would also reduce the communication over the interconnect crossbar. The choice of any small and less resource limited would suffice our needs and we chose 6502. To make the design more generic, all the hardware blocks have the same 8-bit 6502 micro-controller.

- The generic design for each of the units inside the processing engine and the interconnect would allow to add any other block or hardware unit easily into the baseband design.

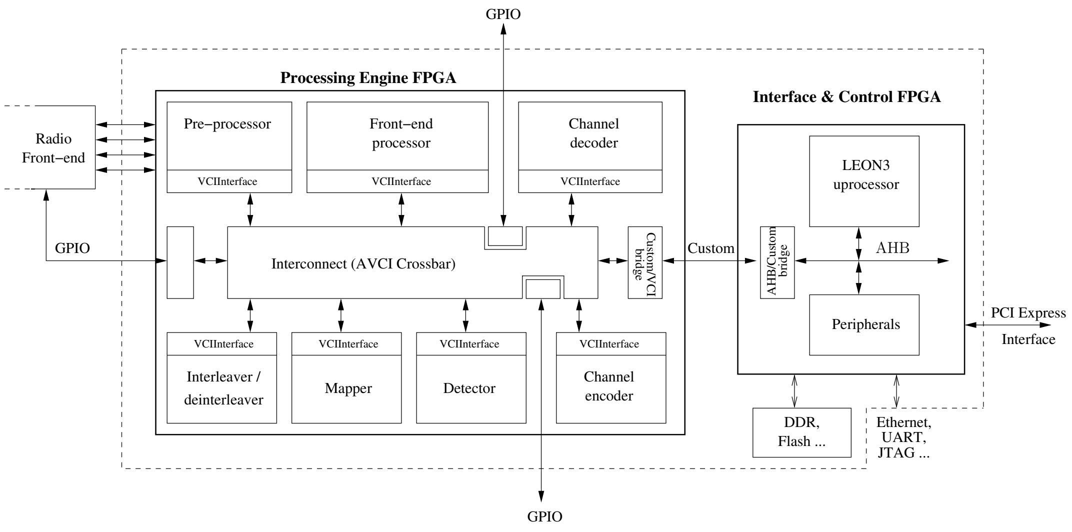

Figure 4.1: Baseband Processing Architectural Overview

After listing the design choices and the reason for the decisions made, we move to the details of our baseband design in the next section.

4.2 Open Air Interface Architecture

The baseband processing takes place between the analog to digital (A/D) and digital to analog (D/A) converters and the raw source coded samples. It implements the digital part of the physical layer. In our baseband design, it is controlled and driven by an embedded microprocessor running a software application whose main purpose is to provide a convenient abstract interface to the Media Access Control (MAC) and upper layers. The MAC and upper layers are implemented as other software applications and run on the host system.
The baseband architecture is separated into two FPGAs which can func-

tion as stand alone (i.e. without host PC). This helps to design and implement the architecture, and is also fruitful to test the design later on. The *Interface & Control FPGA* (control module) is responsible to transfer MAC requests to the *Processing-Engine FPGA* and control data direction flow. The *Processing-Engine FPGA* (DSP Engine) is responsible for all uplink/downlink signal processing.

4.2.1 The Control Module

The control module is based on a SPARC CPU (LEON3 from Gaisler Research) surrounded by its usual peripherals, external memories (DRAM, Flash), a PCI-Express interface and a dedicated interface with the DSP engine. The control module which is in charge of controlling the DSP engine, implements some low-demanding processing (PHY and MAC) and interfaces the system with the host PC through the PCI-Express / ExpressCard interface. Most of the MAC layer processing runs on the host PC while the DSP engine executes most of the PHY layer processing tasks.

4.2.2 The Processing Module

The processing engine or the DSP engine is a collection of data processing IP blocks plugged on a crossbar interconnect. Each hardware block or Intellectual property (IP) is a highly configurable and parameterizable processing unit dedicated to one class of algorithms (Fourier transforms, channel coding, channel decoding, modulation/demodulation, etc.) The chosen interface between the IP blocks and the interconnect is a 64 bits Advanced VCI interface. Each IP block embeds a direct memory access (DMA) engine and an 8 bits micro controller. The synchronization (inter or intra-blocks) is based on a set of interrupts signaling the end of memory transfers and of data processing. The control module programs the DSP engine by configuring the parameters and local software routines of the IP blocks.
A study of the target air interfaces (mentioned in chapter 3), and the communication systems for multi-standard applications (discussed in chapter 2) led to the identification of a set of functional entities for the digital baseband processing. The identified operations are implemented as seven independent processing blocks, and can be called as hardware accelerators:

- Pre-processor
- Frontend Processor
- Mapper

- Detector
- Channel Encoder
- Channel Decoder
- Interleaver / De-interleaver

The pre-processor block is used as an Interface with the external A/D and D/A converters (I/Q multiplexing, control signaling). It also provides several basic signal processing functionalities like filtering, sample rate adjustment, carrier frequency adjustment. The mapper and the detector implement all the modulation schemes ranging from BPSK to QAM256. The (De)Interleaver block, apart from (de)interleaving the data streams with all options in the different standards, performs the frame equalization and rate matching operations. The Front-End-Processor provides the digital signal processing operations at the air-interface, like channel estimation, data detection, carrier phase offset (CPO) estimation etc. The channel encoder implements convolutional encoding, block cyclic codes, turbo coding and m-sequences. The channel decoder IP block realizes trellis-based decoding algorithms; Viterbi and Turbo, to decode convolutional and turbo codes, respectively.

4.2.3 Interconnect

The Interconnect [LIP] is a generic Advanced VCI (AVCI) compliant crossbar. AVCI details are provided in VCI documentation [VSI]. This point-to point communication protocol is a split requests / responses one, supporting out-of-order response transactions. The cells are routed by decoding the VCI address field. The configuration registers and the embedded local memory area of each IP block is mapped in the global memory map of the system. A round robin policy arbitrates between the masters (initiators) requesting the same slave (target).

As shown in figure 4.1, our implementation is not hierarchical, it thus routes all the requests and responses by decoding VCI address and response source identification fields directly. The interconnect uses an internal ROM to keep all the addresses/IDs of the processing blocks for decoding.

The Custom bridge provides interface between the two FPGAs. The signal lines between them are going to be very few but enough to translate the request/response from one FPGA to another in the standard protocols, AVCI and AHB.

In order to configure external devices like RF transmitter/receiver components, ADC/DAC chips, control modules or some external resources; some I/O blocks are also plugged on the interconnect . They implement Serial Parallel Interface (SPI) protocol to communicate with the external devices.

4.2.4 Generic IP Shell

In order to clearly separate the processing on one side and the control and the communication on the other side, our IP blocks all share the same generic model: the IP shell. The IP shell consists of 5 sub-components: an interconnect interface (VCIInterface), a memory sub-system (MSS), an internal DMA engine, an 8-bit micro controller (6502) and a processing IP core. The architecture of the IP shell is depicted in figure 4.2. The main benefits of this organization is the reuse of most of the control and communication logic and the ease of design of the processing IP cores. Only two sub-components are IP-specific: the processing IP core and the memory sub-system. Moreover, the only IP-specific interface is the interface between the IP core and the memory sub-system.

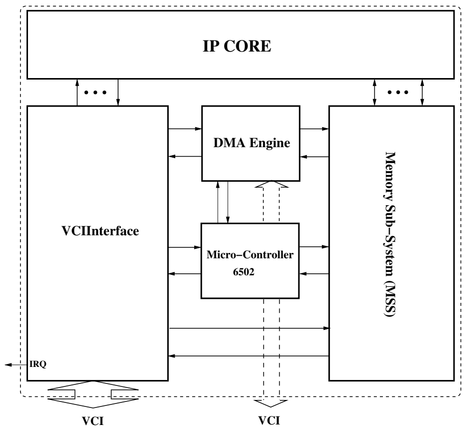

Figure 4.2: Generic IP architecture

The VCI-Interface sub-component is a generic module responsible for interfacing any IP block with the Advanced VCI compliant interconnect. It acts as a master controller of the processing block. In order to separate the communication from the processing, the VCIInterface module has no knowledge of the nature of the processing, neither its duration nor its particular access needs to the shared resources. It implements a target Advanced interface of the VCI (OCB 2.0) standard [VSI]. The VCIInterface module is designed to ease timing closure of the system: every input signal is sampled as soon as possible and every output signal is delivered as early as possible in the master clock cycle. This ensures very short setup and hold times. From IP core designer's perspective, it offers a simple interface for communication through the more complex AVCI protocol.

All the processing blocks contain a memory sub-system, mapped on the global memory map. This distributed memory architecture is complemented by a DMA engine, also embedded in each IP block. The DMA engine handles most of the data transfers between the processing blocks. Each IP block is also optionally controlled by a local small 8−bits micro-controller (6502) capable of driving sequences of data transfers and processing commands. This not only provides efficient and effective computation but also facilitates local data transfers and low-level transactions among blocks without the intervention of the control module. The memory space of this local micro-controller is a subpart of the memory subsystem and is mapped on the global memory map. The main micro-processor starts and halts the local micro-controllers and it downloads and updates their code and data segments.

4.2.5 Software/OS architecture

As explained in the previous section our architecture embeds one main micro-controller and several local micro-controllers inside each IP block. Being a multi-standard application, the said architecture is supposed to be capable of catering multiple air-interfaces at the same time. The software thread for each standard is composed of three procedures: Tx (Transmit), Rx (Receive), Sync (Synchronization). The system is capable of running multiple threads at the same time, and this in-turn means running multiple parallel procedures on the different IPs. This requires the parametrization of each individual IP-micro-controller via the central processor. The computation intensive procedures may introduce too many interrupts from the IP blocks to the LEON3 processor, thus causing a lot of communication overhead and delays. To reduce the number of interrupts generated, procedures for each thread (as defined above) are programmed in the local micro-controllers, which de-localize a part of scheduling from the LEON3 processor to these micro-controllers.

LEON3 hosts Real Time Operating system called *eCos* [eCo], which is highly configurable. The customized software that we run over eCos firstly schedules the tasks over local and main micro-processors to fulfill the timing and throughput requirements for each and every task in the baseband architecture. Secondly, it implements the PCI-Express MAC-layer interface running on the host CPU.

In this section, we have presented an overview of the baseband design in our Open Air Interface multi-standard platform. We gave the design details, and next we give a short description of the each of the hardware accelerators inside the Processing Engine of the baseband design.

4.3 Hardware Accelerators Inside Processing Engine

After discussing the design model for our baseband processing module, we give a brief description of the hardware blocks on the DSP processing engine.

4.3.1 Pre-Processor Block

This block works as an interface of the DSP engine to the external peripherals. The pre-processing block provides the following functionality:

- Provide interface with the external A/D and D/A converters used for I/Q multiplexing, control signaling
- Basic signal processing that includes filtering, sample rate adjustment, and carrier frequency adjustment
- Dual-port swing buffer for transmission and reception of sample streams
- Timing functions i.e. framing, resynchronization and sample synchronous interrupt generation

A re-timing filter with 8 filters in the polyphase filter bank, each comprising 19 filter coefficients, has been designed to cater the requirements of multi-standard radio. The up sampling and down sampling factors can be fractional, and the required resolution is $1Hz$ for a range of $3MHz \leq f_{sampl} \leq 61.44MHz$. The RTL model of this block requires 82 real multipliers and achieves a frequency of $167MHz$ [SKKP10].

4.3.2 Channel Encoder and Decoder

The channel decoder implements trellis-based decoding algorithms; Viterbi and Turbo, to decode convolutional and turbo codes, respectively. Viterbi is maximum-likelihood (ML) algorithm and finds the most likely sequence to have been transmitted. Turbo, on the other hand, is maximum-a-posteriori (MAP) algorithm and finds the most likely symbol to have been transmitted. In practice the MAP algorithm is too complex to implement because of the expensive operations (exponentiation, multiplications etc.), however there exist some hardware-implementation friendly approximations. We thus have chosen Max-Log-MAP (MLM) algorithm for Turbo decoding. The computational operations of Viterbi and MLM algorithms carry similarities and thus a common hardware can be utilized in implementation. Moreover, the channel decoder realizes traceback algorithm for Viterbi and sliding-window algorithm for MLM turbo decoding, which allows efficient reuse of memory resources in both modes.

The channel decoder is in compliance with, but not limited to, IEEE 802.11a/g (WLAN), IEEE 802.16 (WiMAX), 3GPP UMTS and 3GPP UMTS-LTE. The channel decoder is configured with a set of pre-computed parameters, and thus decodes any 64-states and 256-states convolutional codes and 8-states turbo codes. It accepts code rate 1/2 and 1/3, and codes produced by any generator polynomial. The size of the traceback window is $5 \times k$, where k is the constraint length of the code, and the depth of the sliding-window is 16 samples. The number of iterations can be programmed from 1 to 8.

The channel decoder is synthesized with Precision RTL from Mentor Graphics [men] for a Xilinx Virtex V5 VLX220FF1760 device (speed grade -1) [xil]. It takes 6,920 logic cells (5% of total FPGA resouorces) and 24,576 memory bits (0.3% of the total resources). The synthesized operating frequency is 110 MHz. Table 4.1 shows the latency and throughput of the channel decoder in different operating modes [MRP+08].

Mode	**Latency (cycles)**	**Throughput (bits/cycles @ 100MHz)**
Viterbi (k=7)	712	1/8 (12.5 Mbps)
Viterbi (k=9)	3072	1/32 (3.125 Mbps)
Turbo (k=4)	32	1/2 (50 Mbps) in 1 iteration

Table 4.1: Channel decoder results

The channel encoder implements convolutional encoding, block cyclic codes and m-sequences. The tasks are basically carried out by programmable linear control shift registers. For the interleaving, the channel encoder and decoder both use internal interleavers instead of the general purpose hardware block in the baseband design. The reason is to avoid high bandwidth communications among the different hardware blocks, which otherwise might become a performance bottleneck of the whole design. The channel encoder uses bit based interleaving requiring 1-bit per frame, so the communication with the interleaver unit is not that large. The channel decoder has multiple iterations using soft bits requiring large communication with the interleaving unit. To keep the design generic, both encoder and decoder use the internal interleavers.

As stated above, both encoder and decoder use hardwired interleavers thus limiting the flexibility of the design. With the arrival of new interleaving scheme, it would be really hard to include in the current design. The non-regular and highly variant algorithms for the channel encoding and decoding pose a great challenge for a flexible hardwired design.

4.3.3 Interleaver and De-Interleaver

The interleaver/de-interleaver hardware block of the baseband design can be configured to multiple standards including 3GPP UMTS, WiMAX, WiFi and DVB. It uses internal tables to carry out the interleaving and de-interleaving operations for the specific standard, and also capable of performing rate-matching and frame equalization.

4.3.4 Mapper and Detector

The mapper and detector perform the different modulation and demodulation schemes opted by different standards. The blocks use the look up table approach to carry out these operations and use small automata for the control purpose. The schemes supported by these units are BPSK, QPSK, 8PSK, QAM-16, QAM-32, QAM-64 and QAM-256.

The front end processor (FEP), responsible for the digital signal processing operations at the air-interface, like channel estimation, data detection, carrier phase offset (CPO) estimation etc., is described in detail in the following chapter.

Summary

In this chapter, we have provided the brief details for the digital baseband design in our SDR platform. The prototype development procedure, along with the different design options chosen are explained as well. Then a generic IP shell model is presented that is followed by all the hardware blocks in the processing module. The generic model not only helps to have rapid development, easier testing but also will help to add modules in future if required. We have also given a brief description of the individual hardware blocks in the processing engine, and now we move on to one of the most computation intensive processing block inside the baseband design i.e. FEP and explain its design and implementation details in the next chapter.

Chapter 5

The Front End Processor

In this chapter, a detailed design of the digital front end processor (FEP) is explained. We start with the identification of the operations that need to be implemented based on the tasks designated to the FEP in the context of flexible baseband design and explained in chapter 3. Then the analysis of different operations, their design, and implementation is discussed. The large size and complex nature of the functions lead to a multifaceted memory scheme. The memory access and its addressing schema are also part of this chapter. The concluding remarks along with synthesis results of VHDL implementation using *MentorGraphics' precision* tool [men] mark the end of the chapter.

5.1 Identification of Micro-Blocks

The air-interface algorithms explained in the chapter 3 make the basis of the FEP design. The FEP is responsible to take care of air-interface operations including channel estimation, data detection and carrier phase offset for multiple standard baseband platform. The following operations over vectors are defined for this block by the algorithms described earlier in the chapter 3.

1. Discrete Fourier Transform (DFT), Inverse DFT (IDFT)

2. Energy Calculations
3. Maximum, arg-max Calculations
4. Dot Product
5. Component-wise-addition
6. Component-wise-subtraction
7. Component-wise-product
8. Component-wise-division

These functions are over complex input vectors with size range of $\{1 \dots 4096\}$. So basically there are two type of operations in the FEP block: switching between time and frequency domain, and the other processing either before or after the time/frequency domain conversions. Thus the block operations can be divided into two major categories: Time/Frequency conversion (FT Mode) and Pre-Post Processing (PP Mode). Pre and Post here refers to the operations carried out either before or after the domain conversion. It is quite evident from the list of the functions that a lot of multiplications would be taking place inside the FEP block. In the following sections, we describe the design and implementation of both these modes.

As stated before in chapter 4, the development procedure for the said baseband design is to first come up with a research based prototype platform and later on move to the silicon chip; therefore for target technology selection we selected FPGAs. Among the latest technologies available, Virtex-V FPGA by Xilinx was selected.

The selection of number of bits to represent each sample is based on target technology, hardware resources, maximum achievable frequency of end product, and dynamic range of ADC converters. Each complex element of the input and output samples in all baseband blocks is represented by 32 bits with both real and imaginary part as 16 bits in $Q1.15$ format. The least significant bits (LSBs) represent the imaginary part while the most significant bits (MSBs) represent the real part of the complex element. $Q1.15$ means that 16 bits are used to represent the number with 15 LSBs give the value of the real or imaginary part while the '1' MSB gives the sign of that particular value. It is also worth mentioning that any of the baseband IP can use another suitable data representation for its internal configurations while following the global interface data representation.

Next we describe the specifications and designs of the FEP macro blocks.

5.2 Time / Frequency Domain Conversion

The Direct and Inverse Fourier Tranforms of a complex vector X of size N are defined as:

$$DFT_N(X)[k] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} X[n].e^{-\frac{2\pi jnk}{N}}, k \in [0, N-1] \qquad (5.1)$$

$$IDFT_N(Y)[n] = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} Y[k].e^{\frac{2\pi jkn}{N}}, n \in [0, N-1] \qquad (5.2)$$

Thanks to the $\frac{1}{\sqrt{N}}$ normalization term the property $X = IDFT(DFT(X))$ holds. IDFT is computed from $IDFT(Y) = \overline{DFT(\overline{Y})}$ by using the conjugate options. Optionally the conjugate of the input vector may be computed before entering the DFT and the conjugate of the output of the DFT may be computed before storage of the final result in the memory.

The FFT architectures being used in the communication systems can be categorized in two main categories: the pipelined architectures and the memory-based architectures [Coh76] [Ma99] [MW00]. Generally, the memory based architectures are simpler with respect to the hardware complexity. They are composed of butterfly operations, a centralized memory block to store input or intermediate data, and a control unit to handle memory accesses and data flow direction. To keep our design simple and scalable for any changes in the future, we decided to use a memory based design.

The simple and direct computation of the Discrete Fourier Transform (DFT) requires $O(N^2)$ operations where N is the input vector size. However, the Fast Fourier Transform (FFT) algorithm proposed by Cooley and Tukey [CT65], in fact made the use of DFTs quite common in signal processing applications by reducing the complexity from $O(N^2)$ to $O(Nlog_2N)$. Following Cooley and Tukey's algorithm, many other computation simplifications have been proposed for the DFTs. These include radix-2^m algorithms, Winograd algorithm (WFTA) [Win86], Fast Hartley transform (FHT) [KP77] and prime factor algorithms (FPA) [KP77]. However the radix-2, radix-4, and split-radix algorithms [OS89] are most widely used by the developers due to their simple structure, symmetric and periodic computation operations.

A thorough analysis of the DFT/IDFT, the most computation intensive operation inside the FEP block, was carried out to select the *DSP* slices for its mathematical operations. Candidate standards and ease of operations suggest that the number of input samples can be limited as powers of two between 8 and 4096. This allows to calculate DFT using simpler and efficient algorithms such as radix-2 FFT, radix-4 FFT and/or split-radix FFT. Thus the input vector size for the DFT/IDFT block is $\{8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096\}$.

The functionality of DFT macro-block can be globally viewed as DFT-pow-2 and DFT-pow-4, where DFT-pow-2 means the DFT operation for number of input data samples that are power of 2 (i.e. $\{8, 32, 128, 512, 2048\}$). Similarly DFT-pow-4 is meant for number of input samples that are power of 4 (i.e. $\{16, 64, 256, 1024, 4096\}$). The algorithms used to compute each of these sets are explained later in detail.

5.2.1 Roots of unity

With reference to equations 5.1 and 5.2, the factor $e^{-\frac{2\pi jnk}{N}}$ represents the twiddle factors. The roots of unity or the twiddle factors are the equi-spaced roots of unity circle and are used in each multiplication for the computation of the DFT / IDFT as shown in the equations. In the following, we list few operations over the twiddle factors that will be used in the DFT computations. These operations over the twiddles would reduce the complexity of the DFT operations and also reduce the resource utilization by reducing the additions and/or multiplications.

Let $\omega_N = e^{\frac{-2j\pi}{N}}$ be a N^{th} root of unity. The following relations are true:

$$\begin{aligned}
\omega_N^{a+kN} &= \omega_N^a \\
\omega_N^{a+\frac{N}{4}} &= -j.\omega_N^a \\
\omega_N^{a+\frac{N}{2}} &= -\omega_N^a \\
\omega_N^{a+\frac{3N}{4}} &= j.\omega_N^a \\
\omega_N^{-a} &= \overline{\omega_N^a} \\
\omega_N^{\frac{N}{4}-a} &= -j.\overline{\omega_N^a} \\
\omega_N^{\frac{N}{2}-a} &= -\overline{\omega_N^a} \\
\omega_N^{\frac{3N}{4}-a} &= j.\overline{\omega_N^a}
\end{aligned}$$

The preceding relations show that the N N^{th} roots of unity can be deduced from $\frac{N}{8}$ of them (ω_N to $\omega_N^{\frac{N}{8}}$). There are nine different cases to consider:

$$\begin{array}{lll}
a = 0 & & \Rightarrow \ \omega_N^a = 1 \\
0 < a \leq \frac{N}{8} & & \Rightarrow \ \omega_N^a \in \left\{\omega_N \ldots \omega_N^{\frac{N}{8}}\right\} \\
\frac{N}{8} < a \leq \frac{N}{4} & \left(0 \leq \frac{N}{4} - a < \frac{N}{8}\right) & \Rightarrow \ \omega_N^a = -j.\overline{\omega_N^{\frac{N}{4}-a}} \\
\frac{N}{4} < a \leq \frac{3N}{8} & \left(0 < a - \frac{N}{4} \leq \frac{N}{8}\right) & \Rightarrow \ \omega_N^a = -j.\omega_N^{a-\frac{N}{4}} \\
\frac{3N}{8} < a \leq \frac{N}{2} & \left(0 \leq \frac{N}{2} - a < \frac{N}{8}\right) & \Rightarrow \ \omega_N^a = -\overline{\omega_N^{\frac{N}{2}-a}} \\
\frac{N}{2} < a \leq \frac{5N}{8} & \left(0 < a - \frac{N}{2} \leq \frac{N}{8}\right) & \Rightarrow \ \omega_N^a = -\omega_N^{a-\frac{N}{2}} \\
\frac{5N}{8} < a \leq \frac{3N}{4} & \left(0 \leq \frac{3N}{4} - a < \frac{N}{8}\right) & \Rightarrow \ \omega_N^a = j.\overline{\omega_N^{\frac{3N}{4}-a}} \\
\frac{3N}{4} < a \leq \frac{7N}{8} & \left(0 < a - \frac{3N}{4} \leq \frac{N}{8}\right) & \Rightarrow \ \omega_N^a = j.\omega_N^{a-\frac{3N}{4}} \\
\frac{7N}{8} < a < N & \left(0 < N - a < \frac{N}{8}\right) & \Rightarrow \ \omega_N^a = \overline{\omega_N^{N-a}}
\end{array}$$

These relations show, given $\frac{N}{8}$ of twiddle factors the remaining $\frac{7N}{8}$ can be computed with simple conjugation and negation operations. This helps to store only $\frac{1}{8}$ twiddle factors in the FEP memory. The saving is reasonable, given the largest input vector size for our FT module being 4096.

5.2.2 FT Mode Computations

In Fourier transform (FT) mode, the FEP computes the DFT and the IDFT. Some of the key design issues in the variable length DFT computation are :

- High performance butterfly execution to meet throughput requirements
- Efficient data address generator that is capable to support variable length inputs in the same input memory pattern
- Smart multi-bank memory organization to support conflict free fast data access
- Efficient address generator for the variable length twiddle factor access

The throughput requirements, in view of the current and evolving wireless communication standards' timing constraints, are $1 - sample/cycle$. The FEP uses Radix-4 (for input size power-of-4) and Mixed-Radix (for input size power-of-2) algorithms to compute the DFT and the IDFT. The throughput requirement, with the given input vector size range, forces FEP to operate on 8 samples per cycle [MKKP07]. This, in turn, means memory access of eight input samples and eight twiddle factors in each cycle. However, a close look at the twiddle access by radix-2 and radix-4 algorithms reveals that for any number-of-input-samples N, 3 twiddles factors per cycle (instead of 8 twiddle factors for 8 input samples) are enough to compute DFT / IDFT with a simple alteration in the sample access scheme and using the properties of twiddle factors (listed above). Thus the FEP taking advantage of these two factors and using few more signals, accesses 3-twiddles per cycle and 8-input samples.
The input parameters required by DFT macro-block to carry out its operation are:

- size of the input vector (range $1 \ldots 4k$)
- memory addresses (for input samples, output samples and twiddle factors)
- conjugation flags before and after DFT operation

5.2.3 Input Vector Size - Power of 4

When input vector size is power of 4, then radix-4 algorithm is used with decimation in frequency mode. Radix-4 computes FFT for vector sizes that are powers of 4 in M stages (where $M = log_4 N$), N being size of the input vector. Each stage is composed of complex number additions and multiplications, known as butterfly operation. The radix-4 butterfly operates on 4 input samples and generate as many output samples as shown in figure 5.1. The figure shows four input samples I_0, I_1, I_2, I_3 in $Q1.15$ data format and the four resulting output samples in $Q3.14$ format (The data representation for the resultant samples is explained later). There are $N/4$ butterfly operations per stage for radix-4 case.

The intermediate results of the DFT operation i.e. results of each and every stage, are stored in the memory and read out in the next stage as input samples for that particular stage. The Open Air Interface baseband design allows the individual IPs to store the inner results in any format that suits their respective requirements. To achieve reasonable efficiency, the intermediate butterfly results need to be stored with larger number of bits than of input bits (because of multiplications and additions operations add bits). We decided to use 50 bits for each element inside the DFT computation. The number 50 also comes from the fact that target technology $Xilinx$ $Virtex-V$ has $DSP48E$ multipliers that can operate up to $25*18$ bits wide inputs. Thus the real and imaginary part of the samples in the FT mode are stored as 25 bits in the intermediate stages of the computation. The final results, as per global specifications, are saturated and / or truncated back to 16 bits with $Q1.15$ format.

Decimation In Frequency

Let X be a complex vector of size N. $DFT_N(Y)$ is given by:

$$DFT_N(Y)[k] = \sum_{n=0}^{N-1} X[n]\omega_N^{nk}$$

And $DFT_N(Y)[4k]$ can be written:

$$\begin{aligned}
DFT_N(Y)[4k] &= \sum_{n=0}^{N-1} X[n]\omega_N^{n4k} \\
&= \sum_{n=0}^{\frac{N}{4}-1} X[n]\omega_{\frac{N}{4}}^{nk} + X\left[n+\frac{N}{4}\right]\omega_{\frac{N}{4}}^{nk+\frac{N}{4}} + X\left[n+\frac{N}{2}\right]\omega_{\frac{N}{4}}^{nk+\frac{N}{2}} \\
&+ X\left[n+\frac{3N}{4}\right]\omega_{\frac{N}{4}}^{nk+\frac{3N}{4}} \\
&= \sum_{n=0}^{\frac{N}{4}-1} \left(X[n] + X\left[n+\frac{N}{4}\right] + X\left[n+\frac{N}{2}\right] + X\left[n+\frac{3N}{4}\right]\right)\omega_{\frac{N}{4}}^{nk} \\
&= DFT_{\frac{N}{4}}\left(U^0\right)[k], \text{ with} \\
U^0[n] &= X[n] + X\left[n+\frac{N}{4}\right] + X\left[n+\frac{N}{2}\right] + X\left[n+\frac{3N}{4}\right], \forall\ 0 \le n < \frac{N}{4}
\end{aligned}$$

Similarly,

$$\begin{aligned}
DFT_N(X)[4k+1] &= \sum_{n=0}^{N-1} X[n]\omega_N^{n(4k+1)} \\
&= \sum_{n=0}^{\frac{N}{4}-1} (X[n] - j.X\left[n+\frac{N}{4}\right] - X\left[n+\frac{N}{2}\right] + j.X\left[n+\frac{3N}{4}\right])\omega_N^n\omega_{\frac{N}{4}}^{nk} \\
&= DFT_{\frac{N}{4}}\left(U^1\right)[k], \text{ with:} \\
U^1[n] &= \left(X[n] - j.X\left[n+\frac{N}{4}\right] - X\left[n+\frac{N}{2}\right] + j.X\left[n+\frac{3N}{4}\right]\right)\omega_N^n \\
&= \left(X[n] - X\left[n+\frac{N}{2}\right] - j\left(X\left[n+\frac{N}{4}\right] - X\left[n+\frac{3N}{4}\right]\right)\right)\omega_N^n
\end{aligned}$$

$$\begin{aligned}
DFT_N(X)[4k+2] &= \sum_{n=0}^{N-1} X[n]\omega_N^{n(4k+2)} \\
&= \sum_{n=0}^{\frac{N}{4}-1} (X[n] - X\left[n+\frac{N}{4}\right] + X\left[n+\frac{N}{2}\right] - X\left[n+\frac{3N}{4}\right])\omega_N^{2n}\omega_{\frac{N}{4}}^{nk} \\
&= DFT_{\frac{N}{4}}\left(U^2\right)[k], \text{ with:} \\
U^2[n] &= \left(X[n] - X\left[n+\frac{N}{4}\right] + X\left[n+\frac{N}{2}\right] - X\left[n+\frac{3N}{4}\right]\right)\omega_N^{2n}
\end{aligned}$$

$$\begin{aligned}
DFT_N(X)[4k+3] &= \sum_{n=0}^{N-1} X[n]\omega_N^{n(4k+3)} \\
&= \sum_{n=0}^{\frac{N}{4}-1} (X[n] + j.X\left[n+\frac{N}{4}\right] - X\left[n+\frac{N}{2}\right] - j.X\left[n+\frac{3N}{4}\right])\omega_N^{3n}\omega_{\frac{N}{4}}^{nk} \\
&= DFT_{\frac{N}{4}}\left(U^3\right)[k], \text{ with:} \\
U^3[n] &= \left(X[n] + j.X\left[n+\frac{N}{4}\right] - X\left[n+\frac{N}{2}\right] - j.X\left[n+\frac{3N}{4}\right]\right)\omega_N^{3n} \\
&= \left(X[n] - X\left[n+\frac{N}{2}\right] + j\left(X\left[n+\frac{N}{4}\right] - X\left[n+\frac{3N}{4}\right]\right)\right)\omega_N^{3n}
\end{aligned}$$

Thus using decimation in frequency mode for radix-4 algorithm, the input samples to the butterfly are at a distance of $\frac{N}{4}$ for input vector size of N. The butterfly requires 3 instead of 4 twiddle factors. The three twiddle factors are at starting index n, and then at twice and thrice of the starting index i.e. $2n, 3n$.
As each element of input-vector is represented as $32-bit$ with $16-MSBs$ representing real part and $16-LSBs$ representing the imaginary part both in $Q1.15$ format. Radix-4 FFT algorithm implementation is based on butterfly operations, N/4 butterfly operations per stage, each of the butterfly operation requiring four complex additions and then a complex multiplication. Four additions cause an addition of 2 bits to input data resolution, so if input at any stage is represented by 'n.15' bits, the output after four additions would be '(n+2).15' bits. Then this output is multiplied with twiddle factor, represented in $Q1.15$ format, resulting a representation of '(n+2).30', no bit added on whole number portion of fractional number because multiplying by twiddles means rotation of input over unity circle. Least significant 15-bits can be ignored with an acceptable loss of accuracy, so a right truncation of 15 bits gives a result of '(n+2).15' bits as shown in figure 5.1.

To achieve the final result for DFT or IDFT, we also have to divide the whole summation by $\sqrt{N}$ at the end as shown in equations 5.1 and 5.2. We spread this division over all the stages of our algorithms. Each radix-4 stage is divided by 2, while the last stage of mixed-radix is divided by $\sqrt{2}$ to achieve the required results. Thus at the end of each stage, we reduce the data representation from '(n+2).15' to '(n+2).14' bits. This means that starting from $Q1.15$ at any stage s, we will have samples in $Q3.15$ at the end of a stage i.e. an increase of $1-bit$ per stage. The intermediate results stored

in the internal memory of the FEP are of $50 - bits$ each, $25 - bits$ each for real and imaginary part. Therefore, starting from $Q1.15$ format, an increase of $1 - bit$ per stage and with 6 maximum number of stages ($log_4 4096$), the intermediate butterfly results never go outside the range which is 25 bits. The results are sign extended at the end of each stage before storage in the memory.
At the end, to store back the results in the input-output memory (IO Memory), all the resultant samples are saturated back to $Q1.15$ format.

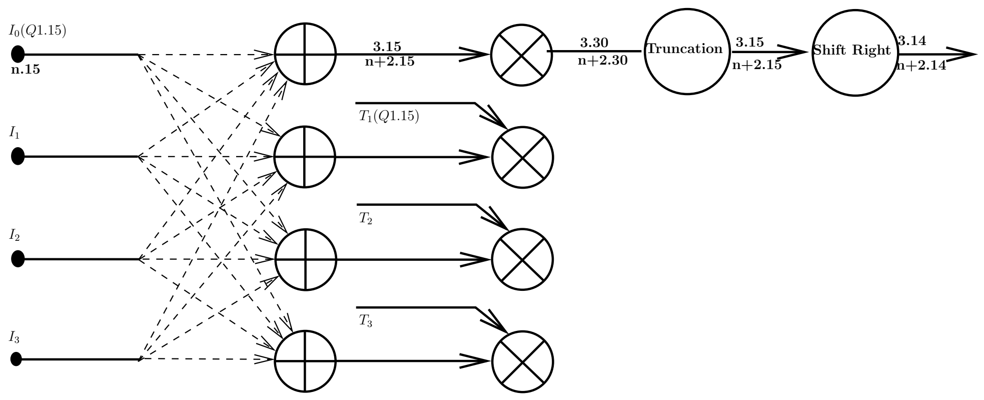

Figure 5.1: Basic Radix-4 Operation

5.2.4 Input Vector Size - Power of 2

For this case, our implementation uses the radix-2 algorithm with decimation in time domain. The details are described here.

Let X be a complex vector of size N. $DFT_N(Y)$ is given by:

$$DFT_N(Y)[k] = \sum_{n=0}^{N-1} X[n]\omega_N^{nk}$$

The summation above can be divided into two parts: even and odd. $n = 2n'$ for even half and $n = 2n' + 1$ for the odd half where $n' = 0, 1, \ldots, N/2 - 1$.

Now the DFT can be re-written as:

$$\begin{aligned} DFT_N(Y)[k] &= \sum_{n'}^{\frac{N}{2}-1} X[2n']\omega_N^{2n'k} + \sum_{n'}^{\frac{N}{2}-1} X[2n'+1]\omega_N^{(2n+1)'k} \\ &= \sum_{n'}^{\frac{N}{2}-1} X[2n']\omega_{\frac{N}{2}}^{n'k} + \omega_N^k \sum_{n'}^{\frac{N}{2}-1} X[2n'+1]\omega_{\frac{N}{2}}^{n'k} \\ &= DFT_{\frac{N}{2}} X[2n'] + \omega_N^k DFT_{\frac{N}{2}} X[2n'+1] \qquad (5.3) \end{aligned}$$

So the DFT can be computed by first computing the DFT over even and odd indices and then having a last stage similar to basic radix-2 computation. The computation of $DFT_{\frac{N}{2}}$ in this case becomes a computation of DFT over input vector size that is power of 4 as the size is divided by 2. Therefore, the radix-4 algorithm to compute $DFT_{\frac{N}{2}}X[2n']$ and $DFT_{\frac{N}{2}}X[2n'+1]$ can be used here as well. Then at the end, in the last stage, the multiplication with twiddles takes place.

The computation process for even and odd $DFTs$ computations is exactly the same as described above for power-of-4 case. The last stage also accesses 8 samples per cycle from the memory to meet the throughput requirements, 4 each from odd and even computation results. The samples are accessed in such a manner that no more than 3 twiddle memory accesses are required per cycle.
As per FEP functional specifications, when input vector size is power of 2, then $\sqrt{N}$ becomes $n\sqrt{2}$ where $n = \{2, 4, 8, 16, 32\}$. Thus we have to divide the result with a natural number 'n' and $\sqrt{2}$ as well. As explained in the previous case, a division by 2 takes place every cycle. For a division by $\sqrt{2}$, we multiply the results by $\frac{1}{\sqrt{2}}$ in the second last stage. Here, the second last stage is the last stage of radix-4 while computing the $DFT_{\frac{N}{2}}$. In the last stage of a radix-4 computation, no twiddle factor multiplication is required. To take advantage of this scenario, the multiplication with $\frac{1}{\sqrt{2}}$ is shifted in the previous stage. The division by 2 is switched to the last stage that sums up the DFT_{even} and DFT_{odd}.
In figure 5.2, the internal FEP Architecture in FT Mode is depicted, while figure 5.3 shows the DFT/IDFT processing unit along with its internal operations.
This completes the details for the FT mode design and implementation. In the next section, the other computation mode of the FEP block i.e. PP Mode is presented.

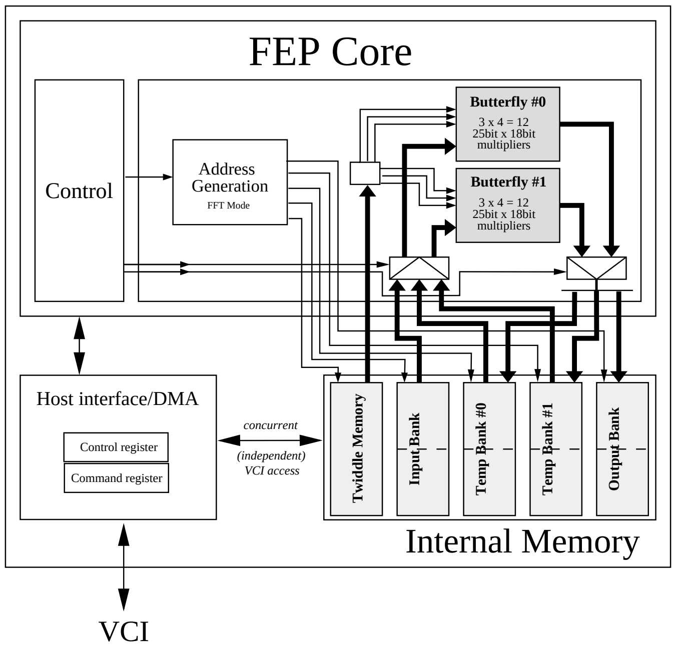

Figure 5.2: Internal FEP Architecture in FT Mode

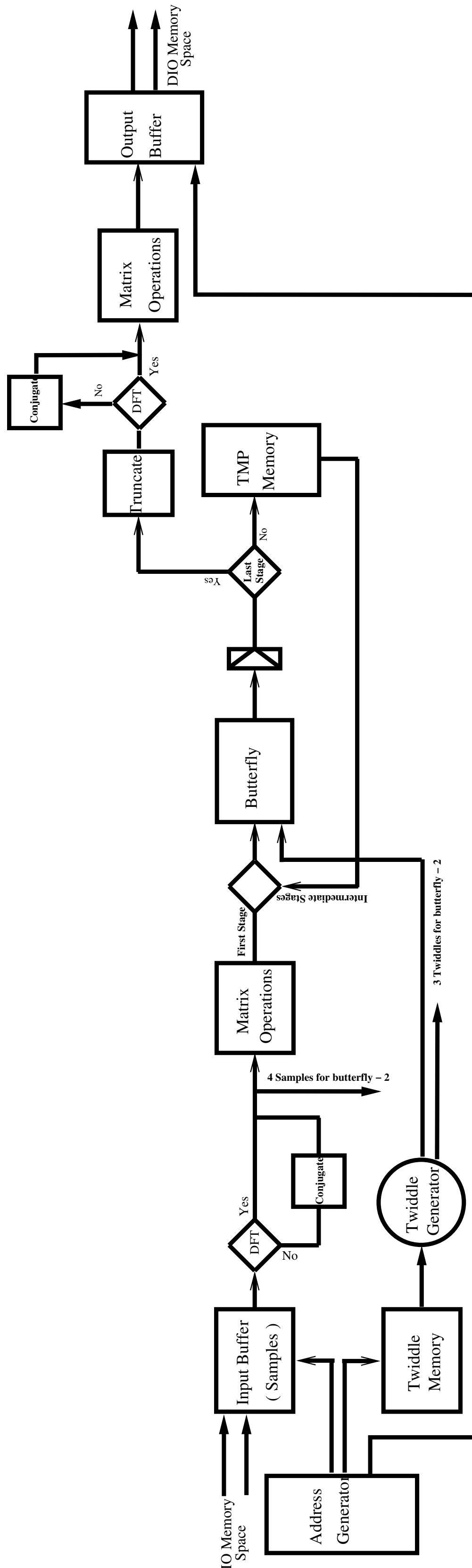

Figure 5.3: The DFT / IDFT Processing Unit

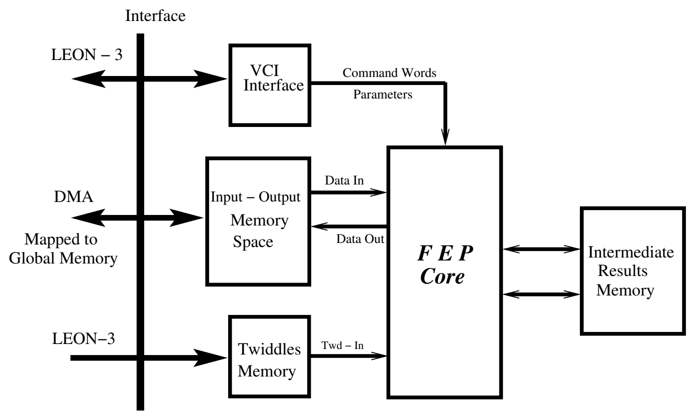

Figure 5.4: Communication among the FEP Blocks

5.3 Pre Post Processing

All the pre-post processing operations in the FEP can be performed over one whole vector and also over sub-band level. The sub-band concept can easily be understood as partitioning inside a large vector. For a given large vector U, there can be as many as 160 sub-bands or sub-vectors. The sub-band span indicates the total length over which the sub-band is formed. The size of sub-band can be less than or equal to its span. The size of sub-band corresponds to the number of samples / complex numbers to be used in the FEP computations. Inside each sub-band there are two important parameters; one is the offset while the other is sub-sampling factor. The offset means the distance between the start of sub-band span and the first sample / complex number to be used in the FEP computations. If the value is 0, then it means that we start from the very first sample i.e. start of the sub-band span. The sub-sampling factor gives the value of the skip between the two consecutive samples over the sub-band span. If the sub-sampling parameter is set to 1, then it means that consecutive samples are read out and used in the computations.

Figure 5.5 shows an example of the sub-band formations. Each filled circle represents a sample inside a large vector, with indices ranging from 0 to 29. Here two sub-bands $sb-0$ and $sb-1$ are shown, the sub-band span is 15. The red circles represent the samples over which the computation should take place in the sub-band mode of the FEP. The offset for each sub-band is 4, while the sub-sampling factor is 3. If the size of the sub-band is to be computed, it is not a part of the parameters passed to the FEP, then it would be 4 in this case. Also it is worth mentioning that FEP has a throughput of 2 samples per cycle in the pre-post processing mode; therefore for this example case the FEP will take 2 cycles to compute the PP-Mode computations. (apart from the set up and write back delays).

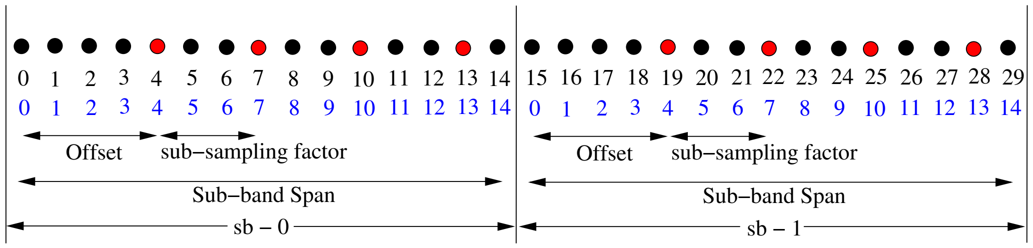

Figure 5.5: Formation of Sub-bands with parameters

In the pre-post processing mode, the next index inside each sub-band will be computed by:

$$\lambda = a + m * k \tag{5.4}$$

where a represents the offset at the start of the sub-band, m is the sub-sampling factor in the sub-band, and k is the natural index such that '$0 \leq k \leq span_{sb} - 1$' while $span_{sb}$ is the span of the sub-band. Also λ ranges 0 to $span_{sb} - 1$.
The next index inside the whole vector with multiple sub-bands can be computed as :

$$\begin{aligned} \gamma &= a + m * k + span_{sb} * j \\ &= \lambda + span_{sb} * j \end{aligned} \tag{5.5}$$

$a, m, span_{sb}$ represent offset, sub-sampling factor, and sub-band span respectively. k is the natural index such that '$0 \leq k \leq span_{sb} - 1$' and it is set to 0 at the start of each sub-band. j is also a natural index that indicates the sub-band-number in operation and is given by '$0 \leq j \leq n_{sb} - 1$' and n_{sb} is the total number of sub-bands in the vector. These notations will be used in the following sections.

The pre-post processing like DFT macro-block provides an option of conjugation of vector before and/or after the processing. Next a short description of these macro-functions is presented along with the list of parameters required.

5.3.1 Component-wise-addition

The complex vector $U+V$ is component-wise additive of U and V:

$$(U{+}V)[i] = (U[i].real{+}V[i].real,\quad U[i].imag{+}V[i].imag) \qquad \forall\, 0 \le i \le N{-}1 \tag{5.6}$$

In FEP module, if the sub-band parameters are set then all the component-wise operations follow the sub-band definitions. This helps to reduce the number of computations taking place for each of the operation because the offset and sub-sampling factor for the sub-bands allow to skip the samples inside the sub-bands (as depicted in figure 5.5). Component-wise-addition (CWA) over sub-band level is given by:

$$CWA_{sb}[i{+}span_{sb}{*}j] = U[i{+}span_{sb}{*}j]{+}V[i{+}span_{sb}{*}j], \qquad \forall i,\;\; i \in \lambda \tag{5.7}$$

where $\lambda = a + m * k$, and $a, m,$ and k represent the offset, sub-sampling factor and the natural index such that '$0 \le k \le span_{sb} - 1$' respectively. $span_{sb}$ is the sub-band span and j is the sub-band counter index i.e. the current sub-band index.

Each input element of a complex number in FEP module is represented in 32-bits, where real and imaginary both having 16-bits in $Q1.15$ format. An addition of two complex numbers causes an increase of 1-bit each to real and imaginary part. First the results are saturated, if required, and then one LSB is truncated to keep the result back in standard format $Q1.15$ for real and imaginary part (i.e. 32-bit complex number).

5.3.2 Component-wise-subtraction

The complex vector $U - V$ is component-wise subtraction of V from U:

$$(U{-}V)[i] = (U[i].real{-}V[i].real,\quad U[i].imag{-}V[i].imag) \quad \forall\, 0 \le i \le N{-}1 \tag{5.8}$$

Component-wise-subtraction (CWSub)can be calculated over sub-band level:

$$CWSub_{sb}[i + span_{sb} * j] = U[i + span_{sb} * j] - V[i + span_{sb} * j], \qquad \forall i,\;\; i \in \lambda \tag{5.9}$$

The notations and operations are the same as of component-wise-addition and the results are also written back in $Q1.15$ format.

5.3.3 Component-wise-product

The vector $U \odot V$ is the component-wise product of U and V:

$$(U \odot V)[i] = U[i] \times V[i], \quad \forall\; 0 \leq i \leq N-1 \tag{5.10}$$

This module also has the sub-band mode, where additional parameters offset, sub-sampling factor, number of sub-bands and span of sub-bands are utilized.

$$CWP_{sb}[i{+}span_{sb}{*}j] = U[i{+}span_{sb}{*}j]{\times}V[i{+}span_{sb}{*}j], \quad \forall i,\ \ i \in \lambda \tag{5.11}$$

The data representation of the output vector(s) is similar to that of input vectors. Each element of output is represented by 32 bits with equal length real and imaginary parts in $Q1.15$ format.

The multiplication of two complex numbers (say $X = a + jb$ and $Y = c + jd$), which in turn is four real multiplications and two additions, is given by:

$$X.Y = (ac - bd) + j(ad + bc) \tag{5.12}$$

Both real and imaginary parts of complex multiplication are composed of two real multiplication which are added to each other. As input vectors are represented in $Q1.15$ data format, each real multiplication gives result in $Q1.30$ format and the addition of two vectors in $Q1.30$ format results each real and imaginary result in $Q2.30$ format. This result is saturated, if required, back to $Q1.30$. The final result is 15-bit truncation of saturated result, and is stored as $Q1.15$ format. Thus the final result of this module follow the pattern of input similar to component-wise-addition and subtraction modules.

Component-wise-product with real vector

This module also offers component-wise multiplication with a real value vector i.e. the second vector contains only the real part and no imaginary part. This does not change the result format, and has no effect on the user interface. The internal truncation and saturation processes differ a bit, however the final results are stored in the same format.

Component-wise-product with scalar

The module offers component-wise multiplication with a scalar i.e. the second vector contains only one value and that is at the start address. This does not change the result format, and has no effect on the user interface. The internal truncation and saturation processes are the same, and the final results are stored in the same format as well.

5.3.4 Component-wise-division

This module performs a division of complex vector by a real vector. The exact operation performed is in fact multiplication using a look-up-table (LUT) which has a listing of inverse values of parameters to be divided. The width of each element in LUT is $8-bits$, though this strategy provides an approximate result but it is not that in-accurate to cause problems in the data detection of the FEP module.

$$\begin{aligned} U \div V &= U[i] * LUT(\text{truncated to 8-bits} \quad \hat{V}[i]) \\ &\approx U[i] \div V[i] \qquad \forall i \end{aligned} \tag{5.13}$$

$\hat{V}[i]$ represents the inverse value of $V[i]$ stored in LUT.

5.3.5 Dot Product

The complex number $U.V$ is the dot product of U and V:

$$U.V = \sum_{i=0}^{N-1} U[i] \times V[i] \tag{5.14}$$

The dot-product function D_{sb} provided by FEP is over sub-band level and is given by:

$$D_{sb} = \sum_{i \in \lambda} U[i + span_{sb} * j] \times V[i + span_{sb} * j] \tag{5.15}$$

where $span_{sb}$ is the span of sub-band, m is sub-sampling factor, and a is offset inside each sub-band. The λ is given by:
$\lambda = (a + m * k)$, with a, m and k representing offset, sub-sampling factor and index inside the sub-band span $(0 \leq k \leq span_{sb} - 1)$. The value of k is set to 0 at the start of each sub-band. j is the sub-band-counter index such

that $0 \leq j \leq n_{sb} - 1$

The resultant dot-product vector contains dot-product of each sub-band with each entity of $64 - bits$ with $32 - bits$ each for real and imaginary in $Q17.15$ format, with real part stored at address $addr_x$ and imaginary part stored at address $addr_{x+1}$. The result for successive sub-bands is stored one after the other in the memory.

$$D_{sb} = D_0 D_1 \ldots D_{n_{sb}-1} \tag{5.16}$$

The data format of dot-product can easily be explained with the help of component-wise-product data format. The dot-product is in fact summation of complex multiplication between the elements of two input vectors. As stated above, the resultant for each complex multiplication are stored back in $Q1.15$ format. All these results (2-results per cycle) are added up in a sub-band or vector. The summation of two complex samples of format $Q1.15$ results addition of '1' bit. The maximum size of vector elements can be 4096 therefore the addition of bits can be $log_2(max - size) = 12$. Thus in the largest input vector size, the output will be of 28 bits in $Q13.15$ format. The FEP sign extends the result for all cases to 32 bits for both real and imaginary part of the result and first write real part at address $addr_x$ and then writes imaginary part of the resultant complex number at address $addr_{x+1}$. The intermediate results are stored in two temporary registers of 32 bits each.

5.3.6 Energy Calculations

The energy of vector $\overline{U}$ with N elements is given by:

$$E(U) = \sum_{i=0}^{N-1} |U[i]|^2 \tag{5.17}$$

where $|U[i]|^2 = Re(U[i])^2 + Im(U[i])^2$ for $0 \leq i \leq N - 1$.
For sub-band energy calculations, the summations would be over sub-band size instead of whole vectors and is given by:

$$E_{sb}(U) = \sum_{i \in \lambda} |U[i + span_{sb} * j]|^2 \tag{5.18}$$

j is the sub-band-counter index such that $0 \leq j \leq n_{sb} - 1$.

Each of the resultant for energy calculations whether over whole vector or sub-band results in 32 bits in sign extended $Q17.15$ format. In sub-band energy calculations, the energy calculations result in a real vector of size n_{sb} (Number of sub-bands) elements with each element being $32-bits$.

$$E(U) = E_0(U)E_1(U)\dots E_{n_{sb}-1}(U) \tag{5.19}$$

The average energy over vector $\overline{U}$ with N elements is given by:

$$E(U) \approx \frac{1}{N}\sum_{i=0}^{N-1} |U[i]|^2 \tag{5.20}$$

The front-end-processor is capable to calculate average energy over sub-bands as well.

$$E_{sb}(U) \approx \frac{1}{size_{sb}}\sum_{i\in\lambda} |U[i+span_{sb}*j]|^2 \tag{5.21}$$

The data representation of the results is the same as for Energy calculation explained above.

The approximation '$\approx$' is used for the average energy calculations because the FEP uses the right shift operation over the approximate size of the vector or sub-band. The approximation used is given by:

$$E_{sb}(U) = \frac{1}{x}\sum_{i\in\lambda} |U[i+span_{sb}*j]|^2 \tag{5.22}$$

where $x = log_2\lfloor size_{sb}\rfloor$, and for whole vector $x = log_2\lfloor N\rfloor$.

Each element of input-vector for energy calculation is represented as $32-bit$ with $16-MSBs$ representing real part and $16-LSBs$ representing the imaginary part both in $Q1.15$ format. The calculation of $|U(i)|^2$ for each i requires two real multiplications and then an addition. Multiplication of two elements in $Q1.15$ format results in $Q1.30$ format, then a real addition of two resulting elements has a final result in $Q2.30$ format. The result is first saturated, if needed, to $Q1.30$ format and then the $15-LSBs$ are truncated to have results in $Q1.15$ format. This implies addition of $log_2(size_{sb})$ bits for a sub-band having size $size_{sb}$, as maximum sub-band size is 4096 so in the largest sub-band size scenario, a total of 12 bits would be added making the final summation result in $Q13.15$ format. However, the stored results in the memory are sign extended to 32 bits.

5.3.7 Maximum Calculations

The number max(U) of a vector U is the maximum of the energy level of individual elements of the vector and is given by:

$$\max(U) \quad = \quad \max_{0\leq i\leq N-1}(|U[i]|^2) \tag{5.23}$$

The natural argmax(U) is the smallest index i such that:

$$|(U[i]|^2 = max(U), \quad 0 \leq i \leq N-1 \tag{5.24}$$

For sub-band max-calculations, the vector traversal would be limited to sub-band-size (and only at the samples defined by sub-band definition) and module provides maximum value and arg-max for each sub-band. Similar to Energy-vector, Max-vector has n_{sb} elements with each element represented by 16 bits.

$$\max(U) = \max(U_0)\max(U_1)\ldots\max(U_{n_{sb}-1}) \tag{5.25}$$

$$\max(U_{sb}) = \max_{i\in\lambda}(|U[i + span_{sb} * j]|^2) \tag{5.26}$$

where $0 \leq j \leq n_{sb} - 1$ and n_{sb} is number of sub-bands.
The resultant for each sub-band or the whole vector is $16 - bit$ real value.

Similarly, argmax for sub-bands is a vector of naturals, length equal to number of sub-bands with each element represented in $16 - bits$.

$$\text{argmax}(U) = \text{argmax}(U_0)\,\text{argmax}(U_1)\ldots\text{argmax}(U_{n_{sb}-1}) \tag{5.27}$$

and each of argmax(U_{sb}) elements represents the smallest index i inside each sub-band. Given the maximum size of input vector 4096, the $argmax$ result is stored as $13 - bits$ right aligned.

The FEP carries out the energy calculations and the max calculations together, and the results are stored in the following fashion: For each sub-band or vector, the energy or the average energy result is stored at output address $addr_X$. It is 32 bit real number.
The argmax and max value inside the sub-band or vector are stored at output address $addr_{X+1}$. The max value is real number stored as $LSBs$, and the argument is number stored at $MSBs$. The results for the next sub-band start at address $addr_{X+2}$ and $addr_{X+3}$.

The FEP needs the following parameters to carry out the PP-Mode computations. These parameters are passed via the VCI-Interface.

- The operation code to be computed
- Number of sub-bands
- Span of each sub-band
- Offset inside the sub-bands
- Sub-sampling factor in the sub-bands
- Memory addresses (for input samples, output samples and twiddle factors)
- Conjugation flags for input vectors

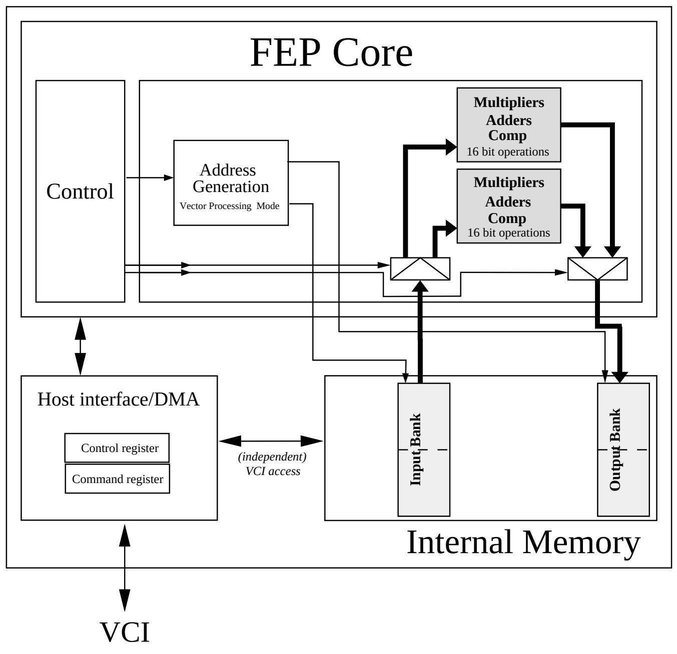

Figure 5.6: FEP Internal Architecture in PP Mode

Figure 5.6 explains the FEP architecture in the PP-Mode. Only 16 bit operations are carried out, and no internal memory bank is utilized in this case. The temporary results in the sub-band operations are stored in the local registers.

After describing the details of all the computation processes in the FEP block along with their specification, the next step is to look for the appropriate memory scheme that can effectively handle the diverse requirements. We explain the memory system of the FEP block in the next section.

5.4 Memory Subsystem (MSS)

The internal memory is used by the FEP to store the input data, intermediate and final results along with twiddle factors. From the external point of view it is a contiguous memory space accessible to all the peripherals. The memory is accessed by the VCI-Interface, DMA engine, micro-controller and by the FEP-core, hence it is a shared memory. The core is always given the highest priority followed by 6502, the DMA engine and the VCIInterface. It is the responsibility of the software to take care of data / result over-writings by any of the peripheries accessing the memory subsystem.

The FEP memory subsystem is composed of three main chunks, namely:

1. Input - Output data space
2. Internal data processing memory space
3. Twiddle factors memory space

The FEP memory sub-system is composed of ram blocks $RAMB36$, $RAMB18$ (Configurable Synchronous True Dual Port Block RAMs) available in $Xilinx\ VirtexV - FPGA$ [xil] each of size $36Kbits$ and $18Kbits$ respectively. For FEP utility, these are configured as a 36-bit wide by 1-K deep and 18-bit wide by 1-K deep true dual port RAMs.

The size of each of three blocks of the FEP memory subsystem is based on input-output data storage, intermediate results storage, and the number of memory-access and processing per cycle (i.e. the over-all performance requirement of the processing block) for all computationally different operations.

Input-Output Memory Space

For the size of the input-output memory, we consider a sub set of command words to be executed by the FEP core as shown in figure 5.7. The figure lists the commands (at some point in time) that are to be loaded in the

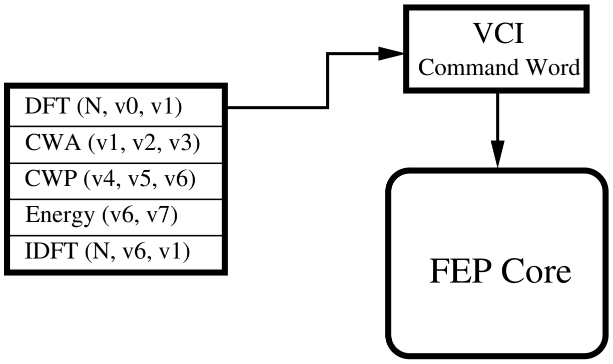

Figure 5.7: Successive Command Words for the FEP Core

VCI command word memory space and then the FEP core reads it out to compute the required operation. Once the FEP core has read out the command word from the VCI memory space, the VCI loads the next command word. The FEP executes one command at a time, and in the mean while the main control software makes sure that the next command starts its execution immediately after finishing the current command. This requires that the data should also be in the local memory subsystem so that the FEP can access it without any delay. Thus there are some constraints on the memory size with respect to the possible order of different command words of the FEP module. e.g. In the figure 5.7, the command words are listed with the vectors to be used in the computation along with the sizes in case of DFT operations. The command word $CWA(v1, v2, v3)$ represents the component-wise addition of input vectors $v1$ and $v2$ resulting in vector $v3$. The remaining parameters required to compute the operation are left out for ease of explanation here. The memory requirements for component-wise-operations (addition, subtraction, product and division) are more than any other operation in the FEP block, and hence enough memory is allocated to process two consecutive component-wise-operations over maximum input vector sizes. The memory allocation for the input-output space also considers that the consecutive operations' operands are mutually exclusive i.e. none of the operands or memory space is re-used in the consecutive instruction. This ensures that all the successive tasks for IP-core can be processed without any delay / lag between them, no matter what is the order of the command words.

Maximum input vector size = 4096 samples
Memory requirement for one Input vector = 4096 * 32 bits = 128 K bits
Memory requirement for input = 2 * 128 K = 256 K bits (CWP has two input vectors)
Memory requirement for output vector = 128 K bits
Memory requirement for one CWP operation = 256 + 128 = 384 K bits
Therefore, Total memory requirement for two CWP operations = 2 * 384 = 764 K bits

Some other results like energy, maximum and dot-products over sub-bands may also be required to be stored over multiple operations duration. Keeping this in view, we assign a total of $1 - Mbits$ of memory for input-out data space. Also almost one half of this memory is accessible by micro-controller.

Intermediate Results Memory Space

The size for Internal memory space is based on the intermediate computation during different operations like DFT, Dot-product, Energy / Max calculations etc. We take the most computation intensive task i.e. DFT for maximum input vector size as the basis to decide the size of internal memory. As per algorithmic implementation of our DFT scheme, the intermediate samples (results) are stored as $50 - bits$ instead of $32 - bits$ (as is the case with input and output samples). The DFT implementation is a pipeline based to meet the throughput and delay requirements; thus few intermediate results are being read while some others are being written in the same clock cycle. With efficient in-place memory access algorithms, memory equal to the maximum samples size can be allocated for the intermediate results. However, to avoid the high performance cost of the in-place memory access algorithms and also having the luxury of large memory space availability in the target technology, we reserve twice the maximum intermediate samples size memory for internal results / processing in the first version of the FEP. This can be reduced to half when we move to the system on chip solution of our block.

The target technology $Xilinx\ VirtexV - FPGA$ provides $RAMB36$ and $RAMB18$ which are $1 - K$ deep, while $32 - bits$ and $16 - bits$ wide respectively. As our intermediate results are $50 - bits$ wide, we make use of 2-parity-bits of $RAMB18$ to achieve a total efficient memory subsystem size. Hence the intermediate data representation used is on $48 - bits$.

Maximum vector size for DFT = 4096
Memory requirement for one vector = 4096 * 48 bits = 192 K-bits
Total Internal memory requirement = 2 * 192 = 384 K bits

Twiddle Factors Memory Space

The implementation scheme of DFT requires an access of 3 twiddle factors in each cycle (out of total 512 twiddle factors), which are stored (duplicated) in three different physical rams $RAMB36$ with one port access to IP-core for twiddle factors while the other port and one-half of the memory space for future usage and/or storing some vectors during debugging of IP-core. Twiddle factor memory is written by VCI-Interface and FEP core can only read it. One by eight of maximum size DFT i.e. 512 twiddle factors are stored inside FEP memory. These twiddle factors are written every time FEP is initialized. The twiddle factors are stored either in the upper half or the lower half of all three $RAMB36$ blocks, and this information is passed in the command word for DFT operations to facilitate the internal address calculations. Memory requirement for Twiddle factors = 3 * 1024 *32 = 96 K bits

Thus total size of memory sub-system = 1024 + 384 + 96 = 1504 K bits. The memory address space with byte-by-byte memory access and the structural overview of the FEP memory subsystem are shown in figures 5.8 and 5.9 respectively.

FEP Memory Size Dilemma

The total size of the FEP memory subsystem is huge, and is not obviously suitable for the system on chip development. There are two solutions that can be considered once we move to silicon:

- Reduce the memory size: It is possible for in-place execution of the results specially for the internal memory of the FEP. The current choice is made because of large memory size available in the FPGAs, and also because the in-place execution requires some complex memory algorithms. The memory size for the IO memory space can also be reduced by relaxing the strict conditions of the mutually exclusive

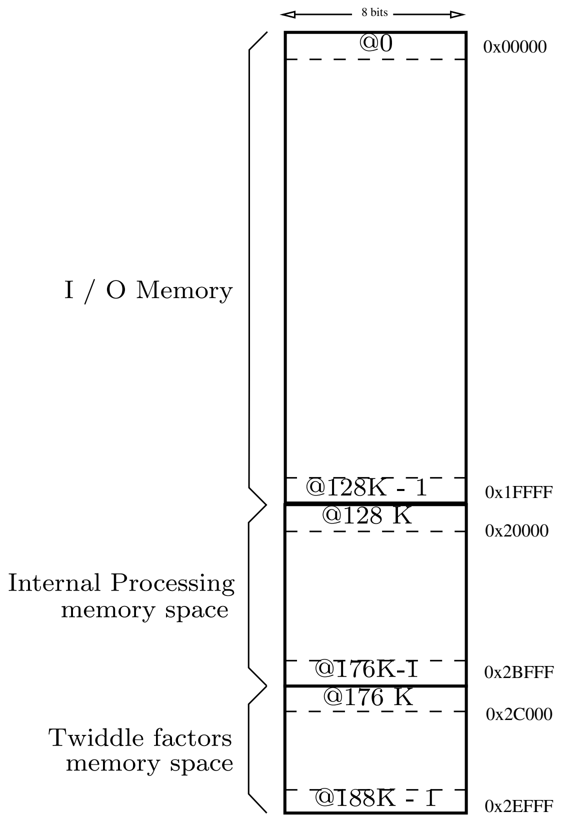

Figure 5.8: The FEP Memory Subsystem : Address Space

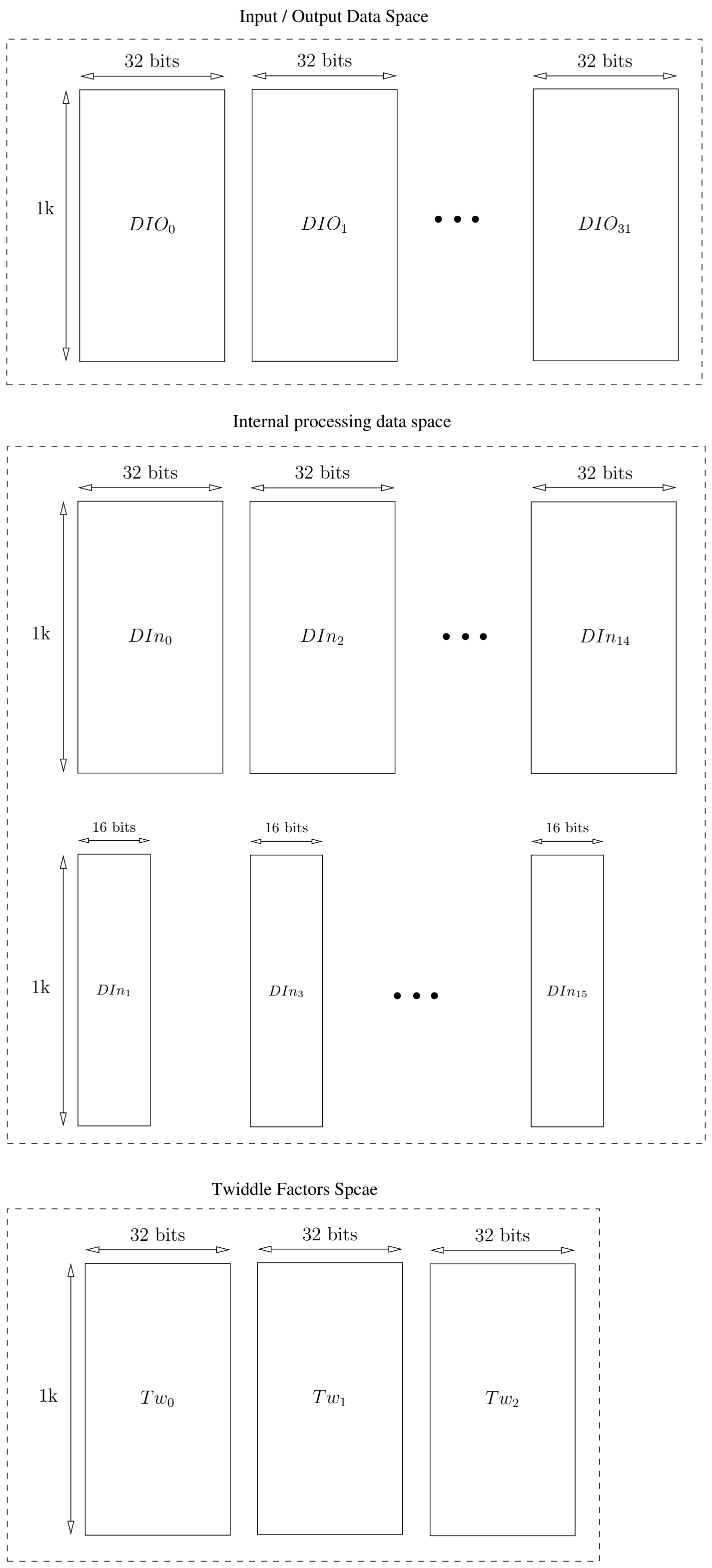

Figure 5.9: Structural view of the FEP memory subsystem

memory space for consecutive command words, as normally the consecutive commands are inter-dependent. The twiddle memory space can be reduced to half without any changes in any of the internal algorithms. The twiddle memory can also be reduced to $\frac{1}{6}$, by storing only one set of twiddles, and using more intelligent access algorithm.

- External Memory: The external memories like double data rate dynamic random access memory (DDR DRAM) can be added to serve the purpose, and the memory space can be shared by the other IPs in the design as well. In this case, the external memory access might become a bottleneck to the FEP high throughput requirements.

5.4.1 Memory Access Schema

To make the memory access scheme look more generic and simpler, we look into the memory access of the two computation modes:

1. FT Mode Memory Access
2. PP Mode Memory Access

In the FT mode, the global throughput of 1-sample per cycle led to the result that 8-samples should be read and processed in each cycle of all the stages in DFT calculations. There are 8 resultant samples to be written back in the memory as well. This means that two butterfly operations when radix-4 algorithm and four butterfly operations when radix-2 algorithm is in process. From the memory system point-of-view, this results in both reading and writing 8-samples each cycle through out the DFT computation.

In a hardware system design, the arrangement of any of the input /output samples should always be independent of the size of the input/output vectors. Considering the radix-4 algorithm, which is more often used: the samples accessed by a butterfly are at a distance of $N/4$ from each other i.e. the $i-th$ butterfly will access samples indexed $i, i+N/4, i+N/2, i+3N/4$.

The physical memory blocks at most can be used as dual ports, and hence the samples should be arranged in such a manner that no matter what is the size of the vector, the butterflies are capable to access the required samples without any latency from the memory. One simple solution can be to store samples in such a way that every $N/4$ chunk of the input samples is in different physical memory block, for each and every possible input vector size. This requires the samples to be stored in as many as 32 memory banks and then block multiplexing for each and every sample access. In case of Virtex-V FPGA of Xilinx, this requires almost 13% of logical functions to perform such an operation, and becomes quite expensive. Some intelligent operations at the input index of samples can reduce the number of banks to 8 and multiplexing at 16 ports, but this requires special arrangements for the smaller input vector sizes. In terms of logical functions, this doesn't improve much either.

In the PP-Mode, the throughput required is 2-samples/cycle. So an access of 2 samples per cycle from each of the input vector. There are 2 samples per cycle to be written back in the IO memory. Thus in case of PP-Mode, 6 memory accesses are required per cycle in the IO memory space.

The memory system in discussion is composed of ram blocks RAMB36, (Configurable Synchronous True Dual Port Block RAMs) available in Xilinx Virtex-V5 FPGA devices [12] each of size 36 k-bits. The blocks are configured as a 32-bit wide by 1-K deep true dual port RAMs. The total size of memory, which is far greater than the individual operations defined in this article, is based on the analysis of all the standards considered which require successive operations over vectors and storage of results for following operations without any latency. The memory is implemented in a matrix style of four rows times eight columns of 1k32 RAMs. In the following, columns are numbered from 0 for the leftmost to 7 for the rightmost and rows are numbered from 0 for the top to 3 for the bottom, and are shown in figure 2. The core operations access the memory through eight read-write channels, plus a mode indicator: either ft mode or pp mode as described earlier.

Access for FT Mode

When in FT mode the eight sample accesses are configured as eight read and eight write channels, one read-write pair per column. The addresses are 32 bit words addresses in the column, which is 12 bits (4k) only. The unused MSBs are discarded. The read addresses of the eight columns are considered as equal. Only the read address of the first channel is used, the others are ignored. The write addresses of the eight columns are considered as equal. Only the write address of the first channel is used, the others are ignored. The read and the write address are not necessarily equal. Address 0 points to the top most 32 bits word of the column, that is, the first 32 bits word of the top RAM of the column (the RAM in row number 0). Address 4095 (0xfff) points to the bottom 32 bits word of the column, the last 32 bits word of the bottom RAM of the column (the RAM in row number 3).

When computing a DFT, the IP core reads eight samples per cycle in the memory space in FT mode. These eight samples are eight consecutive components of the input vector. It uses a small internal cache (described below) to reorder the input samples before feeding them into its radix-4 / radix-2 units. This simple memory access puts a condition on the way the input vector has been stored in the memory space by a DMA transfer: the starting sample (32 bits) address of the vector in the memory space must be a multiple of 8.

We propose to use a set of simple 4x4 caches, shown in figure 5.10. The size of each cache element is the same as of input / output vector's element i.e. 32-bits each, the cache can be looked as 16 registers of 32-bit each. Now instead of using any fancy addressing schemes, the input samples are written in consecutive order in the memory space. The memory is arranged in such a manner that the consecutive samples are written in 8 different banks at the same index. With the help of address generation scheme, the samples accessed are first written in the two caches instead of directly feeding to the butterfly. The samples are written row-wise i.e. in cycle-0 all the 8 samples accessed are written in row-0 of cache-1 and cache-2. In the next cycle, the samples are written in row-1 and so on. Once all the four rows are filled, then the reading process from the butterfly operation starts. At the same time, the access process continues, but it now writes the data in column-wise. The address generation indices are such that the samples at a space of N/4 to each other are at column-0 in all the 3 rows. Once all the rows are written, then the butterfly operation reads column-wise and thus have the required samples. Now the samples are written in column-wise fashion, as the 0-th column is free to store the input samples. The row and column operations are reversed every 4-cycles, the number of rows and columns in the cache. Figure 5.10 [MKP09b] illustrates an example of using one of these caches starting from cycle-0 of DFT computation.

This simple scheme ensures the DFT operation to be smooth with a small latency of four cycles at the input phase. As per split-radix algorithm, the samples after the last stage are required to be bit-reversed before being fed to output. Once again the two 4x4 caches are utilized and avoid extra logic or resources.

Access for PP Mode

When in PP mode the eight read-write channels are configured as eight read or write channels, two per row. Each channel can be used either to read or to

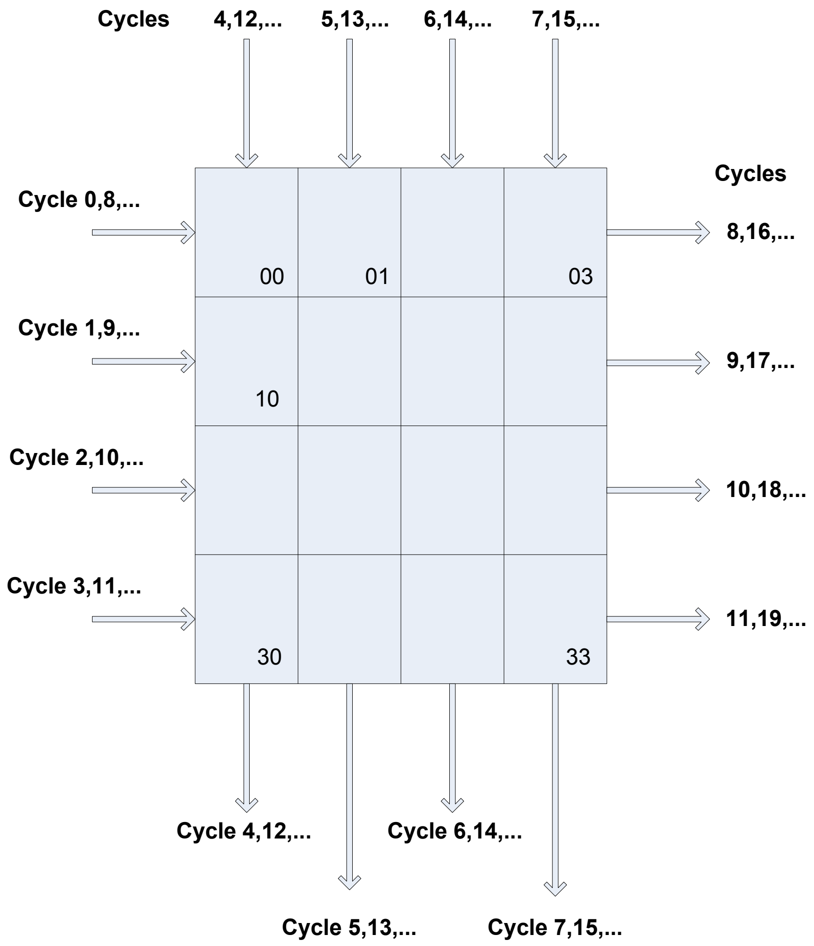

Figure 5.10: Cache at the Input / Output of FT operations

write but not both (exclusive, read-xor-write). The read or write addresses are 32 bits words addresses in the row that is 13 bits (8k). Address 0 points to the first 32 bits word of the leftmost RAM of the row (the RAM in column number 0), address 1 points to the first 32 bits word of the second RAM in the row (the RAM in column number 1), . . . , address 7 points to the first 32 bits word of the rightmost RAM in the row (the RAM in column number 7). Addresses 8184 to 8191 (0x1ff8 to 0x1fff) point to the last 32 bits words of the eight RAMs in the row.

Thanks to these two exclusive modes and to the fact that there are at most one read and one write channel per column (in FT mode) or two read-xor-write channels per row (in PP mode), there are at most two accesses per RAM in the DIO area. So, the two ports of the Xilinx block RAMs are sufficient for the IO requirements of all the operations in the block.

5.4.2 The Addressing Schema

The most computation intensive task of the FEP is DFT/IDFT, and the global throughput for the FEP block is set to be 1-sample/cycle. This requires the processing of 8 samples per cycle for DFT/IDFT operations i.e. both reading and writing of 8-samples in one clock cycle. DFT calculations are based on butterfly operations with each butterfly operating on four samples, so this implies that in our case two butterfly operations are to be performed in one clock cycle. The samples accessed by butterfly are input-vector-size dependent, which in FEP case ranges from 8 to 4096. This, in turn, means the arrangement of input / output samples for all the possible input vector sizes in such a manner that 8 samples can be operated in one clock cycle. As stated before ram blocks $RAMB36$ are used in memory sub-system of FEP, which are dual port RAM blocks and at most two samples can be accessed from each physical RAM block per clock cycle. The memory arrangement scheme is invisible to FEP users, and only FEP core has the exact knowledge of it.
The memory arrangement of samples puts the following conditions on the users of FEP memory block:

- FT-Mode The input and output addresses must be multiple of 32 bytes (byte addressing).
- PP-Mode The three interface addresses, two input and one out address, must be in different blocks. Also, all data of the individual vector must be in one block i.e. it must not expand / spread over two blocks. The pair of start and end-address of each individual vector

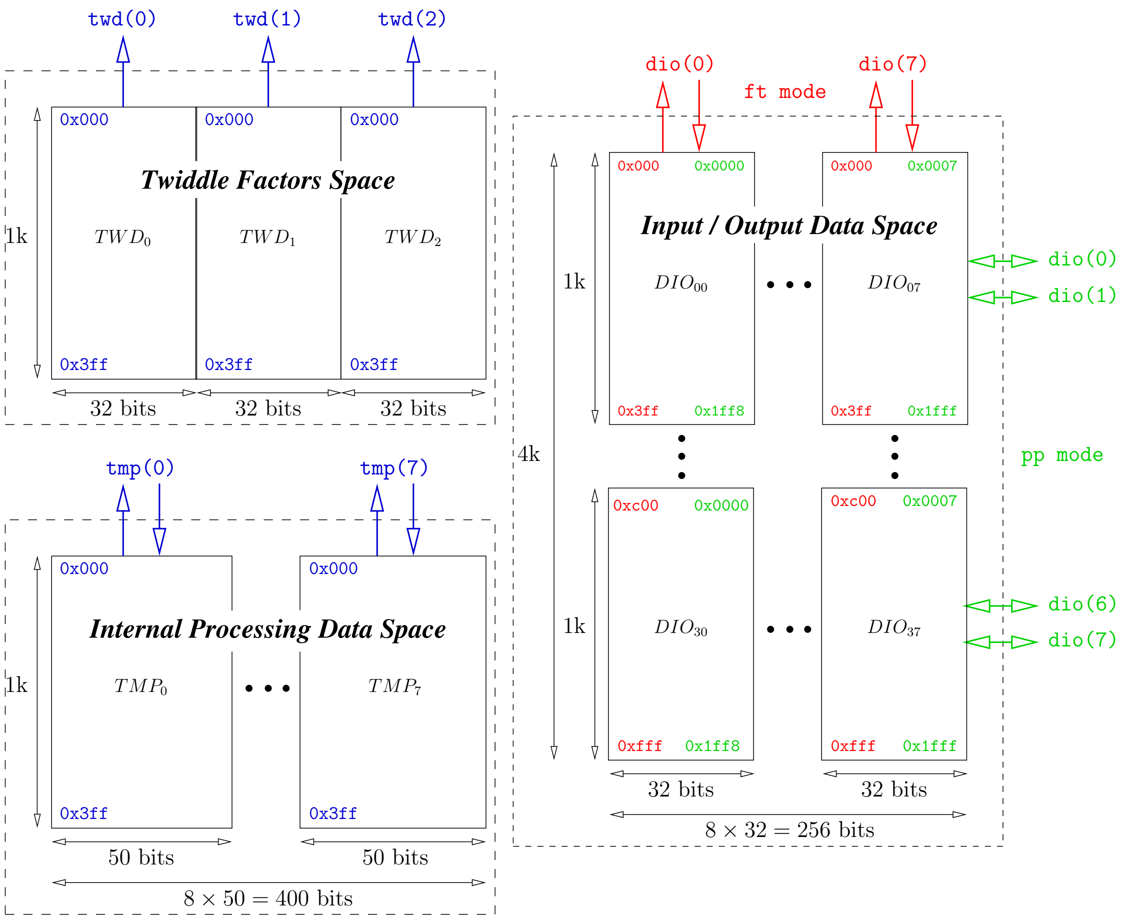

Figure 5.11: The memory layout from IP core point of view

must be in the same block, while the three pairs must be in three different blocks.

The addressing scheme analysis brings an end to the memory subsystem discussion, and we presented the memory access details for all the possible modes and operations inside the FEP. This also completes the design of the FEP block and the implementation is carried out using *VHDL*. The results are discussed in the next section.

5.5 Implementation Results

Using the Xilinx Virtex-5 FPGA as the target technology and Mentor Graphics' Precision as synthesis tool [men], $36 DSP48E$ slices are used. This makes 19% of the total available in the FPGA, which is quite good considering the fact that the FEP is one of the most computation intensive IP of the baseband processor. The maximum achievable frequency for DFT operations is 120 M Hz, and is quite acceptable considering the throughput of the block. The number of cycles spent to calculate the different input vector sizes of the DFT are shown in the table 5.1. The implementation of the DFT macro-block is pipelined to achieve the higher throughput and eventually higher data rates. The higher values of cycles used for the smaller pow-of-2 input sizes is due to the fact that in the Mixed-Radix algorithm the power-of-2 input size requires $log_4 \frac{X}{2} + 1$ stages for an input vector size of X. However for the larger values, the IP performs quite better than the anticipated throughput of 1-sample-per-cycle. The throughput for the component-wise-operation and sub-band-level-operations is 2-samples-per-cycle, and is achieved in the implementation accordingly. The maximum achieveable frequency for these modules is around $150 MHz$, and the number of complex multipliers used is 10 [MKP09a].

DFT Size	**# of Cycles**	**DFT Size**	**# of Cycles**
8	20	16	18
32	46	64	60
128	107	256	174
512	372	1024	695
2048	1597	4096	3136

Table 5.1: DFT : Number of cycles used for different Input Vector Sizes

5.5.1 Limitations of the Architecture

Though the FEP meets the functional specifications of its design and also achieves good performance; there are few limitations to its design and are listed here:

The matrix operation, small cache algorithm described earlier, puts a condition on the way the input / output vector is written / read in the DIO memory area by a DMA transfer or by a direct access through the VCIInterface: the starting sample (32 bits) address of the vector in the DIO area must be a multiple of 8. As a consequence the starting 64 bits word address used by the DMA engine or by the VCIInterface must be a multiple of 4. Symmetrically, when reading an output vector from the DIO area, the sample starting address will always be a multiple of 8. This doesn't require any specific requirement at the higher control level, but still the software has to take care of it while assigning the input and output addresses to the FEP core.

The FEP functions are mutually exclusive i.e. only one set of macro-blocks can run at the same time. e.g. In the channel estimation of OFDM systems, the component-wise product can only start once the DFT has terminated. However, this was foreseen while designing the IP and it doesn't cause any performance degradation in the overall baseband receiver design thanks to the MSS design and higher throughput of the IP. It is also worth mentioning that there is no lag or delay between the start of a task at the termination of the previous task. The new tasks (commands) can always be written in the IP and the memory space can be filled in for next task, while the current task is in progress.

Summary

This chapter provided the hardwired design for the FEP block in the base-band design. The design implements a wide range of operations including DFTs and maximum calculations over vectors. The design specification in line with the requirements of the evolving wireless standards are met. Although the FEP design is flexible in the sense that multiple standards requirements can be processed by parameterizing its command registers, still we feel that more flexible solutions may be studied for the hardware blocks in the SDR designs. A study to explore one such option is described in the following chapter.

Chapter 6

ASIP design for Vector Processor

The emerging digital communication systems and cellular networks require multi-mode operations with increased flexibility and high performance. The silicon area and power efficiency also stand high on the architecture design consideration list. These factors lead to multiple trade off situation of mapping the complex transceiver design tasks on the target software / hardware platform. The Application Specific Instruction Processors (ASIPs) are one of the choices for a complex flexible platform design.

For ASIP design, if high flexibility and customization for the instruction set architecture (ISA) is required then tools that focus on architecture level optimization may be used. These tools use Architecture Description Languages (ADL). There has been numerous research approaches proposed in this field, however the Language for Instruction Set Architecture (LISA) is the one that gained commercial acceptance. We applied this approach to design an ASIP core to evaluate its usefulness in baseband architectures for SDR applications.

6.1 ASIP Design Methodology

Application Specific Integrated Circuits (ASICs) have traditionally been used for the development of the embedded processors for variety of applications. The complexity and electrical design challenges are increase with the arrival of new technology generation. It becomes more and more difficult for the design tools to cope with these challenges, thus decreasing the design productivity. The expensive design tools and the high manufacturing costs for the ASICs are not helping either. The programmable solutions or the Application Specific Instruction Set Processors (ASIPs) have rapidly emerged as an alternate to the ASIC designs. The ASIPs are defined as a processor designed for a particular application or for a set of specific applications. An ASIP exploits special characteristics of application(s) to meet the desired performance, cost and power requirements. Thus the ASIPs are quite efficient when applied to a specific set of applications such as digital signal processing, control systems, avionics, cellular phones etc. [SIH$^+$91]. The programmability in the ASIPs provides the flexibility which is missing in ASIC design. The ASIP solutions decrease the manufacturing cost and time to market [KMN02].

The ASIP design is an attempt to look for a balance between two extremes : ASICs and general programmable processors. ASIPs offer the availability of custom sections for time critical tasks (e.g. a Multiply-Adder for a real time DSP), and offer flexibility through an instruction-set [LMP94]. As new standards / applications keep on arriving in wireless communications, the operations in the SDR baseband design are becoming more complex. These more complex procedures require more flexibility to accommodate design changes, errors and specification changes; which may happen at the later design stages. It is very hard to make many changes in the ASIC, once the design is in place. In such a situation, the ASIPs offer the required flexibility at lower cost [CKY$^+$99].

In the beginning, the ASIPs and the relevant software development tools have been designed manually [JBK01]. The manual processor design is long, tedious, error prone and requires highly skilled engineers. The processor architecture design can be viewed in four parts [HML02]:

- Architecture Exploration
- Architecture Implementation
- Software Application Design
- System Integration and Verification

Without any automation, it is really hard to have the expertise in all these procedures and hence a reasonable ASIP. The LISA Processor Design Platform (LPDP), using language for instruction set architecture (LISA), provides a complete processor design process that is based on target architecture description in LISA language [HKN$^+$01]. The LPDP provides the flexibility to model a processor from the most abstract level to the micro architecture level. The platform uses its HDL generator to implement the architecture, once the micro architecture is finalized. The synthesizable models can be generated both in VHDL and Verilog. Based on the success of the LISA, as an automated processor design, we decided to explore the ASIP design for the multi-standard baseband design

6.2 ASIP for Flexible Baseband Design

In the context of Open Air Interface architecture [ope], an analysis of the functionalities in the baseband design is carried out to identify a set of operations that are used in the transceiver baseband and are a good candidate for the ASIP design. It was noted that different blocks require basic arithmetic operations such as addition, subtraction, multiplication etc. over large size vectors. These operations are quite often used in different routines which are implemented in the hardware accelerators of our baseband design. For example in the digital front-end processor block, the energy calculations over large vectors is carried out using multiplication, addition and division over large complex vectors.

The dedicated routines composed of the vector operations in the baseband design are stand alone and serve only the purposes inside the specific block. To take the advantage of flexibility that ASIPs provide and also to explore the effectiveness of ASIPs for our long term dedicated platforms, we decided to design a Vector Processor as an ASIP. As a starting point, the vector processor is to be embedded in the digital front end processor of the baseband design. The FEP often uses the basic arithmetic operations over large vector sizes in its routines for channel estimation, data detection etc.; therefore we start the design with the aim that the proposed ASIP would at least cater all the needs of the FEP application routines. Later on, considering the performance and cost benefit analysis, the vector processor (VP) might become a stand alone IP on the baseband board of the Open Air Interface.

The current hard wired formation of the FEP is no doubt flexible and parametrize but still lacks the flexibility in some aspects. We try to figure out where flexibility or the programmability can be added in the hardware block. As stated above, the FEP functions are composed of DFT unit and the basic algebraic functions like multiplication, division, addition and comparison etc. In future, addition of functions to meet the new standard requirements or changes in some specifications would be quite a tedious job. So an ASIP design that carries out the basic algebraic functions over complex vectors (with some control options) would be an appropriate candidate to study the flexibility and design cost performance analysis in the context of SDR applications.

With reference to the FEP block (as explained in chapter 5), there are two computation modes namely: FT and PP. The FT carries out the DFT and the IDFT for variable lengths; while the PP carries out operations over large complex vectors such as energy calculations and vector products. The initial functional requirements of the VP are specified to carry out the PP-Mode operations. The existing DFT/IDFT module of the FEP will be used along with the ASIP to run the macros specified by the LEON3 processor. The ASIP formation is shown in Figure 6.1. As stated, the VP will be a part of the FEP in the baseband board. Therefore, it follows the generic IP shell design that is common to all the IPs in the board (details in chapter 4).

In the new formation with the VP, the highest level tasks of the FEP, for example Channel Estimation, are first translated into functions like component-wise-product, energy calculations, and FT etc. Next LEON3 decodes / translates these individual operations (products, energy calculations etc.) into pre-defined ASIP routines and these routines are loaded into the program memory (PM) of the VP. e.g. the dot-product operation of the ASIP is translated into a routine that requires complex multiplications and complex additions over vectors, the energy calculations require the absolute square operation over complex vectors and then division by a real value.

The ASIP formation not only fulfills the requirements of the current specifications of the FEP but can cater other operations in our global design e.g. linear interpolation. The addition of functions based on algebraic functions would just require writing the new libraries in the LEON3 which can be loaded in the ASIP.

In the next section, we list the design features of the ASIP.

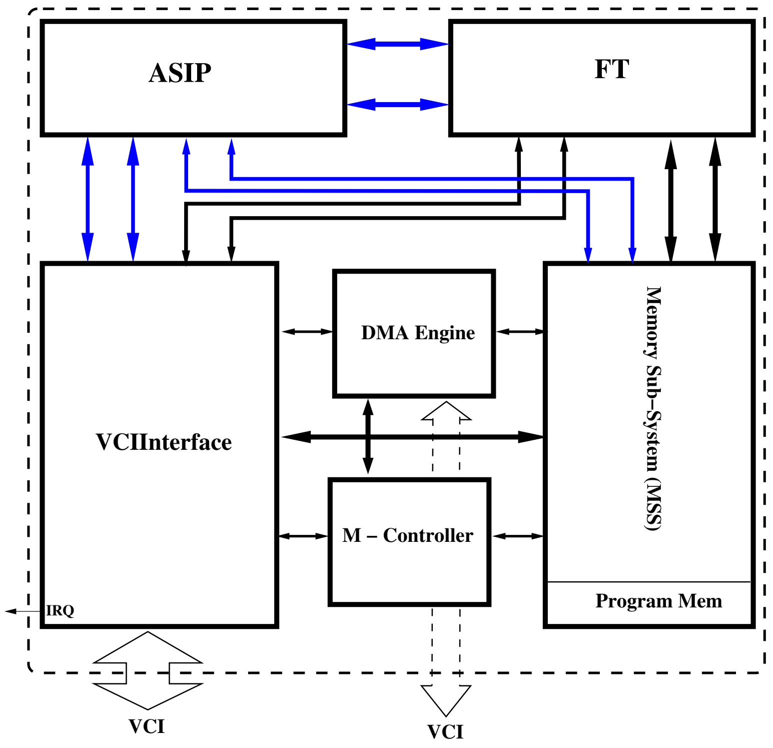

Figure 6.1: The ASIP Architecture with FT (of FEP module)

6.2.1 Vector Processor Design Features

The key design features of the vector processor are listed here:

- The FEP core currently is composed of a FT and a VP block, while the micro controller works as the control processor. In the new formation, VP is replaced by the ASIP while FT block of the current ASIC is utilized to carry out operations requiring time-frequency conversions.

- The ASIP' program memory takes care of its commands, however for the other tasks such as data transfers to and from the FEP, we think that the micro controller is necessary. Thus micro controller is also part of the new formation. This enables coherent design throughout in our global system (there are multiple IPs in the baseband receivers [Figure 4.1]). The micro controller presence also ensures a simpler ASIP without any overhead and only its own routines to take care of. In later versions, this design formation can be altered, if required.

- The program memory of ASIP is inside the MSS, and from time to time it is updated by LEON3. The instruction set for the ASIP is composed of libraries that can be downloaded from LEON3. These libraries can be loaded at the start-up time and also at the run time. Therefore, the program memory of the ASIP is part of MSS and is accessible by LEON3 (main processor) via the VCI and DMA.

- The memory access scheme by the ASIP must result a contention free access to program memory inside the MSS.

- As the existing FT module of the FEP will be accessing the MSS, the design and access of MSS will be the same in ASIP case as of the FEP design. The details of the memory subsystem are given in chapter 5.

- The external interfaces of the modules will be the same, however the ASIP can define its own interfaces / configuration registers as per requirement with other modules of the FEP.

- The MSS priority list: FT is given the highest priority, then the ASIP followed by micro controller, DMA, and VCI.

6.3 Functional Specifications of the Vector Processor

The VP is designed to meet the functional requirements of the FEP block except from DFT; which are explained in chapter 5 in detail. The VP has additional options giving more flexibility and programability. The key design features are:

- The throughput is 2 samples per cycle for all the vector operations i.e. 4 input samples are read from memory in case of an operation requiring 2 input vectors like vector product, and 2 input samples are read for operations requiring 1 input vector like vector absolute square. Depending on the operation either 1 or 2 output samples are written back in the memory each cycle. However there are operations like finding maximum over vector length where the results are written back only at the end of vector or sub-vector.

- The vector length ranges from 1 to 8192. The vector can be composed of any number of sub-vectors of same size and organization.

- Organization of the sub-vectors or vector means that the offset and the skip remain same for all of them. Offset is the distance between the start of a (sub)vector and the first sample to be read. The skip represents the distance between two consecutive samples to be read by the vector operation.
- The output vector can be written in the memory at consecutive addresses or can follow the input vector(s) pattern with offset a, and skip m.
- The input vector(s) are complex with 32 - bits for each sample in $Q1.15$ data format. One of the input vectors can have only real values or have a constant value located at the start address.
- Since VP is to be accommodated in the FEP, therefore the memory sub-system of the FEP is utilized. The address generation unit (AGU) follows the constraints of the FEP memory subsystem which are explained in chapter 5 as well.
- The input vectors can be conjugated before any of the operations inside the VP.
- The format of the output vectors, i.e. the number of truncation bits for results, can be defined by the programmer.

Here is a list of the operations or functions that the ASIP should carry out:

- Additions, Subtraction and Multiplication between two complex vectors
- Multiplication between a complex vector and a scalar
- Multiplication between a complex vector and a real vector
- Division between a complex vector and a real vector
- Absolute square of each element of a complex vector
- Summation of all the elements in a vector
- Maximum or Minimum of a real vector with respective index

Once the functional specifications are in place, the next step in the processor design is to identify the instructions required to carry out the functionality. We present the instruction set of our design in the next section.

6.4 The Instruction Set

The instruction set of an ASIP includes control instructions and processing instructions. For the proposed VP, the arithmetic operations are over large vectors and there are normally more than one vectors with the same formation. The parameters of the formation i.e. number of sub-vectors, size of sub-vectors, skip and the offset are also of large size in terms of number of bits required as explained in the previous section. It would be quite expensive to pass all the parameters with an instruction over a sub-vector and then pass on the parameters again, as in normal mode these parameters are to be repeated for many times. Therefore, we decided to have configuration instructions that are passed before the processing / arithmetic instructions and are used to configure the address generation unit of the ASIP. These instructions pass the required parameters for the following arithmetic instruction over large vectors or sub-vectors. On the other hand, the throughput for the ASIP block is 2-samples per cycle, thus 2-samples are accessed from each of the input vector per cycle. Therefore the arithmetic instructions will continue for multiple cycles. To illustrate with an example: A vector-product of input vectors $v1$ and $v2$ with 64 sub-vectors, each of size 32, lasts for 1024 cycles with offset 0 and skip 1.

The instruction set of the VP can be viewed in three parts; control instructions, configuration instructions, and the arithmetic operations instructions. The control instructions are NOP no-operation or stall, IRQ the interrupt request, and JMP jump to a specific address. The input vector sizes, number of sub-vectors, offsets and skips inside the vectors are all configured during the configuration instructions. The configuration instructions also set the starting addresses for the input and output vectors in the memory. The arithmetic operation instructions list the desired operation along with the different modes required such as conjugation, truncation etc.

To keep the number of instructions small and also to handle the large addresses inside the instructions, the width of the instruction set is 32-bits. Since the ASIP handles vectors of large size and also handles multiple sub-vectors of similar formation, therefore we have multiple configuration instructions. Once configured, the ASIP can operate over the vectors for multiple cycles. If there are more than one operations with same vector configuration, then the configuration instructions are not repeated. Single instruction over multiple cycles also needs to be handled in the program memory access mechanism and is described later in the concerning subsection.

Control Instructions:

- **NOP** (No operation)
- **IRQ** (The interrupt request)
- **Jump-Program** (Jump to a specific address inside the program memory)

Configuration Instructions:

- **agu_cfg_vector** (Configure the basic AGU processing parameters like vector size, number of de-interleaved vectors,...)
- **agu_set_vec0_addr** (Start address of input vector 0)
- **agu_set_vec1_addr** (Start address of input vector 1)
- **agu_set_res_addr** (Start address of the result)
- **agu_cfg_sub_vec_a** (Configure the number of sub-vectors and their size)
- **agu_cfg_sub_vec_b** (Configure the offset and skip of the sub-vectors)
- **agu_set_lut_addr** (Start address of the Look Up Table (LUT))

Arithmetic Instructions:

- **vec_mult** (complex vector multiplication)
- **vec_add** (complex vector addition)
- **vec_sub** (complex vector subtraction)
- **vec_div** (complex vector division with a real vector)
- **vec_mult_r** (complex vector multiplication with a real vector)
- **vec_abs_square** (absolute square of a complex vector)
- **vec_sum** (sum over complex vector or real vector)

- **vec_shift** (vector shift)
- **vec_square** (vector square)
- **vec_max_min** (complex vector maximum or minmum along with arg-maximum or arg-minimum)

6.4.1 Op-code

Configuration Commands

There are six configuration commands, which are used to prepare the data samples for the following *ALU* operations. These commands include: configuration of vectors, base addresses for input and output vectors and sub-vector parameters. agu_cfg_vector is used to set the values of vector size and the number of de-interleaved vectors. This option means writing back the output samples in sub-vectors spaced at a distance of $\frac{N}{d_n}$, where N is the input vector size while d_n represents the number of de-interleaved output vectors which can be $\{1, 2, 4, 8\}$.

The configuration commands for the addresses initialize the base addresses for input-0, input-1 and output vectors respectively. The AGU utilizes this information to calculate the addresses for each and every cycle of the processing.

The sub-vector configurations are split over two command words of 32-bit each. The first one updates the number of sub-vectors and the size of each sub-vector. The second command provides the offset value between start address and the first element inside each sub-vector, and the value of skip between every two elements inside each sub-vector.

Arithmetic Commands

The arithmetic command words (ALU command words) configure the ALUs and activate the internal processing depending on the arithmetic instruction (vec_mult, vec_add, ...). Each command has the same underlying structure:

instruction conjugate truncation crf maxmin constant_info shift_info

The arithmetic instruction that has to be processed by the ASIP is given at the first place in the command (instruction) by its specific code.

NOP	0 0 0 0				
IRQ	0 0 0 1				
jump_prog_mem	0 0 1 0	address program memory, 28 Bits			
agu_cfg_vectors	0 0 1 1	vector size 16 Bits	d_n 4 Bits	op–p 1 Bit	unused 7 Bits
agu_set_vec0_addr	0 1 0 0	start address – 1, 28 Bits			
agu_set_vec1_addr	0 1 0 1	start address – 2, 28 Bits			
agu_set_res_addr	0 1 1 0	output address, 28 Bits			
agu_cfg_sub_vec_a	0 1 1 1	num–sub–vector 12 Bits	size sub–vector 15 Bits	1 Bit	
agu_cfg_sub_vec_b	1 0 0 0	offset 13 Bits	skip 13 Bits	2 Bits	
agu_set_lut_addr	1 0 0 1	address LUT, 28 Bits			

arithmetic		insn 4 Bits	Conjugate 2 Bits	Truncation Information 4 Bits	csf	maxmin	const_info 2 bit	shift_info 5 bit	unused 8 bit
vec_mult	1 0 1 0	0 0 0 0	c c	t3, ... ,t0	x	x	ci ci	x x x x x	
vec_add		0 0 0 1	c c	t3, ... ,t0	x	x	x x	x x x x x	
vec_sub		0 0 1 0	c c	t3, ... ,t0	x	x	x x	x x x x x	
vec_mult_r		0 0 1 1	c c	t3, ... ,t0	x	x	ci ci	x x x x x	
vec_div		0 1 0 0	c c	t3, ... ,t0	x	x	x x	x x x x x	
vec_abs_square		0 1 0 1	c c	t3, ... ,t0	x	x	x x	x x x x x	
vec_sum		0 1 1 0	c c	x x x x	crf	x	x x	x x x x x	
vec_shift		0 1 1 1	c c	x x x x	x	x	x x	m s s s s	
vec_square		1 0 0 0	c c	x x x x	x	x	x x	x x x x x	
vec_max_min		1 0 0 1	c c	x x x x	x	mm	x x	x x x x x	

cc –> complex conjugate bits

ci –> input vectors : vector or constant

crf –> complex / real vector flag

d_n –> number of deinterleaved vectors

m –> shift mode

mm –> maximum or minimum operation select bit

op–p –> Output Samples Pattern in Memory

= 0, Write Back at consecutive addresses

= 1, Write Back as per Input Samples Pattern

s –> shift bits

t3 ... t0 –> truncation bits

x – > Don’t care

Figure 6.2: Instruction Set of the ASIP

The second parameter is '**conjugate**', represented by two flags that indicate either one or both or none of the two input vectors needs conjugation before operation.
Furthermore, the ASIP provides the flexibility of variable truncation over the resultant. This is quite helpful, specially when input vector elements have too small values to be significant for operations like multiplications. The '**truncate**' parameter allows a flexible data truncation of the final results to $Q1.X$ bit, where X is in the range between $1 \ldots 15$.
The 1-bit '**crf**', Complex/Real flag for the operation *vec_sum*, gives the information about the input vector. If this bit is set to '1', it means that the input vector is composed of complex numbers. Thus real and imaginary part are added separately. If **crf** is set to '0', the input vector is composed of real values, each with a size of 32 bits. Instead of 16 bit values, 32 bit values are added in this mode.
The '**mm**' parameter is used for the *vec_max_min* operation to distinguish between the maximum and the minimum determination.
In the case of a multiplication instruction, different options are possible. Either complex vectors with complex vectors (*vec_mult*) or complex vectors with real vectors (*vec_mult_r*) are multiplied. Both these modes offer multiplication with constant as well, and the parameters are set by **constant_info** flags.
The parameter '**s**' contains two different information and is used for the *vec_shift* operation. The MSB stands for the processing mode. If it is '1', the input vector are shift to the left; if it is zero, a right shift operation is performed. The remaining 4 bits contain the information about the number of bits to shift. Table 6.1 illustrates the different possible parameter settings and shows in which arithmetic operations they are used.

Next we look at the pipeline design, and list the different pipeline stages in our design along with their functionality.

6.5 Pipeline Structure

For the architecture itself, we started with a 5 stage RISC like pipeline. Later on, we split the execute stage into two stages to achieve high frequency and efficiency. Figure 6.3 shows the layout of the pipeline. It is composed of six different pipeline stages:

- **Pre-Fetch (PFE)**:
  In Pre-Fetch, the Program Counter (PC) is incremented to retrieve

Parameter	**Values**	**Description**	**Arithmetic Operations**
conjugate (c c)	11 10 01 00	both vectors need conjugation vector 1 to be conjugated vector 0 to be conjugated Nothing to be done	used in all
truncation	t3, t2, t1,t0 value x (1 . . . 15)	truncation to Q 1.x format	vec_mult, vec_mult_s vec_div_s vec_abs_square vec_add, vec_sub
crf	0 1	complex values real values	vec_sum
max-min (mm)	0 1	get the maximum get the minimum	vec_max_min
constant_info (ci ci)	11 10 01 00	not supported vector-1 is constant vector-0 is constant Neither is constant	vec_mult, vec_mult_s
shift_mode (m)	0 1	left shift right shift	vec_shift

Table 6.1: Instruction Set Modes

the next instruction from the program memory. If the configuration instructions are decoded, the PC is incremented by one each cycle. In case of an arithmetic operation, the PC keeps its value (no increment) and is decremented by two when the next instruction is loaded. More detailed information can be found in the subsection about Program Memory Accesses.

- **Fetch (FE)**:
  This stage is kept empty due to the 2 cycles delay of the program memory.
- **Decode (DC)**:
  In Decode, the instruction is loaded from the program memory and interpreted. Based on it, the top level processes are activated. If data has to be retrieved from the MSS (arithmetic operation), the corresponding read signals are set.
- **Execute 1 (EX1)**:
  The data retrieved from the MSS is available in Execute 1 and processed by the ALU. Furthermore, this stage contains several processes of the AGU, that store the configuration data in the Local Registers. If the LUT has to be accessed in case of a *vec_div* operation, the read signals are set and the data is retrieved two cycles later.
- **Execute 2 (EX2)**:
  The second part of the ALU is implemented in Execute 2. This stage is only activated if an arithmetic operation has to be processed.
- **Writeback (WB)**:
  Writeback contains the part of the AGU, that generates the write addresses. The addresses are forwarded to the writeback_result function, that is connected with the ports to the MSS. The number of vectors that has to be written back depends on the number of vectors read in and on the processed instruction. Like Execute 2, this stage is only activated if an arithmetic operation is processed.

The instruction set, op-code and pipeline design are complete, the only missing element in the design is the memory system. The memory subsystem design for VP design needs to consider the three elements and their access by the processor, namely: Program Memory, Data Memory and Look Up Table. The next section is dedicated to memory design and access schemes in the different formations.

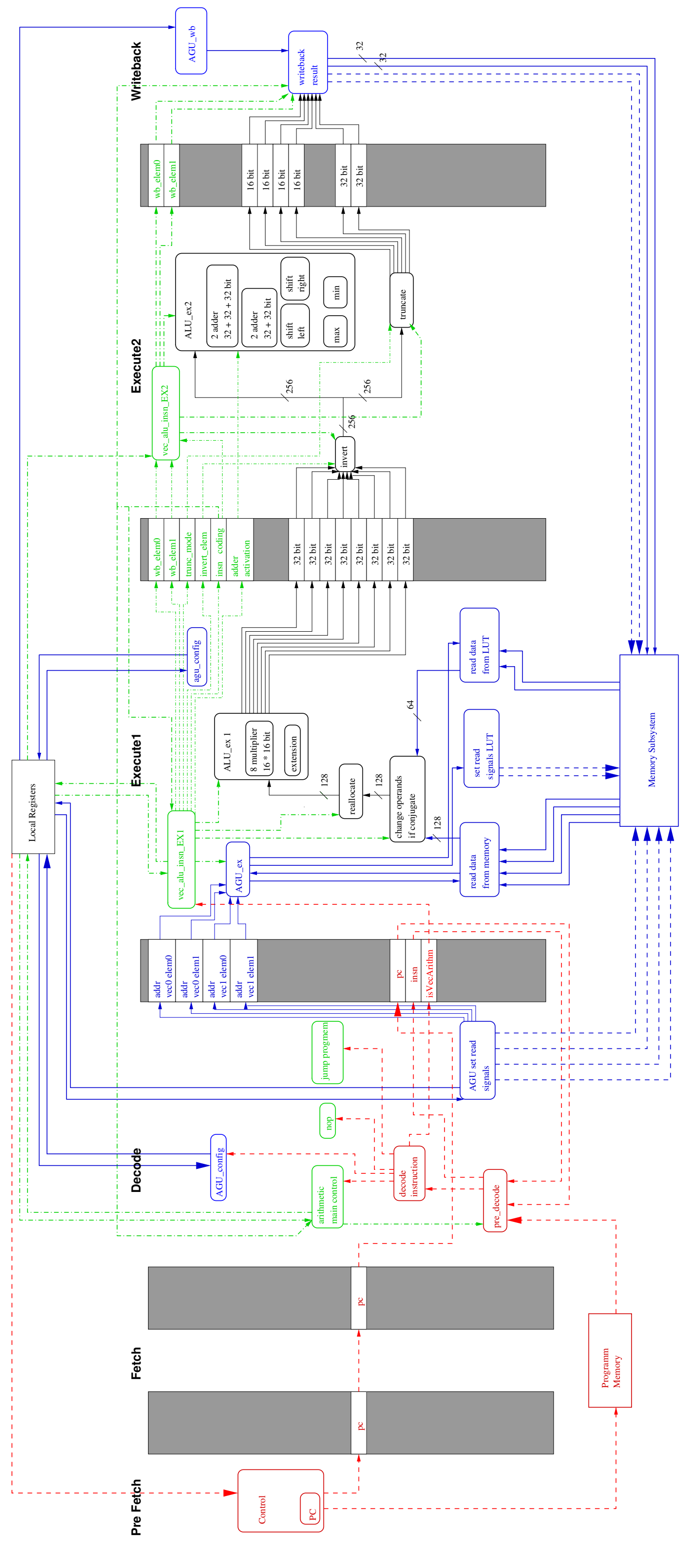

Figure 6.3: Pipeline Structure

6.6 Memory Access

The MSS consists of five different memory blocks: the program memory in which the program code is stored, and four data memories. When reading out of the blocks, the requested data is available after 2 cycles delay. The delay for the write operation is the same. The FEP only required data memories thus the MSS contained only the data memories. In the ASIP version, the program memory is part of the MSS, but designed as an independent memory block with its own ports.

6.6.1 Program Memory Access & Instruction Decoding

The program memory is part of the MSS and contains the program code that is executed by the ASIP. It takes two cycles after setting the address, till the instruction is available in the DC stage. The address is represented by the Program Counter (PC) in the PFE stage. There the function *pre_fetch* first checks if the pipeline registers between PFE & FE and between FE & DC are not stalled. If this is the case, an arithmetic vector operation is being processed, and the PC should not be incremented. Otherwise the PC is incremented by one each cycle. Before the arithmetic operation has finished processing, the PC has to be decremented by two to set it to the next instruction that has to be processed. This is based on the fact, that when the arithmetic operation is decoded, the PC has already passed the next two addresses to the program memory. Thanks to this architecture, no loop has to be part of the instruction set which simplifies the program code. In DC, the instruction is fetched from the program memory and directly processed, thanks to the output registered memory blocks in the MSS. The empty FE stage provides one cycle of delay for program memory access. The missing delay cycle is realized using pipeline registers. The function *pre_decode* in DC uses the PC value stored in the pipeline register between DC and EX1 to fetch the instruction from the program memory. The instruction is then passed to the *decode* function from where the root operation of the decoded operation is activated.

Figure 6.4 shows the the program memory access procedure along with the pipeline registers.

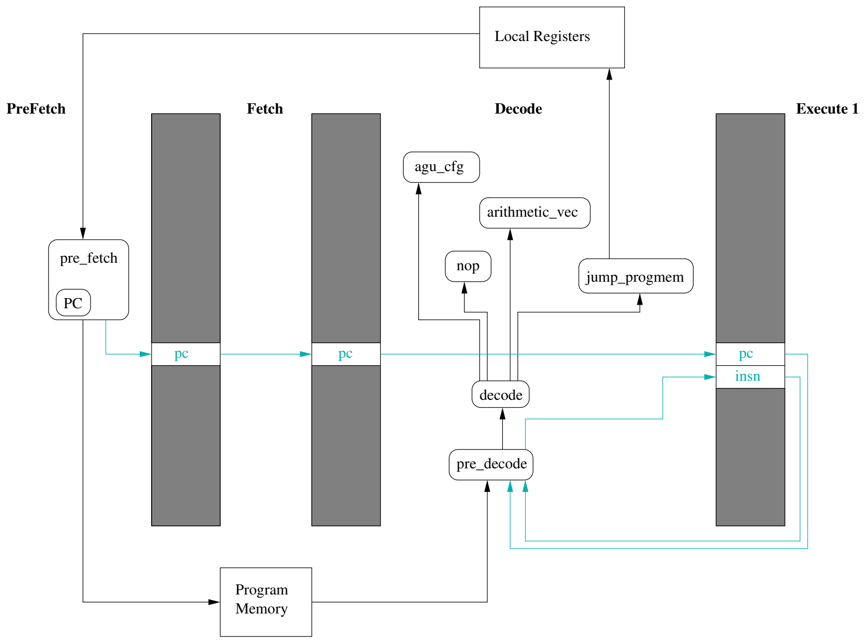

Figure 6.4: Program Memory Access

6.6.2 Data Memory Access

The Data Memory is accessed from EX1 and WB. Its addresses are generated by the Address Generation Unit (AGU) that is explained more detailed later. After passing the first read addresses to the MSS, EX1 is activated after two cycles when the first values are available. One address corresponds to reading of 32 bits from the memory. These 32 bits represent either a 32 bit real element, a 16 bit real (stored in the LSBs) or a complex element with the imaginary part stored in the 16 LSBs, and the real part at the 16 MSBs.
The addresses for the write back are generated in WB pipeline stage. If a component wise operation is processed, then the results are to be written back in each cycle and hence WB is always active. In case if sub-vector wise operation, the results are only written at the end of each sub-vector and hence WB is activated by EX2 when the end of a sub-vector is reached.

The data memory access process along with the pipeline functionality is depicted in figure 6.5.

6.6.3 LUT Design

The Look Up Table (LUT) is used in the *vec_div operation*, which uses multiplication with inverse instead of a real division operation. The LUT contains the possible division factors $\frac{1}{x}$. The input X can be any value in Q1.15 format and the system stores it at input vector location. In this scenario, the value of X is first read from the data memory, and in the next operation the read values becomes an address to be read from the LUT. Without any optimization the required table would have a size of 64-Kbits (2^{16} bits), assuming a 15 bit resolution of its entries (signed 16 bit entry). To decrease the required memory space, the table is optimized in the following ways:

- Only positive values are stored reducing the size to half. For the negative input values, the processor sets a flag when the data is read and uses the information in the address generation and multiplication.
- The bigger the value of x, the smaller is its inverse $\frac{1}{X}$. To obtain a smaller LUT without a significant loss of performance, the efficient schema is used and shown in table 6.2.

The final size of the LUT is 10.2-Kbits. The configuration command *agu_set_lut_addr* is used to define the start address (setting of *lut_start_address*),

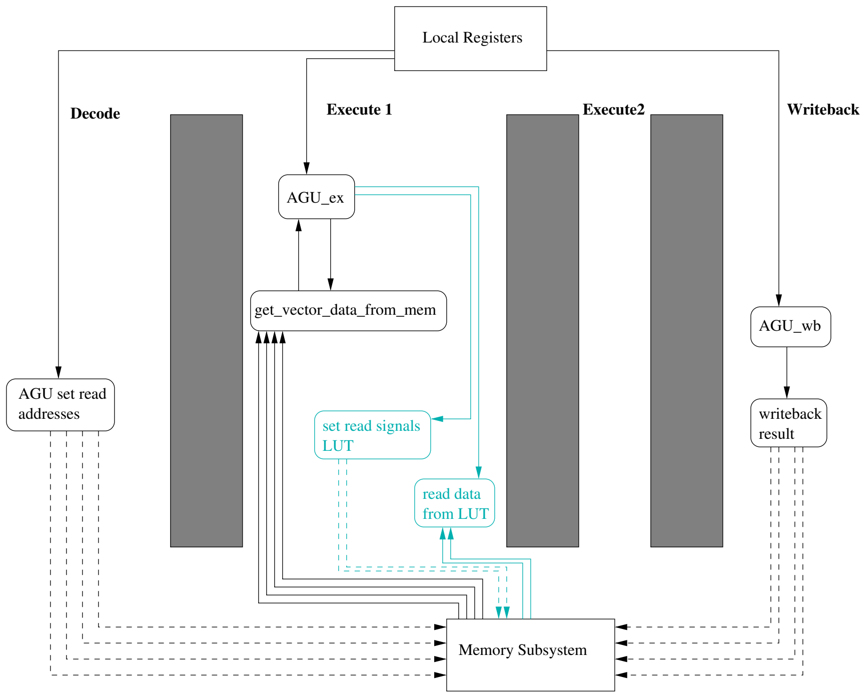

Figure 6.5: Data Memory Access

Start Address	Last Address	The Value / Address Scheme
0	$2^9 - 1$	All $\frac{1}{x}$ values
2^9	$2^{10} - 1$	Each 8th entry stored
2^{10}	$2^{11} - 1$	Each 16th entry stored
2^{11}	$2^{12} - 1$	Each 64th entry stored
2^{12}	$2^{13} - 1$	Each 512th entry stored
2^{13}	$2^{14} - 1$	Each 2048th value stored
2^{14}	$2^{15} - 1$	only one value stored

Table 6.2: Look Up Table (LUT) Schema

if the LUT is to be used in the operation.

In this section, we have listed the memory organization and access mechanism for different modes used in the VP. We give a brief description of the address generation unit in the following section.

6.7 Address Generation Unit

The Address Generation Unit (AGU) is split over two pipeline stages; the decode stage and the write back stage. The decode stage sub-unit generates the addresses for the elements of input vectors to be read while the sub-unit in the write back stage generates the addresses for the result elements to be written back in the memory.

The AGU has two modes: Sub-vector Operations mode and the De-Interleaved Output Mode. Based on the configuration commands, the AGU sub-units choose the implementation mode. Each and every command of the ASIP can have sub-vector operations, except from the case when the component wise operation results are to be written back in de-interleaved mode. The de-interleaved output mode is writing back the results spaced at a distance of $\frac{N}{d_n}$ instead of writing back contiguously; here N stands for the input vector size, while d_n represents the number of de-interleaved vectors.

In sub-vector operations mode, the output results are written back either contiguously or follow the input vectors pattern. The input pattern may also be contiguous or the samples are separated by the offset and the skip or sub-sampling factor. The offset represents the distance between the start of the each sub-vector and the first element to be read in the sub-vector. The sub-sampling or the skip factor is the distance between each element of the sub-vector. For contiguous samples in a vector, the offset value is 0, while skip is set to 1.

The AGU description completes the ASIP design procedure. In the next section, we give an overview of the verification process with the LISA tools before finally providing the implementation results.

6.8 Design Verification Process

The design verification is accomplished on two different levels: During the development phase with LISA, the Processor Debugger is used to simulate the described ASIP while on HDL level, a regression test is performed to verify the generated code. The latter includes LISA tool generation, LISA processor generation, LISA simulation, RTL simulation, verification and synthesis.

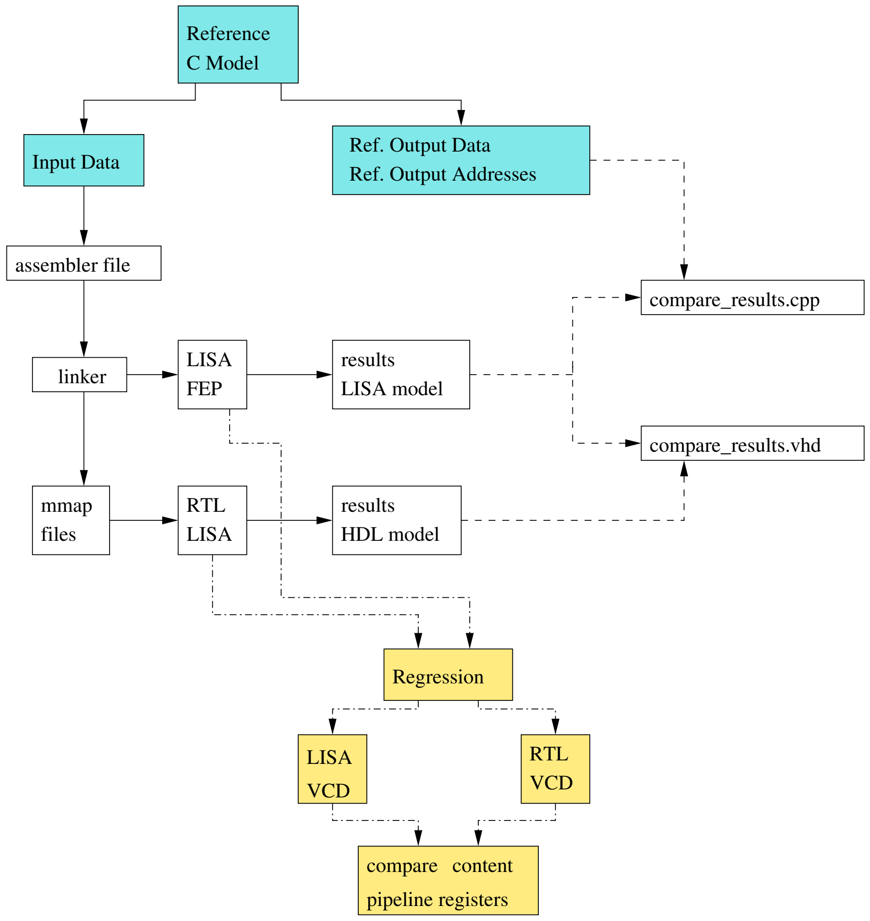

Figure 6.6: Verification Flow

The test environment of the ASIP is shown in Figure 6.6. On top, a reference C model is designed to create the input files for memory initialization, reference output sample files for the basic ASIP operations and routines, and reference read and write address files.

In case of simulations using the Processor Debugger, the simulation memories can be initialized using an assembler file that contains the data for each memory. The comparison of the generated output samples and addresses is performed using C++ functions, that compares the LISA output with the reference values. For LISA simulation procedure, the C++ functions are invoked. When performing the regression test, the memories are initialized using memory map files. After generating the HDL code, the test compares the content of the pipeline registers of the LISA and the HDL models and returns all the mismatches. Furthermore, the LISA simulation is processed again. To verify the output of the HDL simulation, the automatically generated simple test bench is modified to verify the various aspects of the design. Similar to the LISA simulation, the output samples and addresses are compared to their reference values.

The implementation results are described in the next section.

6.9 Implementation Results

The synthesis of the generated RTL code is carried out using Mentor Graphics - Precision RTL Synthesis 2009a [men], and the target technology of our baseband design i.e. Virtex5 LX330-1760 [xil]. The results are summarized in table 6.3.

Resources	VP - Usage
Global Buffers	1
Function Generators	7493 (3.61 %)
CLB Slices	1874 (3.61 %)
Dffs or Latches	1394 (0.66 %)
DSP48E	8 (4.17 %)
Maximum Frequency	78.6 MHz

Table 6.3: Summary of ASIP Implementation Results

The results apart from the maximum achievable frequency are quite encouraging. The ASIP design provides flexibility to the vector operations compared to the FEP (earlier hardwired design of baseband), and also that additional operations such as vector shift, vector minimum are supported by the VP. Given these two facts, the resource utilization is good enough to consider the ASIP designs for future IP development on the SDR platform. The number of DSP slices used are as per expectation, as there are 8 real multiplications taking place in the execute-1 pipeline stage of the VP. The ASIP development time is also less than the ASIC' design and development time.

Summary

This chapter gave the ASIP design approach using LISATeK, for the basic arithmetic operations over vectors with fancy addressing schemes. The effort was to evaluate the use of ASIP based designs for the SDR platforms, and assess the flexibility that they provide. The next chapter has a more detailed comparative analysis of both approaches and also discusses the hardware technologies for the flexible baseband SDR designs.

Chapter 7

Future Wireless Systems Design Approaches

In this chapter we present the design guidelines for baseband architecture of flexible multi-standard radio. We discuss the different hardware technologies available for the baseband design, list the advantages they offer and make comparison among those. We also discuss results of the two approaches, ASIC and ASIP design, that we used for the design of digital front-end processor as described in the previous chapters, are discussed. Next, We try to come up with some recommendations for future flexible wireless systems' baseband design in view of our results and analysis. This chapter also elaborates LTE channel estimation methodology, and lists the procedure that our proposed hardware design adopts to compute the channel response.

7.1 Hardware Design for Baseband Processing

The digital hardware component performance in the software defined radio (SDR) is a key aspect to measure the radio's capability. The set of algorithms to be implemented in the hardware comes from diverse wireless standards and waveforms thus providing a greater design challenge. On the other hand, a very high digital processing power is required to implement the flexible and efficient solutions for the SDR applications. The different hardware components that can be used to carry out these digital processing include:

- digital signal processors (DSPs)
- field programmable gate arrays (FPGAs)
- general-purpose processors (GPPs)
- application specific integrated circuits (ASICs)

Considering the performance requirements of the SDR and then analyzing the different options available to come up with the best solution is a challenging system design task and is discussed in this section.

GPPs are generally designed for high performance computing solutions performing mostly case-based reasoning operations or control algorithms. GPPs have not been used for real-time signal processing tasks because of their poor performance for the said tasks. In case of GPPs, its hard to predict the duration to execute a specific task at system design level because of caches, branch prediction units, and multi-tasking. One more constraint about the GPPs is their high power consumption per operation that would almost exclude them for SDR applications given the number of operations carried out in multi-standard platforms. However, the GPP-based systems can be better utilized in stationary systems where power consumption is of a bit less importance. Also the GPPs provide highest level of flexibility, and are capable of carrying out highly variant tasks over a wide range.

The digital signal processors (DSPs) are microprocessors that efficiently implement computational algorithms with high performance using specialized architectures thus reducing the number of computations for specific operation compared to GPPs. The DSPs are used for the applications that require higher performance execution than is typically found on standard microprocessors or GPPs; the DSP processor architectures provide optimized support for the high performance, repetitive, and numerically intensive mathematical manipulation of digital signals [LBSL97]. The DSP processors have several tailored dedicated operations, such as hardware multipliers, dedicated address generation units, and large accumulators. The DSPs also have special instructions available for common DSP operations with the ability to execute these in parallel, if required. On the other hand, the DSP processors become in-efficient for irregular control tasks because of unavailable specialized instructions and the flexibility is reduced as well. Both

GPPs and the DSPs have low performance for the complex bit based operations as is the case in the interleaving and channel decoding in a transceiver design.

The FPGAs are power hungry devices compared to their competitors (often several times more than ASIC), making it hard to choose them as the solution for baseband design. Though the FPGAs provide dynamic reconfigurability but its usage is limited in the context of SDR platforms. Also in almost all the scenarios, a software update would cost much less than a complex time consuming dynamic hardware reconfiguration in the multi-standard environment. The FPGAs also don't provide any gain in terms of speed or highest achievable frequency in the digital baseband design. However for rapid prototyping of an experimental platform such as [ope], FPGAs are ideal candidate based on reduced design cycle, flexibility, ease of use and lower costs of FPGAs. Once validated and a mass scale production is required, one has to switch to solutions such as ASICs or DSPs. So for on going analysis of baseband design, we take out the FPGA option.

ASICs offer the most optimized, powerful and computationally efficient digital hardware implementation for the signal processing applications at the cost of flexibility. Generally customized ASICs are used to provide added processing power when no other option is available due to design constraints or when designing sufficiently high volume systems. Also the design of sophisticated ASICs requires significant development time and effort in verification. Therefore, ASIC implementations tend to be better suited for highly complex problems or high volume applications or high data throughput requirements, such as cellular phones. On the other hand, following Moore's Law, manufacturers have been making almost 60% more transistors available per area of silicon each year providing large computational potential to design highly efficient ASICs with high data rates.

The hardwired circuits i.e. ASICs are most energy efficient, followed by the DSPs and then the General Purpose Processors. The flexibility moves in the opposite direction to energy efficiency. If software only solutions are adopted, then due to limited computational power of each individual unit there would be a large number of nodes in the design making the interconnect design quite complex. On the other hand, an all hardwired or ASIC based design would have specialized nodes in the design and the interconnection design would not be easier either. With the advantages and the drawbacks of the hardware technologies listed above, it is quite evident to realize that every design is unique and there is no universal solution of selection among the devices encompassing the entire range of baseband algorithms. For the latest wireless standards, the four main digital hardware categories (GPPs, DSPs, FPGAs, and ASICs) provide inadequate computational capability and require a combination of technologies for implementation. These new wireless standards require ASIC assistance for high throughput simple functions and DSP software for complex control functions [Pul08]. In fact, most designers use a combination of devices to implement the overall system, a method often referred to as heterogeneous processing. The trade-offs come in to picture while making the choice of most efficient device for a certain set of algorithms. The parameters such as cost, speed, and flexibility, as well as power and optimization, all have to be considered. Next we try to map the set of algorithms to different hardware technology options.

Once we have discussed the pros and cons of the candidate hardware technologies for baseband design, we move on to look at the different operations or algorithms that would be implemented in the SDR context. The baseband design of the software defined radio (SDR) is composition of different set of algorithms from various standards e.g. combining all the channel coding schemes from all the standards and trying to merge them in one unit that serves all standards. The basic design approach is to take the advantage of the commonalities that exist among different standards and then translate those into the hardware processing blocks. Once the set of operations and the algorithms for each set of tasks is defined e.g. channel decoder, interleaver or mapper; the next decision is to choose the hardware technology for the said unit. The more important parameters in choosing the hardware technology in the context of multi-standard baseband design are:

- Operation Regularity
- Processing Power Requirements
- Operation Homogeneity across standards

The ASIC design is most power efficient design methodology but at the same time requires the algorithms to be highly regular to take full design advantage. Since the power consumption is lowest, so ideally the largest chunk of the operations should be designed with the ASICs; given the available power resources for the baseband design. The operations that are highly regular, require huge processing power, and are highly homogeneous across the standards become a good candidate for ASIC design. The operations like Fourier Transforms (FTs), and digital filtering are ideal candidates for the ASIC design.

The choice of DSPs is good for the operations which are less regular though not irregular, and require dedicated specialized processing such as multiplications and MACs. The regular operations like FTs or the vector processors designed using the DSPs would also result in acceptable or moderate design. Thus for specialized operation (e.g specific data width) and less critical regular operations with medium processing requirements and energy consumption, the DSPs are good candidate in the baseband processing.

The GPPs are the most flexible and can serve hugely variant tasks. The irregular operations should also be carried out by the GPPs, however it is worth mentioning again that the power consumption is high in GPPs thus only limited operations should be assigned to the GPPs in a hybrid design. The control operations in the baseband design may typically be assigned to the GPPs.

So the assignment of a specific set of algorithms to a hardware technology depends on its computation nature, occurrence frequency and its behavior across multiple standards. Next comes the question, how often can we assign the algorithms to one category e.g. can all algorithms be carried out with the DSPs. It may be answered by looking at the number of operations per unit time (second) required in the final design and the energy resources available per second for the baseband part in the product. We illustrate this by an example and use the numbers given in [vB09].

The mobile terminals today have 3 watts of power available, out of which $1W$ is for digital baseband part of the handheld device. On the other hand, the digital workload is $100GOPS$ (10^9 operations per second). Thus we have a power budget of $10 - pico - J/operation$. Now if we have to design the baseband with hardwired i.e. ASICs and DSPs, then we would be using a combination of the two given that the power budget lies between the power consumption of these two hardware technologies. If the average power consumption of the hardwired is $2pJ/op$ and for DSP it is $20pJ/op$; then 55% of the operations can be assigned to ASICs while 45% may be assigned to DSPs. This example illustrates that not only the operation nature but the power budget must also be included as parameter while allocating a set of algorithms to a specific hardware technology.

7.2 ASIC and ASIP Design Comparison

We have presented two hardware designs in the previous chapters, the front end processor (FEP) in chapter 5 and the vector processor (VP) in chapter 6. The FEP design is a hardwired design i.e. an ASIC design, while the VP design is an ASIP one. In this section, we present the comparison of both of these designs on the basis of our observations and results.
The functional specifications of the ASIP were initially set to match that of the FEP except from the Fourier Transforms (FT) i.e. the ASIP is supposed to provide all the functions that the FEP had apart from the DFT / IDFT. So basically ASIP is designed for arithmetic component-wise-operations over the vectors such as addition, subtraction, multiplication and division. Then there are functions such as Energy Calculations, Dot-Products and Max Calculations over vectors or sub-vectors. The design methodology of the ASIP to serve all these operation was different from the FEP approach. The ASIP instruction set is composed of the simple operation over vectors and composed of two or three instructions to carry out a bit complex operations such as Energy Calculations or Dot-Products. To illustrate with an example, to compute the average energy of a vector, the VP would do it in three steps: calculate the absolute square of the individual elements of the vector, then computer the vector sum by adding all the elements calculated in the previous step and finally using the division by the vector size. For the performance of the ASIP with respect to the ASIC design we have the following observations:

- The VP design provides higher programmability using the macro level operations, as these operations can be used in different formations with many options.

- The throughput for the component-wise-operations is the same as that of the ASIC or FEP design.

- The throughput of VP is half for few FEP operations like Dot-Product and Energy Calculations as it decomposed these operations in to simpler vector operations that ASIP instruction set is composed of.

- These macro level basic vector operations in VP give high vector operation flexibility at the expense of lower throughput for few functions.

- The address generation unit (AGU) of the VP is more flexible without loss of performance. In the VP, the output vector addresses for the memory can either follow the input pattern or can be stored at contiguous addresses.

- The VP provides some additional basic arithmetic operations e.g. Vector Multiplication with constants, Vector Shift, Vector Square; thus enhancing the vector functionality of the baseband design.

- The memory utilization for both designs is the same apart from the addition of $64Kbits$ program memory for the ASIP design.

Next is to compare both these designs on the different target technologies. To make a fair comparison, the FEP processing block is re-synthesized without its FT unit and both the designs are synthesized with $Xilinx - Virtex - V$ and also with $65nm$ as the target technologies. The synthesis results using both the target technologies are shown in table 7.1. The parameter target frequency is the synthesis constraint passed to the synthesis tool, while the maximum frequency is what design achieves at the end of the synthesis. The silicon area in the table 7.1 is represented per function generator (FG). In the results table, The parameter F/S (maximum frequency / silicon area) in the table is a classical quality criteria: the larger the better. To achieve the optimal results, we used an aggressive synthesis approach. We first set a very high target frequency to observe the maximum achievable frequency by the target technology without any other constraints. Once the highest achievable frequency is known then a frequency with minimum silicon area close to the highest frequency observed is looked for as is shown by the results in table 7.1. For the initial synthesis with Virtex-V, the number of extreme DSP48E slices used by each of the hardware block were different. Therefore, we disabled the option to use the extreme DSPs to make a fair comparison and the results shown are without using any DSP slice.

From the comparison of the results, the hardwired ASIC design is better than the ASIP design in terms of the area and frequency. The silicon area increases by 19% in case of $65nm$ target technology, while the increase in case of FPGA synthesis is 70%. We could not find an exact reason for the huge increase in case of Virtex-V synthesis, an in depth analysis is required for this specific result. The decrease in maximum achievable frequency is almost 70% in both ASIP synthesis results compared to ASIC. The difference in the maximum achievable frequency was not anticipated and the initial investigation suggest that with some minor changes in the LISA design, an acceptable frequency can be achieved. The difference is not huge in case of the silicon area, ignoring the FPGA results, and ASIP can be chosen and

Target Technology	**Design**	**Target clock period (ps)**	**Target freq (MHz)**	**Silicon area μm^2/FG**	**Slack (ps)**	**Max Frequency (MHz)**	**100 * F / S**	**Normalized F/S**
ASIC 65 nm	**VP**	3300.00	303.03	107968	1585.00	204.71	1.90	51.27
		5400.00	185.19	97987	114.00	181.36	1.85	50.05
		5430.00	184.16	99497	0.00	184.16	1.85	50.05
	FEP	2000.00	500.00	91686	1095.00	323.10	3.52	95.29
		3095.00	323.10	93495	0.00	323.10	3.46	93.45
		3291.00	303.86	82166	0.00	303.86	3.70	100.00
		3350.00	298.51	82168	0.00	298.51	3.63	98.24
Virtex V 5VLX330	**VP**	3333.33	300.00	9737	9377.67	78.67	8.08	33.92
		13157.89	76.00	9665	282.91	77.67	8.04	33.74
	FEP	3333.33	300.00	5683	4054.65	135.36	23.82	100.00
		4000.00	250.00	5709	3387.98	135.36	23.71	99.54
		5000.00	200.00	5709	2387.98	135.36	23.71	99.54
		7692.31	130.00	5707	304.33	135.36	23.72	99.58

Table 7.1: The synthesis results summary for FEP and VP design

replaced in the baseband design for vector processing once the maximum achievable fequency is within acceptable range.

Once the ASIP design is reworked, we think that the performances of both the design options would be comparable in terms of silicon area and frequency. Though the results for the ASIC design would still be a bit better but the ASIP design provides a higher degree of flexibility and increased functionality; the parameters quite important in flexible radio design. One more aspect that we would like to investigate in future is the power consumption for both the designs.

The ASIP designs like the DSPs and other software solutions stand as an alternative to the baseband designs when the complexity increases. Our effort was to investigate the suitability of the ASIPs in the hybrid-platform design for the SDR applications. Our baseband design, [ope] described in chapter 3, is composed of the parameterizable ASICs. The first effort with the prototype is to investigate that *how far we can go with the hardwired design keeping enough flexibility and acceptable silicon.* The reasons for the hardwired are the same as explained in the previous section: high computational power and lowest power consumption. The algorithms such as channel encoder and channel decoder that are neither regular nor homogeneous across the standards have posed enough problems for the hardwired design. The ASIC solutions for such complex blocks in the baseband design lack flexibility and require high performance. The percentage of logic re-usability in the ASIC design of these blocks is not encouraging e.g. in LDPC and Viterbi decoding. The better solution also requires higher programmability and the alternate solutions such as GPPs and DSPs (discussed in the previous section) do not help in this scenario. The researchers have already considered the ASIPs as a solution for decoder solution in multi-standard environment with acceptable results [VW08] [KMF09] [BAS04]. The ASIPs may be a solution for complex set of algorithms such as flexible encoder and decoder and stand as a strong candidate for exploration of such set of algorithms.

7.3 LTE Channel Estimation Approaches

In this section, we elaborate how our designed digital front end processor (FEP) can be used for the channel estimation of the 3GPP Long Term Evolution (LTE). The schema is similar to what we have in the OFDM case, we analyze the interpolation schemes based on the hardware resource utilization with reference to our implementation as it can handle multiple interpolation schemes.

We first look at the downlink frame arrangement and later analyze how the different macro blocks in our design are used for channel estimation procedure. The different possible interpolation schemes are also discussed and relevant procedures both for the FEP and the VP are listed.

7.3.1 LTE Downlink Frame Arrangement

The $3GPP\ E-UTRA$ (Evolved Universal Radio Terrestrial Radio Access) aims to achieve a peak data rate $100Mbps$ for the downlink. The $OFDMA$ modulation scheme is used at downlink to match the higher performance requirements. Each radio frame consists of 10 sub-frames each of $1ms$ duration, while the sub-carrier spacing is $15KHz$. Twelve sub-carriers are joined together to form a resource block (RB). The set of allowed transmission bandwidth consists of $\{1.4, 3, 5, 10, 15, 20\}MHz$. The number of resource blocks for each of the transmission bandwidth are $\{6, 15, 25, 50, 75, 100\}$, and the number of respective sub-carriers are $\{72, 180, 300, 600, 900, 1200\}$.

Figure 7.1 shows the arrangement of reference pilot signals inside one subframe of $LTE - Downlink$ scheme with one antenna port and normal cyclic prefix (CP). Each subframe consists of 14-$OFDM$ symbols and each $OFDM$ symbol is composed of 6 to 100 resource blocks with each resource block composed of 12 sub-carriers. The channel estimation algorithm is first run over each $OFDM$ symbol in the subframe containing the reference pilot symbols. For the channel estimate of symbols in between without reference signals, the temporal interpolation is carried out on the symbols containing reference signals.

The channel estimation is computed using the Least-Square method. First of all, the received signal is converted into frequency domain by using the DFT module of the FEP. The component-wise-multiplication at the pilot positions between the received signal and the reference pilot signal gives the channel estimate and is given by:

$$\hat{H}_{i,p} = R_{i,p} \odot P_i^H \tag{7.1}$$

where $R_{i,p}$ is the received signal at the pilot positions for antenna i, while P_i^H represents the Hermitian of the reference pilot signals for antenna i. The component-wise multiplication with sub-band / sub-vector option of the FEP is used here, and the skip factor is set to the distance between the consecutive pilots for each antenna (which is 6). In case of the VP, the *vec_mult* is used by setting the appropriate sub-vector parameters.

The channel estimation for the sub-carriers in between the pilots can be computed by the interpolation scheme. The interpolation method depends on the scheme chosen, here we discuss three schemes and describe the methodology to be used by our hardware processors to implement each of these methods.

7.3.2 Linear Interpolation in Frequency Domain

In frequency domain linear interpolation scheme, the channel response in a symbol over the sub-carriers positions (other than pilot positions) is calculated by linear interpolation of the two pilots that are around these sub-carriers on each side. On the edges of the resource block, the extrapolation of the pilots in that particular resource block is carried out. In the following, we define the operation for single and multiple antennas.

Case: SISO with normal cyclic prefix

Figure 7.1 shows the arrangements for a single antenna case with normal cyclic prefix. The pilots are distributed over the symbols $\{0, 4, 7, 11\}$ in each subframe. Inside each resource block there are two pilots out of total 12 sub-carriers. Figure 7.2 explains the linear interpolation and extrapolation for a SISO case with 25 resource blocks. As the distance between the consecutive pilots is constant over the whole symbol length, therefore the fractional co-efficients for pilots are periodic over the distance between the pilots i.e. 6. The co-efficients listed are for the case, when the pilots are at positions $\{0, 6\}$ inside each resource block. The pilots position inside each resource block varies depending on the cell ID and it may have any other value like $\{1, 7\}$, however the distance between the consecutive pilots is always the same. At the edges of each symbol, for the sub-carriers that have pilot sub-carrier only on one side, the extrapolation process is carried out to compute the channel estimate for the sub-carriers at the edges. Since in this case (figure 7.2) the first pilot is at position 0, therefore the extrapolation only takes place at the other edge of the symbol and is depicted in the figure along with the multiplicative co-efficients. The fractional co-efficients for the interpolation are constant, while for extrapolation those are bit different for two different pilot locations.

The channel estimate for the symbols without reference pilot sub-carriers, (symbols $\{1, 2, 3, 5, 6, 8, 9, 10, 12, 13\}$ in this case), is determined by temporal linear interpolation between the symbols with reference pilot sub-carriers. The channel estimate for the symbols $\{1, 2, 3\}$ is given by:

$$\hat{H}_{0,i}(1) = \frac{3}{4}\hat{H}_{0,i}(0) + \frac{1}{4}\hat{H}_{0,i}(4) \tag{7.2}$$

$$\hat{H}_{0,i}(2) = \frac{1}{2}\hat{H}_{0,i}(0) + \frac{1}{2}\hat{H}_{0,i}(4) \tag{7.3}$$

$$\hat{H}_{0,i}(3) = \frac{1}{4}\hat{H}_{0,i}(0) + \frac{3}{4}\hat{H}_{0,i}(4) \tag{7.4}$$

This is calculated once we have $\widehat{H_{0,i}}(0)$ and $\widehat{H_{0,i}}(3)$ for all sub-carrier values i, antenna 0 and for symbols 0 and 3 respectively. The procedure is repeated for the other non-pilot symbols as well.

The procedure using the ASIP based vector processor for the above explained operation is listed here:

- Load the calculated channel estimate for each antenna $\hat{H}_i$ over the pilot positions, here only one antenna so $\hat{H}_0$.

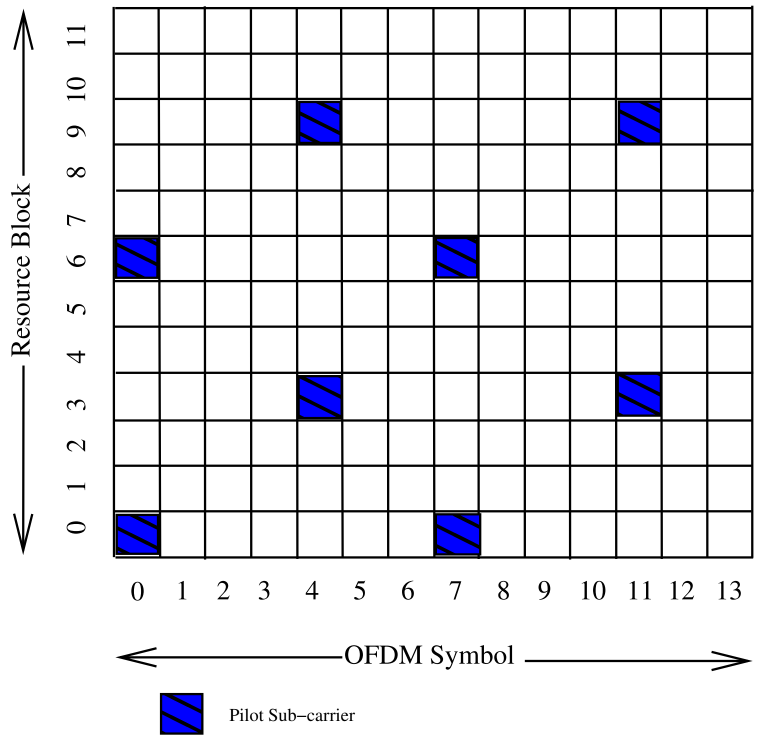

Figure 7.1: Mapping of LTE Downlink Reference Signals for SISO with normal prefix

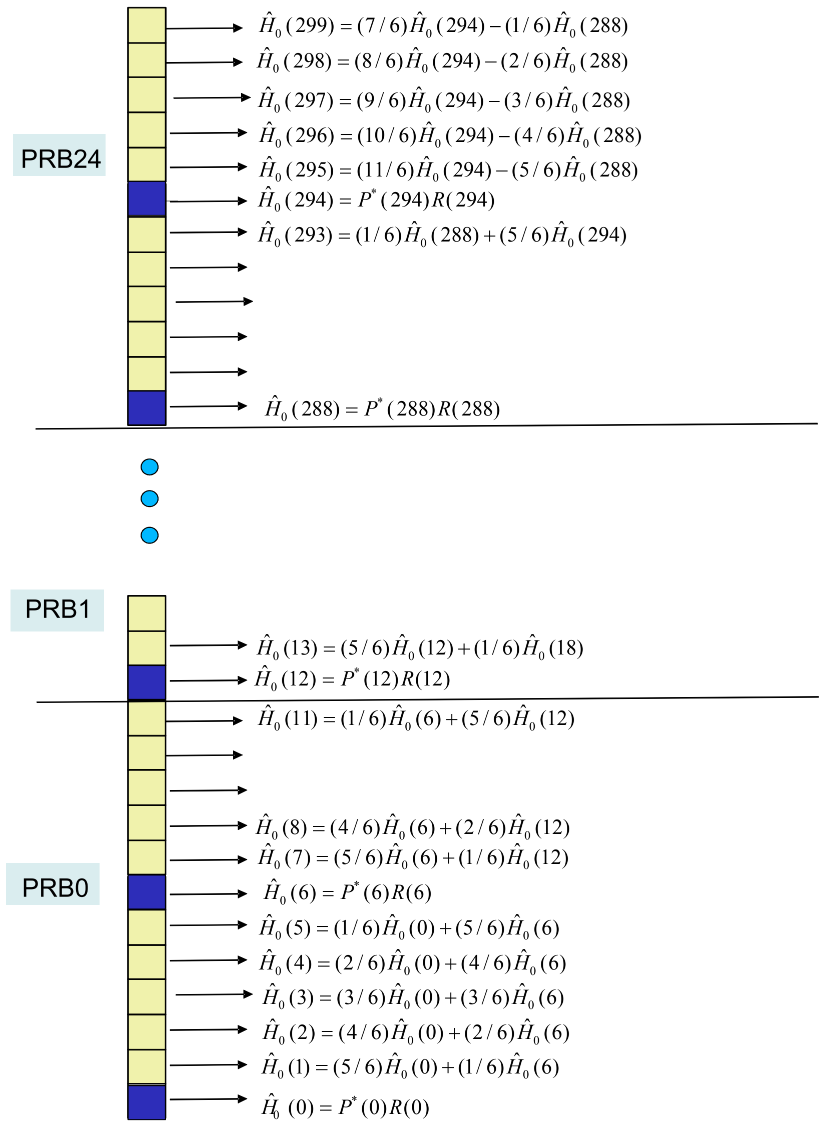
PRB24
$\hat{H}_0(299) = (7/6)\hat{H}_0(294) - (1/6)\hat{H}_0(288)$
$\hat{H}_0(298) = (8/6)\hat{H}_0(294) - (2/6)\hat{H}_0(288)$
$\hat{H}_0(297) = (9/6)\hat{H}_0(294) - (3/6)\hat{H}_0(288)$
$\hat{H}_0(296) = (10/6)\hat{H}_0(294) - (4/6)\hat{H}_0(288)$
$\hat{H}_0(295) = (11/6)\hat{H}_0(294) - (5/6)\hat{H}_0(288)$
$\hat{H}_0(294) = P^*(294)R(294)$
$\hat{H}_0(293) = (1/6)\hat{H}_0(288) + (5/6)\hat{H}_0(294)$
$\hat{H}_0(288) = P^*(288)R(288)$
PRB1
$\hat{H}_0(13) = (5/6)\hat{H}_0(12) + (1/6)\hat{H}_0(18)$
$\hat{H}_0(12) = P^*(12)R(12)$
$\hat{H}_0(11) = (1/6)\hat{H}_0(6) + (5/6)\hat{H}_0(12)$
$\hat{H}_0(8) = (4/6)\hat{H}_0(6) + (2/6)\hat{H}_0(12)$
$\hat{H}_0(7) = (5/6)\hat{H}_0(6) + (1/6)\hat{H}_0(12)$
PRB0
$\hat{H}_0(6) = P^*(6)R(6)$
$\hat{H}_0(5) = (1/6)\hat{H}_0(0) + (5/6)\hat{H}_0(6)$
$\hat{H}_0(4) = (2/6)\hat{H}_0(0) + (4/6)\hat{H}_0(6)$
$\hat{H}_0(3) = (3/6)\hat{H}_0(0) + (3/6)\hat{H}_0(6)$
$\hat{H}_0(2) = (4/6)\hat{H}_0(0) + (2/6)\hat{H}_0(6)$
$\hat{H}_0(1) = (5/6)\hat{H}_0(0) + (1/6)\hat{H}_0(6)$
$\hat{H}_0(0) = P^*(0)R(0)$

Figure 7.2: Liner Interpolation for SISO Case with RB = 25 and normal prefix

- Load the interpolation fractional constants α_0 and α_1 in the memory; where $\alpha_0 = \{1, \frac{5}{6}, \frac{4}{6}, \frac{3}{6}, \frac{2}{6}, \frac{1}{6}\}$ and $\alpha_1 = \{0, \frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}\}$

- Load the extrapolation fractional constants in the memory; β_0 and β_1; where $\beta_0 = \{\frac{11}{6}, \frac{10}{6}, \frac{9}{6}, \frac{8}{6}, \frac{7}{6}\}$ and $\beta_1 = \{\frac{5}{6}, \frac{4}{6}, \frac{3}{6}, \frac{2}{6}, \frac{1}{6}\}$

- Calculate $X = \alpha_0 * \hat{H}_i$ starting at first pilot index inside the symbol (index-0 in this case) and $Y = \alpha_1 * \hat{H}_i$ staring at second pilot index in the symbol (index-6 in this case).
  Here the *vec_mult* command of ASIP is used with the parameters set to multiply a constant complex value with a real valued vector. The results X and Y for the first 10 operations would look like:

  $$X[0:9] = H_0, \tfrac{5}{6}\hat{H}_0, \tfrac{4}{6}\hat{H}_0, \tfrac{3}{6}\hat{H}_0, \tfrac{2}{6}\hat{H}_0, \tfrac{1}{6}\hat{H}_0, \hat{H}_6, \tfrac{5}{6}\hat{H}_6, \tfrac{4}{6}\hat{H}_6$$

  $$Y[0:9] = \ 0, \tfrac{1}{6}\hat{H}_6, \tfrac{2}{6}\hat{H}_6, \tfrac{3}{6}\hat{H}_6, \tfrac{4}{6}\hat{H}_6, \tfrac{5}{6}\hat{H}_6, \ 0, \tfrac{1}{6}\hat{H}_{12}, \tfrac{2}{6}\hat{H}_{12}$$

- Sum up the previous product results $W = X + Y$. This operation uses the *vec_add* i.e. component-wise vector addition over a length $(N_p - 1) * 6 + 1$; where N_p is the number of pilots in the symbol. The remaining 5 sub-carrier positions are filled by the extrapolation operations.

- Now the extrapolation operations for the sub-carriers at the edges of the symbol, in this case the sub-carriers indices 295 - 299 as shown in figure 7.2. Calculate $X = \hat{H}_{(N_p-1)*6} * \beta_0$ and $Y = \hat{H}_{(N_p-2)*6} * \beta_1$. This uses the *vec_mult* command as explained above with length equal to 5.

- Next calculate $\hat{H}_{N_p*6+1} : \hat{H}_{N_p*6+5} = X - Y$ using the *vec_sub* component-wise subtraction of the ASIP module with length equal to 5.

- If the pilots are located at other indices (not at indices $\{0, 6\}$), the extrapolation operations are the same with change in the fractional co-efficients. The respective fractional co-efficients for all the possible pilot locations are stored in the memory (as there are only 6 possible combinations). The pilot index information is passed to the block by the higher layers.

- Now we have the complete vector $\hat{H}_{0,i}$ over sub-carrier length i and for antenna 0. The procedure is repeated for symbols $\{4, 7, 11\}$

- The temporal interpolation procedure is used for symbols $\{1, 2, 3, 5, 6, 8, 9, 10, 12, 13\}$. The procedure for one symbol, lets say symbol-1 is:

- Calculate $\frac{3}{4} * \hat{H}_{0,i}(0)$ and $\frac{1}{4} * \hat{H}_{0,i}(3)$ using the *vec_mult*.

- Use the *vec_sum* for the two fractions calculated above.

- The procedure is repeated for all the other symbols without reference pilots in the subframe.

To illustrate more, a part of the assembler code for the ASIP design for the above operation is depicted here. The code shown computes the channel estimate at pilot positions and then at the intermediate positions using linear interpolation. The extrapolation for the remaining 6 sub-carriers is computed with similar approach and is not shown here. Similarly, the inter-symbol temporal interpolation is computed in the ASIP design.

In case of the FEP, the command words are passed to the VCI-Interface of the FEP-Core. The FEP core reads out the commands from the VCI after completing each task. The command words for the DFT/IDFT mode is of 64 bits, while the command word for the pre-post processing are 128 bits and the parameters are explained in the chapter 5. Here we show a user view of the commands to the FEP, the first command is to compute the DFT of the received signal and the 2-nd is to calculate the component-wise-multiplication of the received signal and the reference signal at the pilot positions.

```
FT_Mode_Command{
                        DFT        ; the operation to be computed, 1-bit
                        Size       ; Input Vector Size
                        Addresses  ; input, output addresses
                }
```

ASIP Assembler Code	**Comments**
agu_cfg_vectors 300 0 0 0 1	Vector size = 300
agu_set_vec0_addr 0x0	Address for reference pilots
agu_set_vec1_addr 0x08000	Address for received signal (after DFT)
agu_set_res_addr 0x10000	Output address
agu_cfg_sub_vec_a 1 300	Configure the sub-vectors, 1 with size 300
agu_cfg_sub_vec_b 0 6	start at index - 0, and skip 6 addresses at each step
nop	
vec_mult 0 0 0 0 0 0	Vector Multiplication of 2 input vectors
nop	So we have channel estimate at pilot positions : $\hat{H_{i,p}}$ as per equation 7.1
	Next is linear interpolation
agu_cfg_vectors 294 0 0 0 1	Set Vector Size for interpolation, for other 6 sub-carriers the extrapolation process
agu_set_vec0_addr 0x10000	Location of $\hat{H_{i,p}}$ from previous step
agu_set_vec1_addr 0x18000	Address of co-efficients α_0
agu_set_res_addr 0x0	Result Address
agu_cfg_sub_vec_a 49 6	Number of sub-vectors = 49, Size of each sub-vector= 6
agu_cfg_sub_vec_b 0 1	Start at index 0, and skip = 1 i.e. consecutive
nop	
vec_mult 0 0 0 0 1 0	Vector Multiplication with second input a real valued vector
nop	*We have* $\alpha_0 * \hat{H_{i,p}}$
agu_cfg_vectors 294 0 0 0 1	
agu_set_vec0_addr 0x10018	Start from 2nd pilot at location 6 (Byte addressing 6*4 = 0x18)
agu_set_vec1_addr 0x18120	Location of co-efficients α_1
agu_set_res_addr 0x08000	
nop	
agu_cfg_sub_vec_a 49 6	
agu_cfg_sub_vec_b 0 1	
nop	
vec_mult 0 0 0 0 1 0	*We have* $\alpha_1 * \hat{H_{i,p}}$
nop	
	Now sum up the last two results
agu_cfg_vectors 294 0 0 0 1	
agu_set_vec0_addr 0x0	$\alpha_0 * \hat{H_{i,p}}$
agu_set_vec1_addr 0x08000	$\alpha_1 * \hat{H_{i,p}}$
agu_set_res_addr 0x10000	
agu_cfg_sub_vec_a 1 294	
agu_cfg_sub_vec_b 0 1	
nop	
vec_add 0 0 0 0 0 0	*The Linear Interpolation result*

Table 7.2: Glimpse of the assembler code for linear interpolation in the ASIP design

```
PP_Mode_Command{
        CWM                   ; The specific bit for component-wise-mult is set '1'
        Sub_band_count        ; The number of sub-bands
        Sub_band_span         ; Span of each sub-band
        sub_sampling_factor   ; distance between 2 consecutive samples inside sub-band
        offset                ; at the start of each sub-band
        conj_1                ; conjugate the 2-nd input
        Addresses             ; input-1, input-2, and output
}
```

So both the processing units are equally capable of computing the required operations. It is also worth mentioning that for any time frequency conversions, only the FEP block can be used, the vector processor (VP) does not compute DFT or IDFT. The approaches to compute the functions in both cases are quite similar, and the reason is the similar functional specifications. However the mechanism to program the units and the computations inside are processor specific, the performance comparison of the two blocks has been presented earlier in this chapter. This section depicts how both the blocks can serve the interpolation tasks. Next we consider the multiple antenna case for LTE receiver.

Case: MIMO with normal cyclic prefix and 2 antennas

Figure 7.3 shows the downlink sub-carriers arrangements for two antennas case with normal cyclic prefix. The pilots are distributed over the symbols $\{0, 4, 7, 11\}$ in each subframe. Inside each resource block there are two pilots for each antenna. Figure 7.4 explains the linear interpolation and extrapolation for MIMO case with 2 antennas and 25 resource blocks. As the distance between the consecutive pilots for respective antenna is constant over the whole symbol length, therefore the fractional co-efficient for pilots are periodic over the distance between the pilots i.e. 6 for both the antennas. The extrapolation is also defined over the last resource block. The co-efficients listed are for the case, when the pilots are at positions $\{0, 6\}$ inside each resource block. The pilots position inside each resource block varies depending on the cell ID and it may have any other values as well. Thus the fractional co-efficients for the interpolation are constant, while for

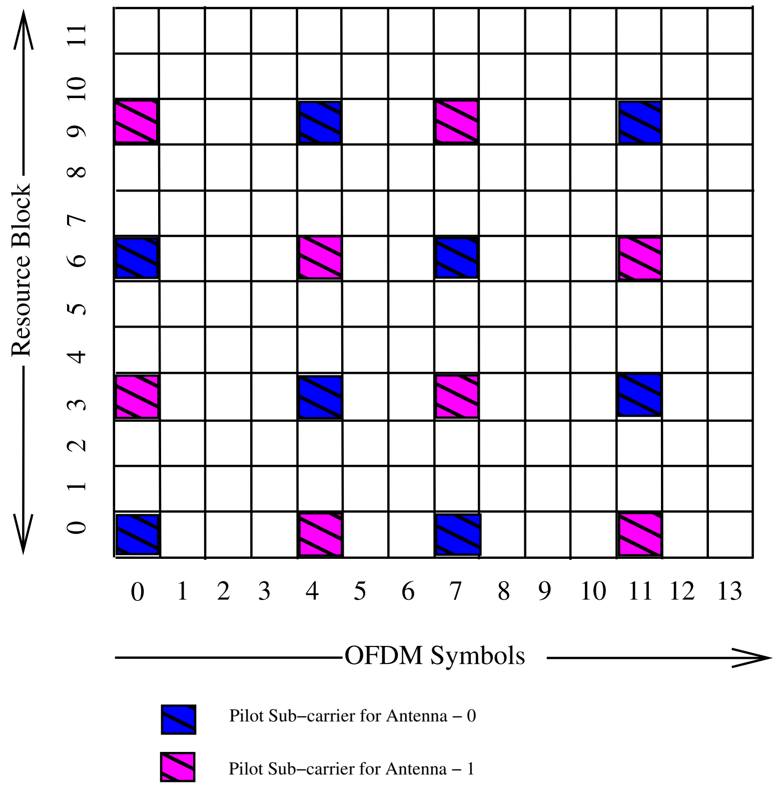

Figure 7.3: Mapping of LTE Downlink Reference Signals for MIMO (2-antennas) with normal prefix

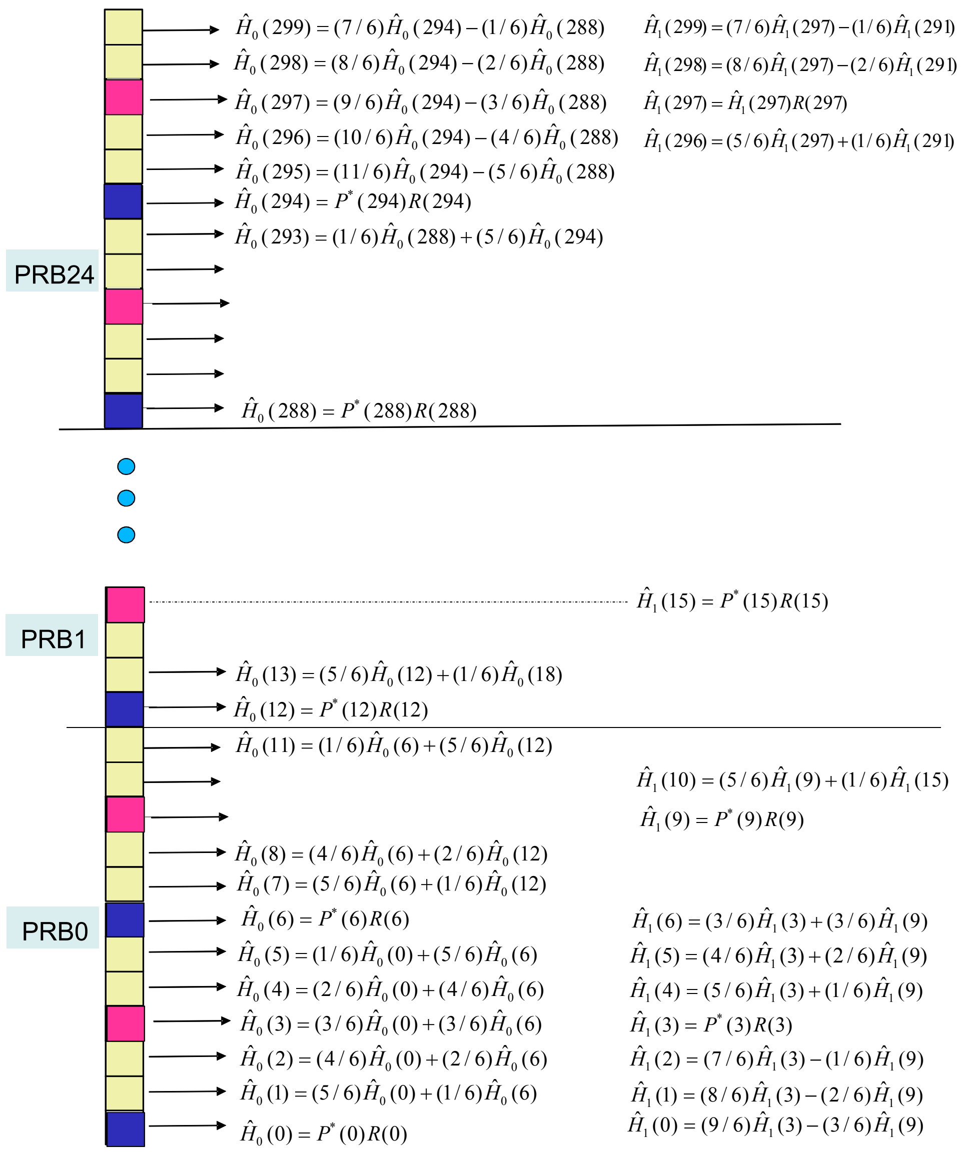
$\hat{H}_0(299) = (7/6)\hat{H}_0(294) - (1/6)\hat{H}_0(288)$
$\hat{H}_1(299) = (7/6)\hat{H}_1(297) - (1/6)\hat{H}_1(291)$
$\hat{H}_0(298) = (8/6)\hat{H}_0(294) - (2/6)\hat{H}_0(288)$
$\hat{H}_1(298) = (8/6)\hat{H}_1(297) - (2/6)\hat{H}_1(291)$
$\hat{H}_0(297) = (9/6)\hat{H}_0(294) - (3/6)\hat{H}_0(288)$
$\hat{H}_1(297) = \hat{H}_1(297)R(297)$
$\hat{H}_0(296) = (10/6)\hat{H}_0(294) - (4/6)\hat{H}_0(288)$
$\hat{H}_1(296) = (5/6)\hat{H}_1(297) + (1/6)\hat{H}_1(291)$
$\hat{H}_0(295) = (11/6)\hat{H}_0(294) - (5/6)\hat{H}_0(288)$
$\hat{H}_0(294) = P^*(294)R(294)$
$\hat{H}_0(293) = (1/6)\hat{H}_0(288) + (5/6)\hat{H}_0(294)$
PRB24
$\hat{H}_0(288) = P^*(288)R(288)$
$\hat{H}_1(15) = P^*(15)R(15)$
PRB1
$\hat{H}_0(13) = (5/6)\hat{H}_0(12) + (1/6)\hat{H}_0(18)$
$\hat{H}_0(12) = P^*(12)R(12)$
$\hat{H}_0(11) = (1/6)\hat{H}_0(6) + (5/6)\hat{H}_0(12)$
$\hat{H}_1(10) = (5/6)\hat{H}_1(9) + (1/6)\hat{H}_1(15)$
$\hat{H}_1(9) = P^*(9)R(9)$
$\hat{H}_0(8) = (4/6)\hat{H}_0(6) + (2/6)\hat{H}_0(12)$
$\hat{H}_0(7) = (5/6)\hat{H}_0(6) + (1/6)\hat{H}_0(12)$
PRB0
$\hat{H}_0(6) = P^*(6)R(6)$
$\hat{H}_1(6) = (3/6)\hat{H}_1(3) + (3/6)\hat{H}_1(9)$
$\hat{H}_0(5) = (1/6)\hat{H}_0(0) + (5/6)\hat{H}_0(6)$
$\hat{H}_1(5) = (4/6)\hat{H}_1(3) + (2/6)\hat{H}_1(9)$
$\hat{H}_0(4) = (2/6)\hat{H}_0(0) + (4/6)\hat{H}_0(6)$
$\hat{H}_1(4) = (5/6)\hat{H}_1(3) + (1/6)\hat{H}_1(9)$
$\hat{H}_0(3) = (3/6)\hat{H}_0(0) + (3/6)\hat{H}_0(6)$
$\hat{H}_1(3) = P^*(3)R(3)$
$\hat{H}_0(2) = (4/6)\hat{H}_0(0) + (2/6)\hat{H}_0(6)$
$\hat{H}_1(2) = (7/6)\hat{H}_1(3) - (1/6)\hat{H}_1(9)$
$\hat{H}_0(1) = (5/6)\hat{H}_0(0) + (1/6)\hat{H}_0(6)$
$\hat{H}_1(1) = (8/6)\hat{H}_1(3) - (2/6)\hat{H}_1(9)$
$\hat{H}_0(0) = P^*(0)R(0)$
$\hat{H}_1(0) = (9/6)\hat{H}_1(3) - (3/6)\hat{H}_1(9)$

Figure 7.4: Liner Interpolation for MIMO Case (2-antennas) with RB = 25 and normal prefix

extrapolation those are bit different for two different pilot locations.

The procedure for inter-symbol interpolation and intra-symbol interpolation in $MIMO$ case is similar to what we explained previously for $SISO$ case. The only difference is that the procedure is repeated for all the antennas (in this case 2 as shown in Figure 7.4). First the $\hat{H}_{0,i}(n)$ for antenna 0, for all sub-carriers i, and for symbols n containing pilots is calculated as explained in the previous section. Then $\hat{H}_{1,i}(n)$ for antenna 1 over sub-carriers i for symbols n is computed. The temporal procedure is also repeated individually for all the antennas.

Case: Extended Cyclic Prefix

In figure 7.5, the subframe arrangement for the reference pilots in case of extended cyclic prefix is depicted. The only difference between the extended prefix and the normal prefix is the number of the symbols per subframe which is reduced to 12 from 14. This has an effect on the temporal interpolation between the symbols while there is no effect on the linear interpolation inside the symbols. The difference in the procedure is explained here:

- Calculate the linear interpolation and extrapolation sub-carrier values using the procedure explained in SISO case for symbols $\{0, 3, 6, 9\}$.
- For the remaining symbols, use the temporal interpolation. The procedure for symbols $\{1, 2\}$ is:

$$\hat{H}_{x,i}(1) = \frac{2}{3}\hat{H}_{x,i}(0) + \frac{1}{3}\hat{H}_{x,i}(3) \tag{7.5}$$

$$\hat{H}_{x,i}(2) = \frac{1}{2}\hat{H}_{x,i}(0) + \frac{1}{2}\hat{H}_{x,i}(3) \tag{7.6}$$

  where $H_{x,i}(1)$ is channel estimate for antenna x (can be 0 or 1 in this case), sub-carrier i, and symbol 1. The changed procedure in the temporal interpolation can be catered by our hardware unit easily as only the fractional co-efficients are changed while the processing units are the same. The similar procedure is repeated for all the other symbols without reference pilots inside the subframe.

7.3.3 Pre-Defined Filter Interpolation in Frequency Domain

Another scheme in the frequency domain is to interpolate using a pre-defined filter. e.g. sinc function which has been used to reconstruct the continuous

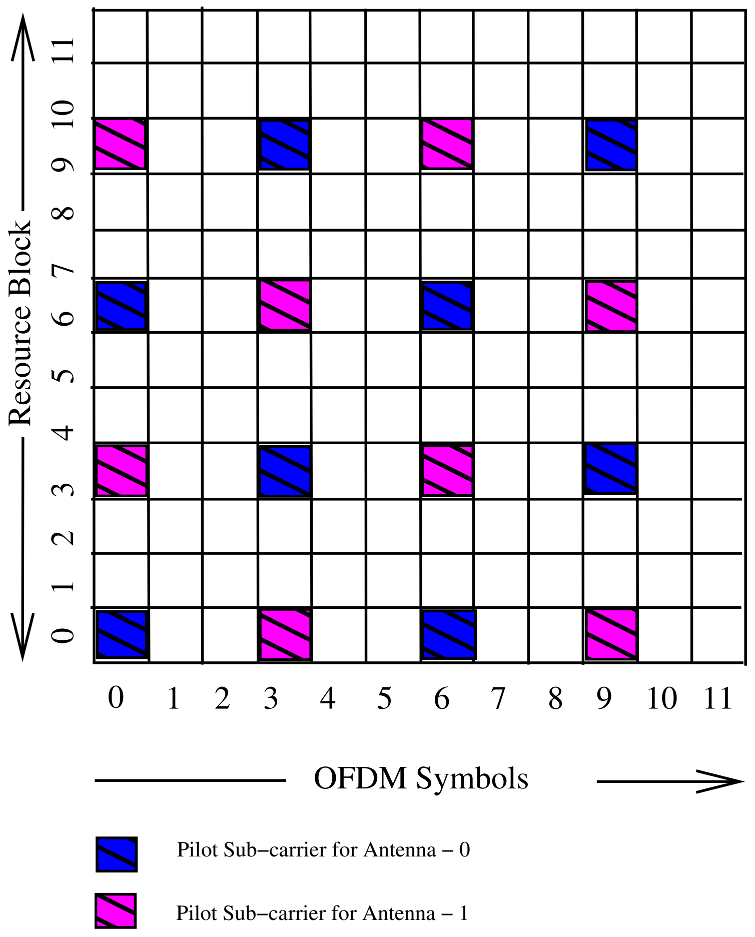
Resource Block
0
1
2
3
4
5
6
7
8
9
10
11
0 1 2 3 4 5 6 7 8 9 10 11
OFDM Symbols
Pilot Sub–carrier for Antenna – 0
Pilot Sub–carrier for Antenna – 1

Figure 7.5: Mapping of LTE Downlink Reference Signals for MIMO (2-antennas) with extended prefix

signal or increase the sampling rate in the digital signal processing domain. To map the calculated channel response at pilot positions to the whole resource block (sub-carriers):

$$\hat{H} = \sum_{i=0}^{f_l-1} H_{p,i} F_i \tag{7.7}$$

F is a predefined filter (a sinc function in this case), and its length f_l is dependent on coherence bandwidth of the channel. The filter length is either 48 or 96 in our case. $\hat{H}$ values except from pilot positions are set equal to zero. The filter is moved over all of the symbols (multiple resource blocks).

Lets consider the following case:
f_l: Filter Length = 96
N_{RB}: Number of Resource Blocks = 25
N_P: Number of Pilots = 50
N_{sc}: Number of sub-carriers = 300

```
    $for(i = 0, i < (N_{sc} - 1), i + 6)$
$\hat{H} += \hat{H} * F(\frac{f_l}{2} - i)$
$end\ for$
```

The length of vector H is equal to number of sub-carriers i.e. 300. However the number of multiplications taking place in each step of loop are equal to the length of filter f_l which is less than the number of sub-carriers (96 in this scenario). The loop is repeated N_P (number of pilots) times, therefore the total number of multipliers are $N_P * f_l$. As the values of the filter are real, the total number of real multipliers is $2 * N_P * f_l$.

At the start, the filter function is centered at $\frac{f_l}{2}$, so that the first pilot is multiplied by the peak value of the *sinc* function. The filter is moved to the next pilot position after each loop cycle, and the procedure is repeated till the end. Then, $\hat{H}$ is summed up giving the final result for the channel estimate.

For hardware implementation, the summation will take place after each loop step as we have the $2-$ input adders for component-wise additions. The number of adders for each loop cycle are equal to filter length and total number of adders will be $N_P * f_l$.

The FEP and the VP both can handle the multiplication and addition processes required by pre-defined filter. From the performance estimation point of view, *sinc* might not be a good choice, however any other filter can be chosen.

7.3.4 Interpolation by Zero Padding in Time Domain

Another scheme that can be used to compute the channel response at the intermediate sub-carriers between the pilot position is by using the DFT / IDFT blocks of the FEP block. The fundamental principles of discrete signals is that "zero padding" in one domain results in an increased sampling rate in the other domain. For example, the most common form of zero padding is to append a string of zero-valued samples to the end of some time-domain sequence, and if we go to frequency domain the signal will be sampled over string length plus the original length.
In our case, since we have the channel estimate at the pilot positions $\hat{H}$ we go in the time domain and append the zeros. The number of appended zeros per symbol per antenna is dependent on the number of resource blocks being transmitted. Each resource block is composed of 12 sub-carriers and has two pilots per antenna. The procedure followed is given by:

$$\hat{H}_{i,6N_p} = DFT_{6N_p}(IDFT(\hat{H}_{i,N_p})||ZE_{5N_p}) \tag{7.8}$$

The $\hat{H_{i,N_p}}$ represents the channel estimate vector for antenna i at the pilot position and has length equal to number of pilots N_p. First the $IDFT$ takes the channel estimate into the time domain and then zeros are appended. The symbol $||$ is used to represent the appending operation while ZE_{5N_p} describes the zero stream with length $5N_p$. Once the zeros are appended then the resultant vector is converted back in to frequency domain by taking the DFT with vector length $6N_p$.

The hardware block, FEP, in our baseband design is well equipped to carry out the above mentioned three step procedure. However, the Fourier Transform (FT) unit of the FEP works only at the vectors with lengths that are power of 2. So the number of zeros appended are increased to make the vector length equal to power of two. e.g. in case of 15 resource blocks, in the first step two zeros are added in the center of frequency domain signal to get the vector length equal to 32 (2 pilots per resource block), then in the time domain instead of appending 148 zeros to achieve the required size of 180, we append 224 zeros to get vector size of 256 so that the DFT unit of the FEP block can be applied. Thus the DFT vector is always 8 times

than the IDFT vector contrary to 6 times as listed in equation 7.8, because of the hardware limitation of the FEP block. The DFT, IDFT pair for all the permissible transmission bandwidth of the LTE are shown in table 7.3. The inter-symbol interpolation procedure remains the same where we used the temporal linear interpolation for the channel estimate of the symbols without any reference pilot symbols.
With a detailed discussion on LTE channel estimation (which uses diffused pilot formation) and performance analysis [STB09], the linear interpolation in frequency domain is better choice. It is simpler as well and performs better.
Once the interpolation schemes are listed, the next step is to compare the schemes on the basis of resource utilized by our hardware blocks. The next section presents the comparative analysis.

7.3.5 Hardware Resource Utilization for the Interpolation Schemes

In this subsection, we look at the resources required for the interpolation schemes stated in the previous section. For the temporal interpolation between the symbols with the pilots, all the schemes use the same procedure; therefore the analysis will focus on the interpolation inside the symbol.

In case of linear interpolation, the number of real multiplications per symbol are in the order of $4N_{sc}$ per antenna; where N_{sc} is the number of the sub-carriers. The number of real additions are in the order of $2N_{sc}$.

The time domain zero padding uses the Fourier Transform mode of the FEP block. The input vector size used to compute the DFT and IDFT is known for each case of transmission bandwidth as explained in the interpolation section. The comparison of the multipliers and adders usage is shown in table 7.3.

The multiplier and accumulator count for the DFT / IDFT operations has been calculated without using any of the fancy schemes available. It is based on the simple implementation we have i.e. radix-4 algorithm and mixed radix algorithm (as explained in chapter 5). The radix-4 algorithm is used for the input vector size that is power of 4, while the mixed radix is used for the vector size that is power of 2 and not power of 4. The number of complex multiplier used for radix-4 algorithm are $\frac{3}{4}N(log_4 N - 1)$, while the number of multipliers for the mixed radix are $\frac{3}{4}N(log_4 \frac{N}{2} - 1) + N$; where N represent the size of input vector. The calculation also takes into

Trans-mission BW (MHz)	RBs	Linear Interpolation		Time Domain Zero Padding		
		Real Mult Count	Real Accum Count	DFT / IDFT Pair	Real Mult Count	Real Accum Count
1.4	6	288	144	16, 128	1328	5312
3	15	720	360	32, 256	2528	10112
5	25	1200	600	64, 512	7040	28160
10	50	2400	1200	128, 1024	13568	54272
15	75	3600	1800	256, 2048	35072	140288
20	100	4800	2400	256, 2048	35072	140288

Table 7.3: Interpolation Schemes Comparison on the basis of arithmetic operations count

account that each complex multiplier is equivalent to four real multipliers. The number of real adders calculations are based on similar pattern.
To improve the results for the time domain zero padding interpolation algorithms, the DFT and the IDFT algorithms can be implemented using some efficient schemes with some added logic and complexity. However, the difference between the two schemes, on the basis of multiplier and accumulator count, is huge and in our opinion that would not add any benefit in this specific case. The linear interpolation scheme remain much more efficient based on the arithmetic operation count.

From the above discussion, it is evident that linear interpolation in frequency domain procedure is much more efficient than time domain method using the proposed hardware blocks in our design. In the following, we consider the memory requirements for the linear interpolation procedure and try to work out the number of symbols that can be worked at the same time inside our front end processor (FEP).

7.3.6 Memory Requirements for Linear Interpolation in Frequency Domain Scheme

We consider the memory requirements for the linear interpolation scheme, and analyze that how many symbols our hardware blocks can accommodate. The aspects to be considered are :

- The ASIC design i.e. FEP and the ASIP design i.e. VP use the same internal memory, therefore the analysis will be the same for both of the processors.
- If the linear interpolation scheme is used, then the internal memory would be accessed in the PP-Mode as described in section Memory Access Schema (5.4.1) of this thesis report.
- The memory subsystem of the FEP is composed of 4 chunks that can be accessed separately as true dual port RAM i.e. two reads OR two writes OR one read and one write operations are possible each cycle. Therefore the total number of read and write operations per cycle with the memory subsystem are 8.
- Considering the LTE Downlink Reference Signals for SISO and MIMO case with normal prefix, five received symbols need to be inside the internal memory of the FEP. This is due to the fact that the temporal interpolation takes place between symbol one and five to compute the channel estimation for the intermediate non pilot carrying symbols. In case of extended prefix, 4 symbols would be loaded in the internal memory of the FEP block.
- Less than 5 symbols for normal prefix may also be loaded in the internal memory but that would require some complex control of the addressing schemes of the memory subsystem and might become inefficient at the end.
- The memory subsystem also needs the reference signal inside the memory to compute the channel estimate. Memory space equal to that of symbol size is required to store the channel estimate of the symbol and as much space is required as working memory to store the intermediate results or the multiplicative co-efficients etc.
- Once the channel estimate is computed for one symbol, the channel compensation for that symbol is carried out and the DMA takes out the final result for that specific symbol if the said symbol is not to be used any further in the channel estimation procedure. This helps to maintain only the minimum symbols inside the FEP memory subsystem.
- If the memory size of symbol is given by mem_{sy}, then the internal memory space requirement would be $8 * mem_{sy}$. Five are the received

Transmission BW	**Resource Blocks**	**Memory Usage - Bits**	
		SISO	MIMO (2-Antennas)
1.4	6	18432	36864
3	15	46080	92160
5	25	76800	153600
10	50	153600	307200
15	75	230400	460800
20	100	307200	614400

Table 7.4: Memory Requirements for linear interpolation scheme with normal prefix

symbols, one reference symbol, one for the channel estimated symbol and one for temporary results.

- The size of symbols varies with the transmission bandwidth, and the table 7.4 summarizes the memory requirements for single antenna and 2-antennas.

- Thanks to the 4 independent chunks of the FEP memory subsystem, no computation deadlock or added latency is expected between the two consecutive computations for channel estimation or compensation. The operations require 2 input operands which are stored in two different blocks, while the output is stored in the third block, thus at most three memory sub-blocks out of four sub-blocks are used at a time.

- The 2-antenna case has independent stream for each antenna, thus requires extra memory to store the channel estimate for each antenna and intermediate results.

The total memory size of the FEP or the VP block for the input and output data is $1Mbits$, therefore for the SISO case the memory size is not sufficient for transmission bandwidth of $\{10, 15, 20\}MHz$ if we load 5 symbols at a time for normal prefix case. Therefore, a swapping with the outside memory would be required. The external memories like double data rate dynamic random access memory (DDR DRAM) can be added to serve the purpose, and the memory space can be shared by the other IPs in the design, when the FEP is performing other operations. Another option can be the usage of the internal memory of the FEP block, that is otherwise used to store the intermediate results while computing the DFT or IDFT. Since the DFT mode and pre-post processing mode of the FEP block are mutually exclusive, therefore the internal FEP memory can be used with out an inter dependency in the data. However, the use of external memories and moving data out for the high data rate and computation intensive standards like LTE is unavoidable. The usage of such memories along with access frame work is an interesting research issue to study and stands as one of the future tasks of the presented work.

Summary

This section described the channel estimation schemes for the LTE standard, and also listed the procedure that our hardware blocks follow to carry out the operations. Then the comaprison of the processes with our design lead to the conclusion to use the linear interpolation scheme for the channel estimation procedure for LTE. Further we depicted the memory space utilization for the linear interpolation scheme and found out that swapping would be required to compute the channel estimates with our current memory subsystem. The precise swapping mechanism and access methodologies for the external memories are to be studied to make a final decision.

Chapter 8

Conclusions and Future Work

The presented thesis has demonstrated a practical approach for implementing the baseband radio architectures for flexible platforms. The multi-standard baseband architecture poses great design challenges that include but are not limited to diverse data throughput, latency, and timing constraints. The baseband design not only needs to address the different standards or specifications but the diverse set of functions / operations for each and every wireless application. The digital front end processor (FEP) takes care of operations at the air interface in the baseband design. We presented the functional specifications, the hardware design, and implementation of this block in detail. To make the analysis spectrum wider, we also presented the ASIP design of a vector processor using LISA language. The vector processor carries out the basic arithmetic operations over complex vectors. We also discussed the hardware design approaches in the context of the multi-standard designs and proposed some recommendations for different categories of operations.

In this work, we first proposed a global baseband processing architecture that can be configured to support almost all the existing and in-progress wireless communication standards. The baseband design keeps the control and processing separate to keep a hierarchy, and making the design simpler and easy to implement and modify. The modification property is more significant in the SDR or multi-standard context as the new wireless applications and standards keep pouring in. As far as the hardware technologies are concerned, our baseband design is a hybrid solution. We use FPGAs to implement the control and processing unit. The choice of the FPGAs is solely based on the initial goal to first come up with an experimental platform and once the prototype design is finalized, the baseband design would be reworked and moved to the system on chip technology. The choice of FPGA for prototyping is based on on its reduced design cycle, flexibility, ease of use and lower costs.

Our proposed baseband design is composed of two main parts: a high level control module and a Digital Signal Processing engine. Based on our analysis of the hardware technologies and types of control / processing tasks inside a multi-standard wireless communication system, the control of the baseband design is assigned to a general purpose processor. Moreover, the hierarchical control design takes care of multiple complex procedures / threads running on the platform simultaneously. It is also worth mentioning here that our design and implementation, both are open source and available to research community. The design of the processing engine inside the baseband design is based on flexible generic units in the wireless communication systems such as Channel decoder, demodulator etc. Each of these units is capable of carrying out the specific set of functions for all the standards in an efficient manner i.e. meeting the performance requirements as specified by the all the individual standards. Each of the individual block design follows a global pattern, where the communication model with the global design, memory system and the interfaces are pre-defined. This approach helps to keep the global design as much synchronized as possible, and also makes the additions of new units or blocks quite easy in case of emergence of new applications or standards. The design approach for individual hardware units or IPs is to identify small macro processing blocks based on the functionalities required which can be re-used for multiple standards, thus providing hardware flexibility and higher efficiency.

The front end processor (FEP), the flexible hardware unit inside the processing engine, which is designed to cater all the operations at the air interfaces is presented in detail in this report. The operations at the air-interface include channel estimation, channel compensation, synchronization, and data detection etc. First of all, the set of operations is defined based on the analysis of the different air-interfaces in the wireless communication standards. The aim of this analysis was to find out the commonalities that exist among the standards, and then identify the macro-blocks that are required to carry out all the operations for all the air-interfaces. Once the macro operations are defined, then the performance criteria was set for these units considering the specifications of all the standards. The FEP carries out the variable length DFT / IDFT operation along with arithmetic operations on long complex vectors. The FEP is also capable of computing the dot product, energy and maximum calculations over complex vectors. The design is implemented at RTL level and is synthesized with FPGA as target technology, however the design is totally portable. The resource utilization is quite reasonable and it meets the performance requirements easily as well.

Our baseband design is mainly composed of hardware blocks that are flexible enough to adopt any existing communication standard. In the first prototype, we designed these blocks as the hardwired or the ASIC units while the flexibility comes from the fact that the units can be parametrized as per the requirements of any standard. The units can serve any of such requirements with no changes in the hardware, and usage of internal flexible hardware macro blocks that support multiple set of inputs for each functionality. e.g. the DFT block supports all the power of 2 input vector sizes between 8 and 4096.

During the first prototyping, it was realized that more flexible solutions are required to efficiently execute some of the operations in the multi-standard transceiver design e.g. the channel decoder interleaving. The Application Specific Instruction Processors (ASIPs) stand as one of the choices for a complex flexible platform design. For the ASIP design, if high flexibility and customization for the instruction set architecture (ISA) is required then tools that focus on architecture level optimization may be used. We used LIA, architecture description language, to design an ASIP core to evaluate its usefulness in baseband architectures for SDR applications. We designed a vector processor that carries out almost all the arithmetic operations on complex vectors. Complex vector arithmetic operations are integral part of the FEP and used in other blocks of baseband design as well. The design results are encouraging and we think that the tool based programmable design can be quite handy in the rapid design of flexible baseband design. We showed in our design that the ASIPs do provide higher flexibility with not much cost on the performance. The ASIPs stand a strong candidate in a hybrid design for flexible radio designs.

Based on our experience with global baseband design and hardware block design using ASIC and ASIP approaches for the flexible radios, we did analyze the hardware technologies being used for communication systems design. We analyzed the GPPs, DSPs, FPGAs, ASICs, and the ASIPs against the key parameters of flexible baseband design such as flexibility, throughput, complexity and regularity of the operations. We think that hybrid solutions, where specific set of tasks are assigned to individual category in the hardware technologies would be the most effective ones e.g. the control tasks to GPPs, regular and frequently occurring tasks to the ASICs etc.

In short, the research work is an attempt to lay down a solid foundation for a practical design of a multi-standard baseband design. We not only explored the design options, but also provided the design and the implementations using the ASIC and ASIP approaches. The flexible, programmable multi-standard baseband designs have a tremendous scope in the wireless communication devices in the years to come. Our analysis, design and implementation may be useful for the researchers to explore new horizons in the fields of flexible baseband design for multi standard applications.

8.1 Future Work Directions

Like most of the research works, there are questions that still need to be answered and there are features that need to be explored in the context of the presented work. Some of those are listed here:

- How more flexibility may be added to the current design of the FEP. The current design of the FEP block supports only one operation at a time, and the next task has to wait till the completion of the in-progress task. The FEP block should provide a feature to carry out multiple operations at the same time, or at least switching between the operations if a high priority task is assigned by the system.
- Though the ASIP design was compared with the hardwired RTL solution, but still there is a lot of room for more rigorous analysis. The both units must also be compared on the power consumption analysis before making a final call for the use of hardware units in the baseband design.
- The integration of the FEP and the ASIP design at the same time in the Open Air Interface platform and the evaluation of the real time results would also be a real plus to have.

- We recommended to use a hybrid structure for the flexible baseband design. It would be really interesting to have an ASIP design for the channel decoder unit, for example, in the baseband design and have a performance evaluation. This would provide some practical results to support our hypothesis.

Chapter 9

Résumé en Français

9.1 Abstarct

Les dernières décennies ont vécu un développement technologique rapide et une diversité dans les services de communications sans fil. Ce développement s'explique par l'utilisation de plus en plus répandue, dans la vie de tous les jours, de dispositifs sans fil avec des nouvelles fonctionnalités. Cette diversité technologique est due aussi à la diversité et la nouveauté des applications proposées. Aujourd'hui, il existe plusieurs standards pour les réseaux des téléphones sans fil (GSM, EDGE, WCDMA etc.), également pour les réseaux locaux sans fil (IEEE 802.11a, b, g). Actuellement, chaque standard possède sa propre fréquence porteuse, largeur de bande et modulation. Le développement considérable de ces standards modernes et de leurs applications nécessite une plateforme matérielle flexible capable de gérer ces standards divers dans toute la bande de fréquences de communication sans fil. Donc, avoir des processeurs flexibles et efficaces est impératif pour supporter des plateformes radio multistandards.

On présente un prototype d'architecture générique de traitement numérique en bande de base pour une application radio logicielle compatible avec les besoins actuels de traitement UMTS mais qui anticipe également d'une manière efficace les besoins de traitement des standards émergents (3GPP Long Term Evolution - LTE). L'architecture bande de base proposée est capable d'implémenter les standards 2G, 3G, 4G de communication ainsi

que les standards de réseau local sans fil. Le partitionnement entre matériel et logiciel induit un compromis coût/complexité et rapidité. Le contrôle s'effectue dans la partie logicielle qui passe les paramètres pertinents pour des fonctionnalités spécifiques à la partie matérielle. Le matériel est conçu de façon à effectuer la plus part des tâches de calcul intensif efficacement, c'est-à-dire soutenir les débits requis et respecter les délais nécessaires. Le matériel est aussi suffisamment flexible pour utiliser les mêmes ressources de traitement en bande de base pour plusieurs standards. L'architecture configurable présentée profite des ressemblances qui existent entre les différents standards à implémenter d'une manière efficace. Les ressemblances et les dissemblances sont traduites dans l'architecture matérielle et conduisent à un système qui réalise tous les traitements nécessaires pour toutes les applications. Le produit final va permettre à l'utilisateur d'exécuter le standard désiré en fournissant seulement les paramètres sans intervenir dans les détails de l'architecture. La solution proposée est largement plus efficace que d'avoir des blocs dédiés à chaque application.

Les solutions multistandards doivent avoir une haute performance pour se conformer avec le débit moyen et les contraintes temporelles de tous les standards avec le même ensemble logiciel/matériel. Afin d'explorer les critères de performance qui président à la conception de la bande de base, nous présentons une spécification et une implémentation des blocs matériels en utilisant deux approches : circuits intégrés spécifiques (ASIC) et processeurs à jeu d'instruction spécifique (ASIP). L'ASIP offre plus de flexibilité et programabilité au dépend d'une petite perte de performance. Nous considérons également les autres cibles technologies existantes, leurs avantages et défauts spécifiques et comparons ceux-ci pour différentes catégories de type de calcul dans la bande de base et proposons quelques recommandations sur la conception d'une bande de base multistandard.

9.2 Introduction

Les trois dernières décennies ont vu les communications sans fil comme le plus grand succès de génie, pas seulement dans la recherche et développement, mais aussi la taille du marché et l'impact sur la société. Les appareils sans fil sont devenus une partie intégrante de notre vie quotidienne, et sont également tirer un gros morceau de l'économie. Au début, les téléphones cellulaires sont considérés comme des dispositifs de communication sans fil, mais maintenant c'est changé beaucoup; des réseaux informatiques sans fil, réseaux de capteurs sans fil et les systèmes de positionnement sans fil sont quelques-uns pour citer ici; qui ont changé la composition de leurs applications et, surtout, ont créé de nouvelles orientations de la recherche en systèmes de communication.

Différents types d'applications et usages ont permis à l'élaboration de différentes standards utilisées dans les systèmes de communications sans fil. Bien que ces systèmes ont presque les mêmes blocs fonctionnels, mais l'approche et, partant, les algorithmes utilisés diffère beaucoup de la standard à la standard. Dans les systèmes de communications sans fil, le spectre radioélectrique, les technologies d'accès radio et des piles de protocoles change leur comportement pour les différents systèmes et réseaux. Aujourd'hui, il existe plusieurs standards pour les réseaux cellulaires (GSM, EDGE, WCDMA, etc), et réseaux locaux sans fil (IEEE 802.11a, b, g). Chacun de ces standards a différentes fréquences, largeurs de bande et schémas de modulation. Comme de nouvelles standards et applications ne cessent d'augmenter, il ya un besoin pour une plate-forme matérielle flexible qui est capable de supporter toutes les différentes standards dans toute la gamme de fréquence de communication sans fil. Software Defined Radio (SDR), est un système de communication radio reconfigurable qui peut être accordé à une bande de fréquences, et peut gérer tous les schémas de modulation dans un large gamme de fréquences; servant ainsi de multiples services et protocoles de communication.

L'utilisation d'une architecture flexible qui peut servir de plusieurs standards de communication sans fil offre de nombreux avantages, notamment:

- Une fois une architecture configurable est en place, il est prévu pour aider le déploiement rapide de nouvelles standard. En outre, il est fort possible que la conception actuelle peut calculer les nouveaux algorithmes sans aucune modification. Le coût d'entretien est également prévu de descendre par un facteur raisonnable.

- Les nouvelles tendances du marché conduisant à de nouvelles exigences de service et donc la conception et le développement de nouveaux systèmes ne partira pas de zéro, mais plutôt ces systèmes configurables aidera un développement plus rapide et les déploiements.

- L'effort de conception de SDR est d'aider les portables multi-radios à adopter dans les environs qui changent fréquemment. Cela signifie le concept de la radio cognitive ou opportuniste permettant d'utiliser le spectre plus efficacement. Le passage d'un protocole d'accès à l'autre devient transparent pour les utilisateurs.

- Une conception évolutive de radio ne serait pas seulement utile pour les terminaux, mais aussi des stations de base.

9.3 Contributions

Dans ce rapport de thèse, nous concentrons sur les options de conception différente de l'architecture numérique de bande de base dans le contexte des applications de SDR. L'architecture de bande de base numérique proposé est capable de mettre en oeuvre 2G, 3G, 4G, de communication et de diffusion des standards LAN sans fil utilisant les mêmes HW / SW architecture. Dans notre conception, le traitement en bande de base numérique est effectuée dans un HW / SW conception combiné capable de supporter toutes les exigences fonctionnelles à tout l'air-interface et à chaque étape du traitement des SDR, des niveaux les plus bas au plus petits. Le cloisonnement entre les HW et SW suit un coût général et de complexité par rapport à la vitesse de compromis. Le matériel est conçu de telle manière qu'il appuierait la tâche la plus efficace de calcul intensif à savoir répondre aux exigences de débit et de latence pour toutes les standards dans la conception. Le matériel est également suffisamment souple pour utiliser les mêmes ressources de traitement en bande de base pour les standards multiples. Le contrôle est en partie logiciel de la conception, qui passe les paramètres pertinents à du matériel pour des fonctionnalités spécifiques. Le défi dans la conception est de synchroniser tous les traitements à l'air-interface de manière efficace avec l'utilisation des ressources minimales et de grande précision. L'architecture matérielle de bande de base est divisé en deux parties principales: un module de haut niveau de contrôle et un traitement numérique du signal (DSP) du engine. Le module de contrôle est chargé de transférer les demandes MAC au DSP engine et la direction d'écoulement de contrôle des données. Le DSP engine est responsable de tous traitement du signal up-link/downlink. L'architecture de bande de base ainsi que son approche de conception est décrite en détail dans ce travail. Pour répondre aux besoins des dispositifs multi-standard sans fil, les diverses tâches du DSP engine de la gamme de conception de bande de base du taux d'échantillonnage correspondant à un décodage de Viterbi. Une étude approfondie de l'air cible interfaces conduit à l'identification d'un ensemble d'entités fonctionnelles pour le traitement en bande de base numérique. Les actions identifiées sont mises en oeuvre de sept blocs de traitement indépendants, et peut être appelé comme accélérateurs matériels. Il s'agit notamment de: Pré-processeur, Frontend Processeur, Mapper, Détecteur, Channel Encoder, Channel Decoder, et Interleaver / De-

Interleaver. La démarche de conception de chacun de ces blocs est de profiter des points communs qui existent entre les différentes standards à mettre en oeuvre. Les points communs et disjoints sont convertis en architecture matérielle de trouver un système qui assure toutes les opérations requises par toutes les applications. Le produit final permettra utilisateur d'effectuer régime désiré / fonctionnement standard en fournissant les paramètres sans entrer dans les détails de l'architecture. Le système proposé ne peut pas être aussi efficace que d'un seul système dédié d'une standard particulière, mais il serait certainement beaucoup plus efficace que d'avoir des blocs dédiés à chaque application. Le Front End Processor (FEP) à l'intérieur du bloc DSP engine est affectée à l'adresse de toutes les exigences au niveau de l'interface air qui comprennent l'estimation de canal, les données de détection, Channel Phase Offset (CPO) Estimation, et de synchronisation. Le mécanisme de ces fonctions diffèrent de l'air-interface à l'air-interface et les différentes interfaces utilisé pour les appareils de communication sans fil sont Orthogonal frequency-division multiplexing/multiple-access (OFDM/A), Single Carrier FDMA (SC-FDMA), Wideband Code Division Multiple Access (W-CDMA), et Space-division multiple access (SDMA). Un bloc flexible FEP de la conception de bande de base est capable d'effectuer les opérations mentionnées ci-dessus pour toutes les standards d'une manière efficace et invisible à dire à un utilisateur externe sans aucune obligation de changer de configuration matérielle.

La Fast Fourier Transform (FFT) a été utilisé comme blocs de construction pour des architectures de l'air-interface à la fois à l'émetteur et le récepteur. Au cours des dix dernières années, différentes architectures ont été proposées pour les récepteurs OFDM avec la FFT que le bloc de traitement touche [LL07] [CN06]. Similaires à OFDM, différentes architectures de calcul de domaine de fréquence pour le WCDMA / HSDPA ont été proposés avec des prestations tout à fait semblable au classique égaliseurs domaine temporel. Suivant la même méthode, FFT a été également utilisé pour MMSE turbo equalization dans Global System for Mobile Communications (GSM) [LLBL05]. Sur la base de ces contributions, pour les standards individuelles [YGC06], [LIMVS05], [LIZMP05], nous analysons et proposons un bloc de traitement dans le domaine fréquentiel capable de la restauration de toutes les opérations air-interface. Les implémentations de ces opérations sont généralement adaptés à la standard en question. L'approche de conception détaillée, la description fonctionnelle et la mise en oeuvre de l'câblée, FEP bloc paramétrable est discuté dans le rapport de thèse.

Les application specific integrated circuits (ASICs) sont plus optimisé et efficace en termes de superficie, la vitesse et la consommation d'énergie par rapport à d'autres méthodologies de conception. D'autre part, la souplesse offerte est à peu près rien, voire dans certains cas. L'Application Specific Instruction Set processeurs (ASIP) se présenter comme l'un des choix de conception de plate-forme flexible complexes pour gérer les tâches complexes de conception émetteur-récepteur sur le logiciel cible / plate-forme matérielle. L'ASIP également fournir une plus grande flexibilité qui est vraiment utile pour la conception de matériel pour servir les applications de communication sans fil en constante évolution. Pour la conception ASIP, si une grande flexibilité et de personnalisation de l'architecture de jeu d'instructions (instruction set architecture - ISA) est exigée, des outils qui mettent l'accent sur l'optimisation niveau de l'architecture peut être utilisée. Ces outils utilisent Architecture Description Langues (ADL). Il a été beaucoup de recherches dans ce domaine, mais jusqu'à présent, la langue LISA est la seule qui a gagné l'acceptation commerciale. Nous avons utilisé cette approche pour la conception d'un noyau ASIP appelé vecteur processeur (VP) pour évaluer son utilité dans les architectures de bande de base pour les applications de SDR. Les détails de conception ainsi que les résultats sont présentés dans ce document plus tard. Les deux approches de conception adoptée pour la FEP, ASIC et ASIP, sont analysés en détail dans ce rapport de thèse ainsi.

Dans le cadre de la conception de bande de base de SDR, l'ensemble des algorithmes à mettre en oeuvre le matériel proviennent de différents standards sans fil et de formes d'onde offrant ainsi un plus grand défi de conception. D'autre part, une puissance de traitement numérique de très haute est nécessaire pour mettre en oeuvre les solutions souples et efficaces pour les applications de SDR. Dans ce scénario, les performances du matériel numérique composant dans la conception de logiciels radio devient un aspect très important de mesurer la capacité de la radio. Les différents composants matériels qui peuvent être utilisés pour effectuer ces traitements numériques sont des processeurs de signaux numériques (Digital Signal Processors - DSPs), prédiffusés programmables (field programmable gate arrays - FPGAs), processeurs à usage général (general-purpose processors - GPPs), et circuits intégrés spécifiques (application specific integrated circuits - ASIC). Les paramètres tels que le coût, la rapidité, la souplesse et la consommation d'énergie sont considérées pour chacune de ces technologies matérielles. Compte tenu de ces options technologie matérielle contre l'ensemble des algorithmes dans la conception de bande de base SDR avec les exigences de performance, des lignes directrices de conception sont proposées à la fin du rapport.

9.4 Open Air Interface

Dans cette section, la conception de bande de base de notre plateforme émetteur-récepteur multi-standard est présenté. La base de notre conception de l'architecture est expliqué ainsi que les choix de conception.

9.4.1 Les Choix De Conception

Dans une conception de système de communication numérique, la phase la plus important est de choisir la technologie cible, le niveau de partitionnement entre matériel et logiciel, l'identification à l'intérieur des sous-blocs, et l'interface entre les sous-blocs. Alors tout d'abord, les choix effectués pour la conception de bande de base sont énumérés, avec les raisons de ces choix.

- La conception de bande de base SDR devraient être transférables à des technologies différentes. L'objectif de l'architecture de bande de base est d'abord de trouver un outil de recherche ou une plate-forme prototype expérimental, et n'est pas destinée à la production massive. Par conséquent, la cible choisie est la technologie FPGA et pas ASIC. Ce choix est basé sur de réduit le cycle de conception, flexibilité, facilité d'utilisation et de réduire les coûts des FPGA. Pour les mêmes raisons les couches supérieures sont implémentées dans des logiciels uniquement et exécuté sur un PC hôte. Une fois validé cette architecture sera retravaillé et adapté à la technologie ASIC.

- Le choix d'une technologie cible spécifique, à savoir FPGA dans ce cas, les contraintes du design légèrement. Le dessinateur est censé prendre en compte les blocs de mémoire spécifiques et les tranches de DSP qui viennent avec la technologie spécifiée. Ainsi, la conception est sous-optimal dans une certaine mesure dans le contexte mondial, et les résultats de synthèse doit être utilisé avec une approche prudente. Compte tenu de ces faits, tous les modules à l'intérieur du conception sont parfaitement synthétisable avec toutes les technologies existantes mais optimisé pour un particulier, dans certains cas.

- L'architecture matérielle proposée est subdivisé en deux parties principales: un module de haut niveau de contrôle et une unité de traitement numérique du signal. La séparation du contrôle et de traitement non seulement de faciliter une conception plus simple, mais rend également le système évolutif pour l'arrivée de nouvelles normes ou de fonctionnalités. Les deux modules, le contrôle et le traitement, sont mises en oeuvre dans les FPGA Virtex-V de Xilinx. La figure 9.1 illustre la présentation de l'architecture du système.

- Le *FPGA d'interface et de contrôle* dans la figure 9.1 a besoin d'un processeur à usage général pour gérer tout l'intérieur de transformation et de la communication à travers des interfaces externes. Le processeur principal est également censé gérer la programmation en vue des tâches à des blocs de traitement. Depuis la spécification Open Air Interface est une conception open source, nous avons choisi SPARC - processeur LEON3. Tout autre usage général processeur 32-bit aurait été aussi utile que soit LEON3.

- Pour interconnecter les unités de traitement à l'intérieur de l'unité de traitement nécessite une interface générique et standardisé. Ceci peut être réalisé en utilisant l'un des bus standard / méthodologie, nous avons choisi Advanced VCI Compliant bar, plus de détails sont énumérés dans le chapitre 4.

- Les blocs de traitement à l'intérieur du *Processing Engine FPGA* unité ont un contrôleur local (micro controller) pour contrôler les transferts de données et le traitement des commandes à l'intérieur du bloc. Cela réduirait également la communication sur l'interconnexion crossbar. Le choix de toute ressource petites et moins limité suffirait à nos besoins et nous avons choisi 6502. Pour rendre la conception plus générique, tous les blocs matériels ont le même 8-bit 6502 microcontrôleur.

- La conception générique pour chacune des unités de l'intérieur de l'unité de traitement et l'interconnexion permettrait d'ajouter n'importe quel autre bloc ou unité de matériel facilement dans la conception de bande de base

9.4.2 L'unité de traitement du DSP - DSP Processing Unit

Une étude des interfaces air cible (mentionné dans le chapitre 3), et les systèmes de communication pour les applications multi-standard (voir le chapitre 2) conduit à l'identification d'un ensemble d'entités fonctionnelles pour le traitement en bande de base numérique. Les actions identifiées sont mises en oeuvre de sept blocs de traitement indépendants, et peut être

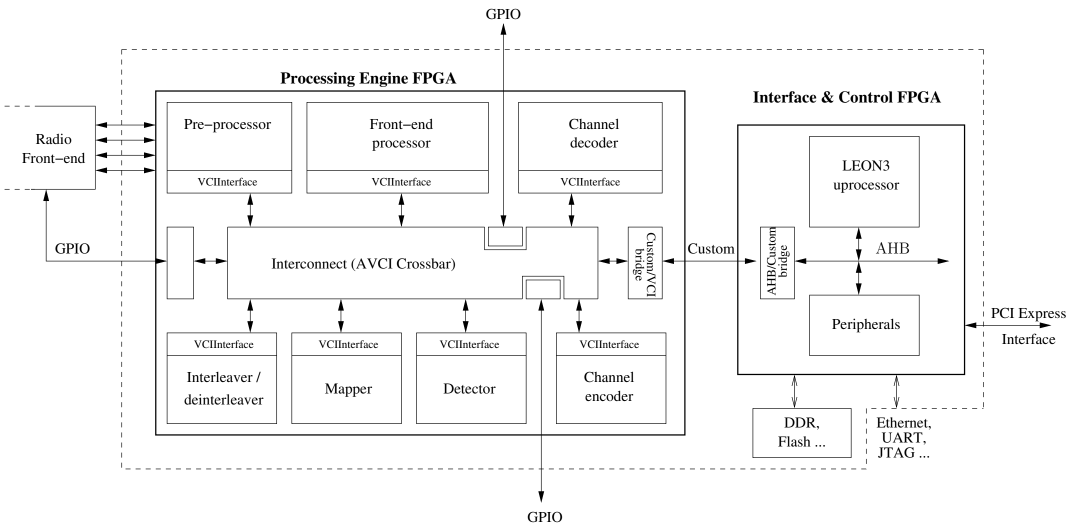

Figure 9.1: Prsentation de l'architecture de processeur de bande de base

appelé comme accélérateurs matériels (Hardware Accelerators). Ces blocs sont présentés dans la figure 9.1 et énumérés ici:

- Pre-processor
- Front End Processor
- Mapper
- Detector
- Channel Encoder
- Channel Decoder
- Interleaver / De-interleaver

Le bloc de pré-processeur est utilisé comme une interface avec l'externe A/D et D/A convertisseurs. Il fournit également plusieurs fonctionnalités de base de traitement du signal comme le filtrage, réglage du taux de l'échantillon etc. Le mappeur et le détecteur mettre en oeuvre tous les schémas de modulation allant de BPSK à QAM256. Le bloc interleaver fournit les données d'entrelacement avec toutes les options dans les différentes normes. Il effectue également de la péréquation cadre et les opérations taux d'appariement. Le Front-End-processeur fournit les opérations de traitement numérique du signal à l'interface aérienne, comme l'estimation de canal, détection de données etc. Le codeur implémente codage convolutionnel, turbo codage et de m- séquences. Le décodeur de canal réalise algorithmes de décodage en treillis à base; Viterbi et Turbo, à décoder les codes convolutifs et les turbo, respectivement. Dans cette thèse, la conception détaillée et de l'implémentation de Front-End-Processor est présenté.

9.5 Front End Processor (FEP)

L'enquête sur les algorithmes d'air-interface rend la base de la conception FEP. Le FEP est responsable de prendre soin des opérations air-interface, y compris l'estimation de canal, détection de données et de phase de porteuse de compensation pour la plate-forme de bande de base multiple standard. Les opérations suivantes sur les vecteurs sont définis pour ce bloc par la spécification fonctionnelle de la conception de bande de base (Les détails peuvent être trouvés dans le chapitre 3 de la thèse.

1. Discrete Fourier Transform (DFT), Inverse DFT (IDFT)
2. Les calculs de l'énergie - Energy Calculations
3. Les calculs de Maximale - Maximum, arg-max Calculations
4. Produit scalaire - Dot Product
5. L'addition (composant par composant) - Component-wise-addition
6. Soustraction - Component-wise-subtraction
7. Multiplication - Component-wise-product
8. Division - Component-wise-division

Ces fonctions sont sur vecteurs complexes avec une taille d'entrée allant de $\{1 \ldots 4096\}$. Donc, fondamentalement, il ya deux types d'opérations dans le bloc de FEP: la commutation entre domaine temporel et fréquentiel, et le traitement d'autres avant ou après le conversions de domaine (temps / fréquentiel). Ainsi, les opérations de bloc peut être divisé en deux grandes catégories: Conversion de Temps / Fréquence (FT Mode) et pré-post-traitement (PP Mode). Pre et Post se réfère ici à des opérations effectuées avant ou après la conversion de domaine. Les détails de la conception du FEP bloc sont fournis dans le chapitre 5 de thèses. La mise en oeuvre RTL est également expliquée lors de la conception de système de mémoire.

Utilisation de la Xilinx Virtex-5 FPGA que la technologie cible et Mentor Graphics' Precision comme outil de synthèse [men], $36 DSP48E$ sont utilisés. Cela fait 19% du total disponible dans le FPGA, ce qui est assez bon, compte tenu du fait que le FEP est l'un des plus IP calcul intensif du processeur en bande de base. La fréquence maximale possible pour les opérations de DFT est de $120 MHz$, et est tout à fait acceptable compte tenu du débit du bloc. Le nombre de cycles passé pour calculer les différentes tailles vecteur d'entrée de la DFT sont indiqués dans le tableau 9.1. La mise en oeuvre de la macrobloc de DFT est pipeline pour atteindre les plus haut débit et des débits de données éventuellement supérieur. Les valeurs plus élevées de cycles utilisés pour les tailles plus petites entrée est à cause de initialiazation de pipeline, qui a lieu pour toutes les valeurs d'entrée, mais devient important pour les tailles plus petites d'entrée. Toutefois, pour les valeurs les plus élevées, l'IP effectue tout à fait mieux que le débit prévu du 1-échantillons par cycle. Le débit des operations composante par composante au niveau de sousbande est 2-échantillons-par-cycle, et est réalisé dans la mise en oeuvre en conséquence. La fréquence maximaleréalisable de ces modules est d'environ $150 MHz$, et le nombre de multiplicateurs complexe qui est utilisée est de 10 [MKP09a].

Taille DFT	**Nombre de cycles**	**Taille DFT**	**Nombre de cycles**
8	20	16	18
32	46	64	60
128	107	256	174
512	372	1024	695
2048	1597	4096	3136

Table 9.1: DFT : Nombre de cycles utilisés pour différentes tailles de vecteur d'entrée

9.5.1 Limitations de l'architecture

Bien que le FEP est conforme aux spécifications fonctionnelles de sa conception et réalise également une bonne performace, il ya peu de limites à sa conception et qui sont énumérés ici:

L'opération de la matrice, l'algorithme de cache petites utilisé en mode

DFT pour le stockage des échantillons à l'entrée et de sortie, impose une condition sur la façon dont le vecteur d'entrée / sortie est écrit / lu dans la zone de mémoire par un transfert DMA ou par un accès direct à travers le VCIInterface: Les adresses (entrée et sortie) doit être un multiple de 8. Cela ne nécessite aucune exigence spécifique au niveau supérieur de contrôle, mais le logiciel doit prendre soin d'elle, tout en attribuant les adresses d'entrée et de sortie à la base FEP.

Les fonctions de FEP sont mutuellement exclusifs dire un seul ensemble de macro-blocs peuvent fonctionner en même temps. Par exemple, Dans l'estimation du canal de systèmes OFDM, le produit composante par composante ne peut commencer une fois que la DFT a pris fin. Cependant, cela a été prévu lors de la conception de la propriété intellectuelle et elle ne cause pas de dégradation des performances dans la bande de base grâce du débit élevé de processeur bande de base. Il convient également de mentionner qu'il n'y a pas de lag ou décalage entre le début d'une tâche à la fin de la tâche précédente. Les nouvelles tâches (commandes) peut toujours être écrit dans la période d'enquête et de l'espace mémoire peut être rempli pour tâche suivante, alors que la tâche actuelle est en cours.

9.6 Conception d'ASIP pour Vector Processor

Les nouveaux systèmes de communication numérique et les réseaux cellulaires nécessitent des opérations multi-mode avec une flexibilité accrue et des performances élevées. La surface de silicium et d'efficacité énergétique sont également des considérations très importantes pour la conception de l'architecture. Ces facteurs conduisent à la situation de compromis multiples de la cartographie des tâches complexes de conception émetteur-récepteur sur le cible logiciel / matérielle plate-forme. L'Application Specific Instruction processeurs (ASIP) sont l'un des choix de conception de plate-forme flexible complexes.

La conception ASIP est une tentative pour trouver un équilibre entre deux extrêmes: les ASIC et les généraux des processeurs programmables (GPP). ASIP offre la disponibilité des articles personnalisés pour le temps des tâches critiques (par exemple un Multipliez-Adder pour un DSP en temps réel), et offrent une flexibilité grâce à un ensemble d'instructions [LMP94]. Comme de nouvelles normes et applications continuent à arriver dans les communications sans fil, les opérations dans la conception de bande de base de SDR sont de plus en plus complexe. Ces procédures plus complexes exigent plus de flexibilité pour s'adapter aux changements de conception, des erreurs et des changements de spécifications, ce qui peut se produire au stade de la conception plus tard. Il est très difficile de faire beaucoup de changements dans l'ASIC, une fois la conception est en place. Dans une telle situation, l'ASIP offre la flexibilité nécessaire à moindre coût [CKY$^+$99].

Au début, l'ASIP et les outils pertinents de développement logiciel ont été conçus manuellement [JBK01]. La conception de processeurs manuelle est longue, fastidieuse, sujette aux erreurs et exige des ingénieurs hautement qualifiés. La conception de l'architecture du processeur peut être consultée dans les quatre parties [HML02]:

- L'exploration d'architecture
- L'implémentation d'architecture
- La conception d'applications de logiciels
- L'intégration et vérification du système

Sans automatisation, il est vraiment difficile d'avoir l'expertise dans tous ces procédures et donc une ASIP raisonnable. Le LISA Processeur Design Platform (LPDP), en utilisant language for instruction set architecture (LISA), prévoit un processus de traitement complet de conception qui s'appuie sur la description d'architecture cible dans un langage LISA [HKN$^+$01]. La LPDP offre la flexibilité d'un modèle de processeur de niveau le plus abstrait au niveau micro-architecture. La plate-forme utilise son générateur de HDL pour mettre en oeuvre l'architecture, une fois l'architecture micro est finalisé. Les modèles synthétisables peut être généré à la fois dans le VHDL et Verilog. Basé sur le succès de la LISA, comme la conception de processeurs automatisés, nous avons décidé d'explorer la conception ASIP pour la conception de bande de base multi-standard.

Dans le contexte d'architecture de Open Air Interface [ope], une analyse des fonctionnalités dans la conception de bande de base est effectuée afin d'identifier un ensemble d'opérations qui sont utilisés dans la bande de base émetteur-récepteur et êtes un bon candidat pour la conception ASIP. Il a été noté que les différents blocs nécessitent des opérations arithmétiques de base telles que l'addition, soustraction, multiplication, etc sur les vecteurs de grande taille. Ces opérations sont très souvent utilisés dans les différentes routines qui sont mises en oeuvre dans les accélérateurs matériels de notre conception de bande de base. Par exemple dans le bloc Front End Processor (FEP), les calculs d'énergie sur de grands vecteurs est réalisée en utilisant la multiplication, l'addition et la division sur les vecteurs complexes de grande taille.

Les routines dédié composé des opérations sur les vecteurs dans la conception de bande de base sont autonomes et ne servent qu'à l'application à l'intérieur du bloc spécifique. Pour prendre l'avantage de la souplesse que ASIP fournir et aussi d'étudier l'efficacité d'ASIP pour nos plates-formes dédiées à long terme, nous avons décidé de concevoir un processeur vectoriel comme ASIP. Comme point de départ, le processeur vectoriel doit être intégré dans le Front End Processor (FEP) de Open Air Interface plateforme. Le FEP utilise souvent les opérations arithmétiques de base sur la taille des vecteur important dans ses routines pour l'estimation de canal, détection de données etc., donc on commence à la conception dans le but que le projet de ASIP aurait au moins répondre à tous les besoins de l'application des routines FEP. Plus tard, compte tenu de la performance et l'analyse coût-bénéfice, le processeur vectoriel (VP) pourrait devenir un propriété intellectuelle autonome de l'Open Air Interface.

La formation actuelle de la FEP (conception ASIC) ne fait aucun doute souple et paramétrer mais il manque encore la flexibilité dans certains aspects. Nous essayons de déterminer où la flexibilité ou la programability peut être ajouté dans le bloc de matériel. Comme indiqué plus haut, les fonctions FEP sont composés de l'unité de DFT et les fonctions de base algébriques comme la multiplication, division, addition et la comparaison etc. à l'avenir, plus de fonctions pour répondre aux nouvelles exigences standard ou des changements dans certaines spécifications serait tout à fait un travail fastidieux . Ainsi, une conception ASIP qui réalise les fonctions de base algébrique sur des vecteurs complexes (avec quelques options de contrôle) serait un bon candidat pour étudier la flexibilité et l'analyse des performances de conception des coûts dans le contexte des applications de SDR.

En ce qui concerne le bloc de FEP, il existe deux modes de calcul nommée: FT et PP. Le FT effectue le DFT et le IDFT pour des longueurs variables, tandis que le PP effectue des opérations sur de grandes vecteurs complexes par exemple les calculs de l'énergie et des produits vectoriels. Les exigences initiales fonctionnelle de la VP sont spécifiés pour effectuer les opérations de PP-Mode. L'actuel module DFT / IDFT de la FEP sera utilisée en conjonction avec l'ASIP à exécuter les macros spécifiés par le processeur LEON3. La formation ASIP est illustré à la figure 9.2. Comme indiqué, le VP sera une partie de la FEP dans l'architecture d'Open Air Interface. Par conséquent, il suit la conception générique IP qui est commun à toutes les IP dans le processeur de bande de base.

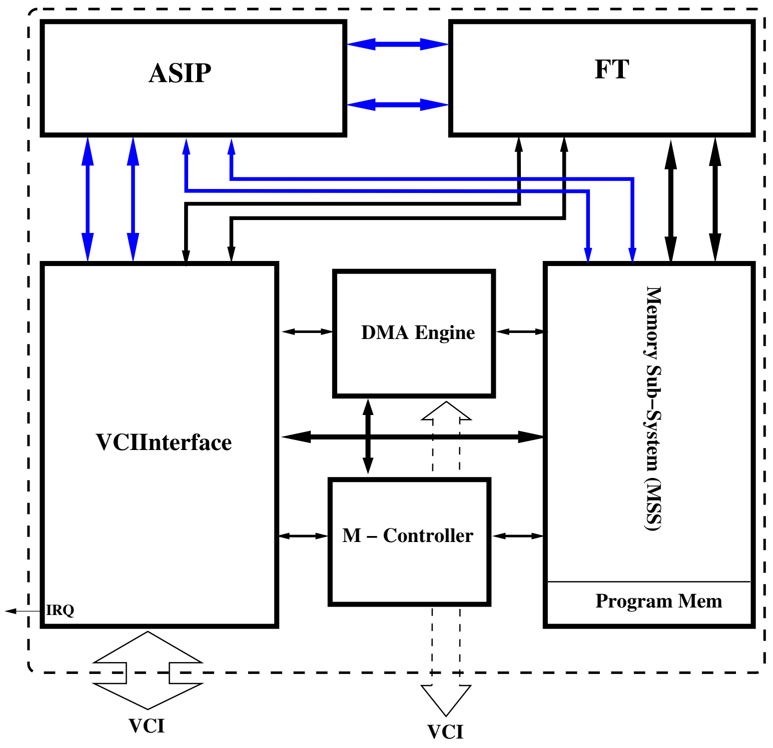

Figure 9.2: L'architecture ASIP avec FT

Dans la nouvelle formation avec le VP, les tâches plus haut niveau de la FEP, par exemple l'estimation de canal, sont d'abord traduites en fonctions comme component-wise-product sur vecteurs, les calculs de l'énergie, et FT etc. Suivant LEON3 décode / traduit ces différentes opérations (vector products, les calculs de l'énergie, etc.) dans les pré-définies routines de ASIP, et ces routines sont chargées dans la mémoire de programme (Program Memory -PM) de la VP. Par exemple, l'opération de produit scalaire (dot-product) de l'ASIP est traduit dans une routine qui nécessite multiplications complexes et additions complexes sur les vecteurs.

La formation ASIP non seulement répond aux exigences de la règlementation de la FEP, mais peut accueillir d'autres opérations dans notre conception globale, par exemple interpolation linéaire. L'ajout d'une fonction basée sur les fonctions algébriques voudrais juste besoin d'écrire les nouvelles bibliothèques dans le LEON3 qui peut être chargé dans l'ASIP.

La synthèse du code RTL généré par LISATeK est réalisée à l'aide de Mentor Graphics - Precision RTL Synthesis 2009a [men], et la technologie cible de notre bande de base de conception Virtex5 LX330-1760 [xil]. Les résultats sont résumés dans le tableau 9.2.

Resources	VP - Usage
Global Buffers	1
Function Generators	7493 (3.61 %)
CLB Slices	1874 (3.61 %)
Dffs or Latches	1394 (0.66 %)
DSP48E	8 (4.17 %)
Maximum Frequency	78.6 MHz

Table 9.2: Résumé des résultats d'implémentation d'ASIP

Les résultats au-delà de la fréquence maximale atteignable sont assez encourageants. La conception ASIP fournit la flexibilité pour les opérations vectorielles par rapport à la FEP (précédemment conception ASIC), et aussi que des opérations supplémentaires telles que Vector Shift et Vector Minimum sont pris en charge par le VP. Compte tenu de ces deux faits, l'utilisation des ressources est assez bon pour examiner les conceptions de la propriété intellectuelle ASIP développement futur sur la plate-forme de SDR. Le nombre de tranches DSP utilisées sont par l'attente, comme il ya 8 multiplications réelles prenant place dans la phase d'exécuter pipeline de la VP. Le temps de développement ASIP est également inférieur à la conception de l'ASIC et le temps de développement. Les deux approches de conception, l'ASIC et ASIP, et leurs résultats sont comparés en détail dans la section suivante.

9.7 Options de conception matériel pour le traitement en bande de base

La performance du partie matériel dans le Software Defined Radio (SDR) est un aspect clé pour mesurer la capacité de la radio. L'ensemble des algorithmes à mettre en oeuvre le matériel provient de diverses sans fil des normes et des formes d'onde offrant ainsi un plus grand défi de conception. D'autre part, une puissance de traitement numérique de très haute est nécessaire pour mettre en ouvre les solutions flexibles et effcient pour les applications de SDR. Les différents composants matériels qui peuvent être utilisés pour effectuer ces traitements numériques comprennent:

- processeurs de signaux numériques (DSPs)
- prédiffusés programmables (FPGAs)
- processeurs généralistes (GPPs)
- Application Specific Integrated Circuits (ASICs)

Considérant les exigences de performance du DTS et puis à analyser les différentes options disponibles pour trouver la meilleure solution est une tâche difficile système de conception et est discutée dans cette section.

Les GPP sont généralement conçus pour des solutions de haute performance de calcul d'effectuer des opérations de raisonnement pour la plupart fondées sur des cas ou des algorithmes de contrôle. GPP n'ont pas été utilisés en temps réel des tâches de traitement du signal en raison de leur piètre performance pour ces tâches. En cas de GPP, c'est difficile de prédire la durée d'exécution d'une tâche spécifique au niveau de la conception du système en raison des caches, des unités de prédiction de branchement, et multi-tâches. Une autre contrainte sur la GPP est leur forte consommation d'énergie par opération qui serait presque les exclure pour des applications de SDR étant donné le nombre d'opérations effectuées dans les plates-formes multi-standard. Toutefois, les systèmes à base de GPP pourraient être mieux utilisées dans des systèmes stationnaires où la consommation d'énergie est d'une importance peu moins. Aussi les patrons gradués fournir plus haut niveau de flexibilité, et sont capables de réaliser des tâches très variante sur une large gamme.

Les processeurs de signaux numériques (DSP) sont des microprocesseurs qui oeuvre efficacement des algorithmes de calcul à haute performance à l'aide spécialisée architectures réduisant ainsi le nombre de calculs pour une utilisation spécifique par rapport aux GPP. Les DSP sont utilisés pour les applications qui exigent l'exécution de meilleures performances que se trouve généralement sur les microprocesseurs standard ou GPP; les architectures de processeurs DSP fournir prise en charge optimisée pour la haute performance, répétitif, et numériquement intensive manipulation mathématique des signaux numériques [LBSL97]. Les processeurs DSP sont adaptées plusieurs opérations dédiées, tels que les multiplicateurs de matériel, des unités spécialisées de génération d'adresses, et accumulateurs grande. La DSP a également fourni des instructions spéciales disponibles pour les opérations DSP commun avec la capacité d'exécuter ces en parallèle, si nécessaire. D'autre part, les processeurs DSP deviennent en efficacité pour des tâches de contrôle irrégulier en raison de indisponible instructions spécialisées et la flexibilité est réduite ainsi. Les deux GPP et DSP ont de faibles performances pour les opérations peu complexes basés, comme c'est le cas dans l'entrelacement et de décodage de canal dans un design émetteur-récepteur.

Les FPGA sont des appareils gourmands en énergie par rapport à leurs concurrents (souvent plusieurs fois plus ASIC), qui rend difficile de les choisir comme solution pour la conception de bande de base. Bien que les FPGA offrent reconfiguration dynamique, mais son utilisation est limitée dans le contexte des plates-formes de SDR. Aussi dans presque tous les scénarios, une mise à jour de logiciels coûterait beaucoup moins cher que consommer un temps complexe reconfiguration du matériel dynamiques dans l'environnement multi-standard. Le FPGA aussi ne fournissent aucun bénéfice en termes de vitesse ou de fréquence la plus élevée possible dans la conception de bande de base numérique. Cependant pour le prototypage rapide d'une plate-forme expérimentale, comme [ope], les FPGA sont candidat idéal fondé sur le cycle de conception réduit, flexibilité, facilité d'utilisation et de réduire les coûts des FPGA. Une fois validé et une production à grande échelle de masse est nécessaire, il faut passer à des solutions telles que des ASIC ou DSP. Donc, pour aller sur l'analyse de la conception de bande de base, nous prenons à l'option FPGA.

ASIC offre la plus optimisée, puissant et efficace de calcul mise en oeuvre de matériel numérique pour les applications de traitement du signal au détriment de la souplesse. ASIC personnalisés général sont utilisés pour fournir la puissance de traitement a ajouté lorsque aucune autre option n'est disponible en raison des contraintes de conception ou lors de la conception des systèmes volume suffisamment élevé. Aussi la conception des ASICs sophistiqués nécessite un temps de développement et des efforts importants en matière de vérification. Par conséquent, les implémentations ASIC ont tendance à être mieux adaptés à des problèmes très complexes ou des applications à volume élevé ou des exigences élevées débit de données, tels que les téléphones cellulaires. D'autre part, à la suite la loi de Moore, les fabricants ont fait près de 60 % transistors plus disponibles par zone de silicium offrant chaque année un fort potentiel de calcul pour la conception ASIC hautement efficaces avec des débits de données élevés.

Les ASIC sont plus économes en énergie, suivie par la DSP et les processeurs à usage général (GPPs). La flexibilité se déplace dans la direction opposée à l'efficacité énergétique. Si des solutions logicielles sont adoptées seulement, puis en raison de la puissance de calcul limitée de chaque unité il y aurait un grand nombre de noeuds dans le système qui rend la conception d'interconnexion très complexe. D'autre part, la conception ne contenant que des ASIC aurait noeuds spécialisés dans le système et la conception d'interconnexion ne serait pas facile non plus. Avec les avantages et les inconvénients de la technologie du matériel énumérés ci-dessus, il est bien évident de réaliser que chaque modèle est unique et il n'y a pas de solution universelle de sélection parmi les dispositifs qui englobe toute la gamme d'algorithmes de bande de base. Pour obtenir les dernières normes sans fil, les quatre principales catégories de matériel numérique (GPP, DSP, FPGA, ASIC) n'apportent pas la capacité de calcul et nécessitent une combinaison de technologies pour la mise en oeuvre. Ces nouvelles normes sans fil ont besoin d'aide pour les ASIC à haut débit des fonctions simples et des logiciels DSP pour contrôler les fonctions complexes [Pul08]. En fait, la plupart des concepteurs utilisent une combinaison de dispositifs à mettre en oeuvre l'ensemble du système, une méthode souvent appelée traitement hétérogènes. Les arbitrages viennent d'image tout en faisant le choix du dispositif le plus efficace pour un certain ensemble d'algorithmes. Les paramètres tels que le coût, la rapidité et la flexibilité, ainsi que l'énergie et l'optimisation, toutes doivent être considérées. Ensuite, nous essayons de la carte l'ensemble des algorithmes pour différentes options de technologie matérielle.

Une fois que nous avons examiné les avantages et les inconvénients des technologies matérielles candidat pour la conception de bande de base, on passe à regarder les différentes opérations ou des algorithmes qui serait mis en uvre dans le cadre de SDR. La conception de bande de base de la radio réalisée par logiciel (SDR) est la composition ensemble d'algorithmes de normes différentes, par exemple combinant tous les canaux de codage de toutes les normes et en essayant de les fusionner en un seul appareil qui sert à toutes les normes. L'approche de conception de base est de prendre l'avantage des points communs qui existent entre les différentes normes, puis les traduire les blocs de traitement du matériel. Une fois l'ensemble des opérations et les algorithmes pour chaque ensemble de tâches est définie par exemple décodeur de canal, entrelacement ou mappeur, la décision suivante consiste à choisir la technologie du matériel pour l'unité a dit. Les paramètres les plus importants dans le choix de la technologie du matériel dans le cadre de la conception de bande de base multi-standard sont:

- Régularité de fonctionnement
- Exigences de traitement
- Homogénéité des opérations dans normes différentes

ASIC, méthodologie de conception, est le plus efficace par la puissance mais en même temps exige des algorithmes d'être très régulièrement pour profiter de conception complet. Parce que la consommation d'énergie est la plus faible, si idéalement la plus grande part des opérations devraient être conçus avec les ASIC, compte tenu des ressources de puissance disponible pour la conception de bande de base. Les opérations qui sont très régulières, exigent d'énormes puissance de traitement, et sont très homogènes à travers les normes deviennent un bon candidat pour la conception ASIC. Les opérations comme des transformées de Fourier (FTs), et filtrage numérique sont des candidats idéaux pour la conception ASIC.

Le choix de la DSP est bon pour les opérations qui sont moins régulières mais pas en situation irrégulière, et ont besoin de traitement dédié spécialisées telles que des multiplications et MAC. Les opérations régulières comme FT ou processeurs vectoriels conçu en utilisant le DSP se traduirait également par la conception acceptable ou modérée. Ainsi, pour l'exploitation spécialisée (par exemple la largeur des données spécifiques) et moins critique opérations régulières avec les exigences de traitement moyen et la consommation d'énergie, les DSP sont bon candidat dans le traitement en bande de base.

Les GPP sont les plus flexibles et peuvent servir énormément tâches variante. Les opérations en situation irrégulière devraient également être menées par GPP, mais il convient de mentionner à nouveau que la consommation est élevée dans GPP ainsi que des opérations limitées devraient être affectés à la GPP dans une conception hybride. Les opérations de contrôle dans la conception de bande de base peut généralement être affecté à la GPP.

Ainsi, la cession d'un ensemble spécifique d'algorithmes pour une technologie matérielle dépend de sa nature de calcul, fréquence d'occurrence et son comportement à travers de multiples normes. Vient ensuite la question, combien de fois peut-on attribuer les algorithmes à une catégorie, par exemple tous les algorithmes peuvent être effectuées avec la DSP. Il peut être répondu en regardant le nombre d'opérations par unité de temps (seconde) nécessaire à la conception finale et les ressources énergétiques disponibles par seconde pour la partie bande de base dans le produit. Nous illustrons ceci par un exemple et utiliser les nombres donnés dans [vB09].

Les terminaux mobiles ont aujourd'hui 3 watts de puissance disponible, à partir de laquelle $1W$ est disponible pour le traitement en bande de base numérique de l'appareil de poche. D'autre part, la charge de travail numérique est $100GOPS$ $((10)^9$ d'opérations par seconde). Ainsi, nous avons un budget de puissance de $10 - pico - J/operation$. Maintenant, si nous devons concevoir la bande de base avec les ASIC et DSP, nous serions alors en utilisant une combinaison des deux, étant donné que le budget de puissance se situe entre la consommation d'énergie de ces deux technologies matérielles. Si la consommation moyenne de ASIC est $2pJ/op$ et pour les DSP, il est $20pJ/op$, puis 55% des opérations peut être affecté à ASIC, tandis que 45% peuvent être affectés à DSP. Cet exemple illustre le fait que non seulement la nature mais le fonctionnement du budget de puissance doivent également être inclus en tant que paramètre lors de l'allocation d'un ensemble d'algorithmes pour une technologie matérielle spécifique.

9.8 Comparaison des approches de conception (ASIC - ASIP)

Le travail de recherche a présenté deux modèles de matériel, le Front End Processor (FEP) et le Vector Processor (VP). La conception FEP est une conception câblée soit une conception ASIC, tandis que la conception VP est un ASIP une. Dans cette section, nous présentons la comparaison de ces deux modèles sur la base de nos observations et les résultats.

Les spécifications fonctionnelles de l'ASIP ont d'abord été réglé pour correspondre à celle de la FEP à l'exception de la Fourier Transform (FT), à savoir l'ASIP est censé fournir toutes les fonctions que la FEP a en dehors de la DFT / IDFT. Donc, fondamentalement, ASIP est conçu pour l'arithmétique opérations composante par composante sur les vecteurs tels que l'addition, soustraction, multiplication et division. Ensuite, il existe des fonctions telles que les calculs de l'énergie, Dot-Produits et calculs de Max sur les vecteurs ou sous-vecteurs. La méthodologie de conception de l'ASIP pour servir l'ensemble de ces opérations est différente de l'approche FEP. Le jeu d'instructions ASIP est composé de la simple opération sur les vecteurs et composé de deux ou trois instructions pour effectuer une opération sur bits complexes tels que les calculs de l'énergie ou de produits scalaires. Pour illustrer par un exemple, pour calculer l'énergie moyenne d'un vecteur, le VP serait de le faire en trois étapes: calculer le carré absolu des différents éléments du vecteur, alors la somme vectorielle en ajoutant tous les éléments calculés dans la précédente étape et enfin l'aide de la division par la taille du vecteur. Pour la performance de l'ASIP à l'égard de la conception ASIC, nous avons les observations suivantes:

- La VP conception offre une plus grande programmabilité utilisant les opérations au niveau macro, que ces opérations peuvent être utilisés dans différentes formations avec de nombreuses options.
- Le throughput de les opérations composante par composante est est la même dans les deux cas.
- Le throughput de VP est la moitié pour quelques opérations de FEP comme Dot-produit et les calculs de l'énergie parce que VP se décompose ces opérations en opérations vectorielles simple (ASIP Instruction Set).
- VP donne une grande flexibilité opération vectorielle au détriment du plus faible throughput pour quelques fonctions.
- L'unité de génération d'adresse (Address Generation Unit - AGU) de la VP est plus souple, sans perte de performance. Dans le VP, les adresses vecteur de la mémoire peut soit suivre le modèle d'entrée ou peuvent être stockées à des adresses contigus.
- Le VP fournit quelques opérations arithmétiques de base supplémentaires, par exemple multiplication de Vecteur par des constantes, vecteur de décalage, Vector Square; améliorant ainsi la fonctionnalité de vecteur de la conception de bande de base.
- L'utilisation de la mémoire pour les deux modèles est la même en dehors de l'ajout de 64 Kbits de mémoire de programme pour la conception ASIP.

La tâche suivante consiste à comparer ces deux modèles sur les technologies cibles. Pour faire une comparaison équitable, le bloc de traitement FEP est resynthétisé sans son unité de FT et les conceptions sont synthétisés avec Xilinx Virtex V et aussi avec 65 nm comme les technologies ciblées.

Les résultats de synthèse en utilisant les technologies cibles sont indiqués dans le tableau 9.3. La fréquence cible paramètre est la contrainte de synthèse transmis à la outil de synthèse, tandis que la fréquence maximale est de ce que réalise la conception à la fin de la synthèse. La surface de silicium dans le tableau 9.3 est représenté par générateur de fonction (Function Generator - FG). Dans le tableau des résultats, le paramètre F / S (maximum fréquence / surface silicium) dans le tableau est un des critères de qualité classique: la plus la meilleure. Pour obtenir les meilleurs résultats, nous avons utilisé un agressifs synthèse approche. Premier temps, nous fixer un objectif très élevé de fréquence à observer la fréquence maximale atteignable par la technologie sans cible d'autres contraintes. Une fois la fréquence la plus élevée possible est connu alors une fréquence à la surface de silicium moins proche de la fréquence la plus élevée observée est recherché, comme le démontre les résultats dans le tableau 9.3. Pour la première synthèse avec Virtex-V, le nombre de tranches DSP48E extrême utilisée par chacun des bloc matériel étaient différents. Par conséquent, nous avons désactivé l'option d'utiliser les DSP extrêmes de faire une comparaison équitable, et les résultats indiqués sont sans en utilisant DSP.

Selon la comparaison des résultats, la conception ASIC est mieux que la conception ASIP en termes de superficie et de la fréquence. La surface de silicium augmente de 19% en cas de 65nm cible, tandis que l'augmentation cas de la synthèse FPGA est de 70%. Nous n'avons pas pu trouver une raison précise pour l'augmentation considérable des cas de la synthèse FPGA Virtex-V, une analyse en profondeur est nécessaire pour ce résultat spécifique. La diminution de la fréquence maximale atteignable est presque 70% dans les deux résultats de la synthèse par ASIP que ASIC. La différence de la fréquence maximale atteignable n'était pas prévu et les premiers enquête suggèrent que, avec quelques modifications mineures dans la conception LISA, une fréquence acceptable peut être atteint. La différence n'est pas énorme en cas de la surface de silicium, en ignorant les résultats FPGA, et ASIP peut être choisi et de le remplacer dans la conception de bande de base pour le traitement vectoriel, une fois le maximum fréquence réalisable est acceptable.

Une fois la conception ASIP est retravaillé, nous pensons que les performances des deux options de conception serait comparable en termes de surface de silicium et de la fréquence. Bien que les résultats de la conception

Target Technology	**Design**	**Target clock period (ps)**	**Target freq (MHz)**	**Silicon area** μm^2/**FG**	**Slack (ps)**	**Max Frequency (MHz)**	**100 * F / S**	**Normalized F/S**
ASIC 65 nm	**VP**	3300.00	303.03	107968	1585.00	204.71	1.90	51.27
		5400.00	185.19	97987	114.00	181.36	1.85	50.05
		5430.00	184.16	99497	0.00	184.16	1.85	50.05
	FEP	2000.00	500.00	91686	1095.00	323.10	3.52	95.29
		3095.00	323.10	93495	0.00	323.10	3.46	93.45
		3291.00	303.86	82166	0.00	303.86	3.70	100.00
		3350.00	298.51	82168	0.00	298.51	3.63	98.24
Virtex V 5VLX330	**VP**	3333.33	300.00	9737	9377.67	78.67	8.08	33.92
		13157.89	76.00	9665	282.91	77.67	8.04	33.74
	FEP	3333.33	300.00	5683	4054.65	135.36	23.82	100.00
		4000.00	250.00	5709	3387.98	135.36	23.71	99.54
		5000.00	200.00	5709	2387.98	135.36	23.71	99.54
		7692.31	130.00	5707	304.33	135.36	23.72	99.58

Table 9.3: Le résumé des résultats de synthèse pour la conception de FEP et VP

ASIC serait encore un peu mieux, mais la conception ASIP fournit un degré plus élevé de flexibilité et une fonctionnalité accrue, les paramètres très importants dans flexible radio design. Un autre aspect que nous voudrions étudier à l'avenir est la consommation électrique des deux conceptions.

Les dessins ASIP comme le DSP et d'autres solutions logicielles se présenter comme une alternative à la conception de bande de base lorsque la complexité augmente. Notre effort a été d'étudier la pertinence de l'ASIP dans la conception hybride plate-forme pour les applications de SDR. Notre conception de bande de base [OPE], décrit ci-dessus, est composé de l'ASIC paramétrable.

Le premier effort avec le prototype est d'étudier que la manière dont jusqu'où nous pouvons aller à la conception ASIC garder suffisamment de flexibilité et de silicium acceptable. La raison de la conception ASIC sont les mêmes que celles expliquées dans la section précédente: grande puissance de calcul et une faible consommation de puissance. Les algorithmes tels que codeur et le décodeur (Channel Encoding and Channel Decoding) qui ne sont ni régulières ni homogène dans l'ensemble les normes ont posé des problèmes assez pour la conception ASIC. Les solutions ASIC pour ces blocs complexes dans le manque de souplesse de conception de bande de base et nécessitent des performances élevées. Le pourcentage de réutilisabilité de logique dans la conception ASIC de ces blocs n'est pas encourageant, par exemple dans LDPC et Viterbi décodage. La meilleure solution nécessite également plus élevé et le programability des solutions de rechange telles que les patrons gradués et DSP (discuté dans la section précédente) n'aident pas à ce scénario. Les chercheurs ont déjà envisagé l'ASIP comme une solution pour décodeur solution dans un environnement multi-standard avec des résultats acceptables [VW08] [KMF09] [BAS04]. L'ASIP peut être une solution pour algorithmes complexe tels que codeur et le décodeur flexible et de se présenter comme un candidat solide pour l'exploration d'un tel ensemble d'algorithmes.

9.9 Conclusions et travaux futurs

La thèse présentée a démontré une approche pratique pour la mise en oeuvre des architectures radio en bande de base pour les plates-formes souples. L'architecture multi bande de base standard pose défis de conception qui incluent, mais ne se limitent pas à des données diverses débit, la latence et des contraintes de temps. La conception de bande de base ne doit pas seulement répondre aux différentes standards ou spécifications, mais l'ensemble varié de fonctions / opérations pour chaque application sans fil. Le processeur numérique avant la fin (FEP) prend en charge des opérations à l'interface air dans la conception de bande de base. Nous avons présenté les spécifications fonctionnelles, la conception du matériel, et la mise en oeuvre de ce bloc en détail. Pour faire l'analyse du spectre plus large, nous avons également présenté la conception ASIP d'un processeur vectoriel en utilisant un langage LISA. Le processeur vectoriel effectue les opérations arithmétiques de base sur les vecteurs complexes. Nous avons également discuté des approches de conception de matériel dans le cadre de la conception multi-standard et a proposé des recommandations pour les différentes catégories d'opérations.

Le travail de recherche est une tentative d'établir une base solide pour un design pratique d'une conception de bande de base multi-standard. Nous n'avons pas seulement examiné les options de conception, mais aussi assuré la conception et l'implémentation en utilisant les approches ASIC et ASIP. Le flexible, programmable dessins de bande de base multi-standard ont une portée considérable dans les dispositifs de communication sans fil dans les années à venir. Notre analyse, la conception et la mise en oeuvre peut être utile pour les chercheurs à explorer de nouveaux horizons dans les domaines de la conception de bande de base flexible pour des applications standard.

Comme la plupart des travaux de recherche, il ya des questions qui doivent encore être répondu et il ya des caractéristiques qui doivent être explorés dans le contexte du travail présenté. Certains d'entre eux sont énumérés ici:

- Comment une plus grande flexibilité peut être ajouté à la conception actuelle de la FEP. La conception actuelle du bloc FEP soutient une seule opération à la fois, et la prochaine tâche doit attendre l'achèvement de la tâche en cours. Le bloc FEP doit fournir une fonction pour effectuer des opérations multiples dans le même temps, ou au moins la commutation entre les opérations, si une tâche prioritaire est attribué par le système.

- Bien que la conception ASIP a été comparée avec la solution câblée RTL, mais il ya encore beaucoup de place pour une analyse plus rigoureuse. Les deux unités doivent être comparés à l'analyse de la consommation électrique avant de faire un dernier appel pour l'utilisation des unités de matériel dans la conception de bande de base.

- L'intégration de la FEP et la conception ASIP en même temps dans la plate-forme Open Air Interface et l'évaluation des résultats en temps réel serait également un réel plus à avoir.

- Nous avons recommandé que l'aide d'une structure hybride pour la conception de bande de base flexible. Il serait vraiment intéressant d'avoir une conception ASIP pour l'unité de décodage canal, par exemple, dans la conception de bande de base et avoir une évaluation de la performance. Ce serait donner des résultats concrets à l'appui de notre hypothèse.

Bibliography

[Ala98] S. Alamouti, "A simple transmit diversity technique for wireless communications," *Selected Areas in Communications, IEEE Journal on*, vol. 16, no. 8, pp. 1451 –1458, oct. 1998.

[alt] Altera. [Online]. Available: http://www.altera.com/

[BAS04] A. Blaickner, S. Albl, and W. Scherr, "Configurable computing architectures for wireless and software defined radio - a FPGA prototyping experience using high level design-tool-chains," in *International Symposium on System-on-Chip*, 2004, pp. 111 – 116.

[BCT99] C. Bergstrom, S. Chuprun, and D. Torrieri, "Adaptive spectrum exploitation using emerging software defined radios," in *IEEE Radio and Wireless Conference, RAWCON*, 1999, pp. 113 –116.

[BDBM$^+$04] M. Bocchi, C. De Bartolomeis, C. Mucci, F. Campi, A. Lodi, M. Toma, R. Canegallo, and R. Guerrieri, "A XiRisc-based SoC for embedded DSP applications," in *Proceedings of the IEEE Custom Integrated Circuits Conference,*, Oct. 2004, pp. 595 – 598.

[BHFN02] H. Blume, H. Hubert, H. Feldkamper, and T. Noll, "Model-based exploration of the design space for heterogeneous systems on chip," in *The IEEE International Conference on Application-Specific Systems, Architectures and Processors,*, 2002, pp. 29 – 40.

[BMH$^+$06] R. Bagheri, A. Mirzaei, M. Heidari, S. Chehrazi, M. Lee, M. Mikhemar, W. Tang, and A. Abidi, "Software-defined radio receiver: dream to reality," *IEEE Communications Magazine*, vol. 44, no. 8, pp. 111 –118, Aug. 2006.

[Bon00] A. B. Bondi, “Characteristics of scalability and their impact on performance,” in *Proceedings of the 2nd international workshop on Software and performance, Ottawa, Ontario, Canada*, 2000, pp. 195 – 203.

[Bra83] R. Bracewell, “Discrete hartley transfrom,” *J. Opt. Soc. Am.*, vol. 73, pp. 1832–1835, 1983.

[CEPA02] S. Colieri, M. Ergen, A. Puri, and B. A, “A study of channel estimation in OFDM systems,” in *IEEE 56th Vehicular Technology Conference, VTC Fall*, vol. 2, 2002, pp. 894 – 898 vol.2.

[CEPB02] S. Coleri, M. Ergen, A. Puri, and A. Bahai, “Channel estimation techniques based on pilot arrangement in OFDM systems,” *IEEE Transactions on Broadcasting*, vol. 48, no. 3, pp. 223 – 229, Sep. 2002.

[CKY$^+$99] H. Choi, J.-S. Kim, C.-W. Yoon, I.-C. Park, S. H. Hwang, and C.-M. Kyung, “Synthesis of application specific instructions for embedded DSP software,” *IEEE Transactions on Computers*, vol. 48, no. 6, pp. 603 –614, Jun. 1999.

[CN06] W.-H. Chang and T. Nguyen, “An OFDM-specified lossless FFT architecture,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 6, pp. 1235 –1243, June 2006.

[Coh76] D. Cohen, “Simplified control of FFT hardware,” *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 24, no. 6, pp. 577 – 579, Dec. 1976.

[CRS$^+$04] G. Cichon, P. Robelly, H. Seidel, E. Matus, M. Bronzel, and G. Fettweis, “Synchronous transfer architecture (STA)s,” in *In proceedings of International Symposium on Systems, Architectures, Modeling and Simulation (SAMOS) IV Workshop, Greece*, Jul. 2004.

[CT65] J. Cooley and J. Tukey, “An algorithm for the machine computation of the complex fourier series,” *Math. Computat.*, vol. 19, pp. 297–301, 1965.

[CVSI06] A. Cortes, I. Velez, J. Sevillano, and A. Irizar, “AFORE: an IFFT/FFT core generation tool different wireless communication standards,” in *International Conference on Consumer Electronics*, Jan. 2006, pp. 193 – 194.

[DGW+03] L. Davis, D. Garrett, G. Woodward, M. Bickerstaff, and F. Mullany, “System architecture and ASICs for a MIMO 3GPP-HSDPA receiver,” in *The 57th IEEE Semiannual Vehicular Technology Conference VTC-Spring*, vol. 2, Apr. 2003, pp. 818 – 822 vol.2.

[eCo] eCos. Open Source Real Time Operating System. [Online]. Available: http://ecos.sourceware.org/

[EFX+04] C. Ebeling, C. Fisher, G. Xing, M. Shen, and H. Liu, “Implementing an OFDM receiver on the RaPiD reconfigurable architecture,” *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1436 – 1448, Nov. 2004.

[GIL+03] J. Glossner, D. Iancu, J. Lu, E. Hokenek, and M. Moudgill, “A software-defined communications baseband design,” *IEEE Communications Magazine*, vol. 41, no. 1, pp. 120 – 128, Jan. 2003.

[GK10] R. Ghaffar and R. Knopp, “Low complexity metrics for bicm siso and mimo systems,” in *Vehicular Technology Conference (VTC 2010-Spring), 2010 IEEE 71st*, may. 2010, pp. 1 –6.

[Gol05] A. Goldsmith, *Wireless Communications.* Cambridge University Press, Aug 2005.

[HCC04] C.-P. Hung, S.-G. Chen, and K.-L. Chen, “Design of an efficient variable-length FFT processor,” in *Proceedings of the International Symposium on Circuits and Systems*, vol. 2, May. 2004, pp. II – 833–6 Vol.2.

[HKN+01] A. Hoffmann, T. Kogel, A. Nohl, G. Braun, O. Schliebusch, O. Wahlen, A. Wieferink, and H. Meyr, “A novel methodology for the design of application-specific instruction-set processors (ASIPs) using a machine description language,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 11, pp. 1338 –1354, Nov. 2001.

[HML02] A. Hoffmann, H. Meyr, and R. Leupers, *Architecture Exploration for Embedded Processors with LISA*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[ice] Icera Semiconductors. [Online]. Available: http://www.icerasemi.com/

[iee91] (1991) Ieee standard computer dictionary. a compilation of ieee standard computer glossaries,. [Online]. Available: http://ieeexplore.ieee.org/stamp/

[JBK01] M. Jain, M. Balakrishnan, and A. Kumar, “ASIP design methodologies: survey and issues,” in *Fourteenth International Conference on VLSI Design*, 2001, pp. 76 –81.

[KMF09] S. Kunze, E. Matus, and G. Fettweis, “ASIP decoder architecture for convolutional and LDPC codes,” in *IEEE International Symposium on Circuits and Systems, ISCAS*, 24-27 2009, pp. 2457 –2460.

[KMN02] K. Keutzer, S. Malik, and A. Newton, “From ASIC to ASIP: the next design discontinuity,” in *IEEE International Conference on Computer Design: VLSI in Computers and Processors,*, 2002, pp. 84 – 90.

[Kno05] S. Knowles, “The SoC future is soft,” in *IEE Cambridge Processor Seminar*, Dec. 2005.

[Kob01] B. Kobb, *Wireless Spectrum Finder*. McGraw Hill, NY, 2001.

[KP77] D. Kolba and T. Parks, “A prime factor FFT algorithm using high-speed convolution,” *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 25, no. 4, pp. 281 – 294, Aug 1977.

[KWD$^+$02] J. Kneip, M. Weiss, W. Drescher, J. V. Aue, M.Bolle, and G. Fettweis, “Hipersonic: Single-chip programmable baseband assp for 5 GHz wireless LAN applications,” *IEICE TRANS. ELECTRON*, vol. E85, no. 2, pp. 359 – 367, Feb. 2002.

[LBSL97] P. Lapsley, J. Bier, A. Shoham, and E. Lee, *DSP Processor Fundamentals (Architectures and Features)*. Wiley-IEEE Press, 1997.

[LIMVS05] D. Lo Iacono, E. Messina, C. Volpe, and A. Spalvieri, "Serial block processing for multi-code WCDMA frequency domain equalization," in *IEEE Wireless Communications and Networking Conference,*, vol. 1, 13-17 2005, pp. 164 – 170 Vol. 1.

[LIP] LIP6. Soclib modeling and simulation platform. [Online]. Available: http://soclib.lip6.fr/

[LIZM$^+$06] D. Lo Iacono, J. Zory, E. Messina, N. Piazzese, G. Saia, and A. Bettinelli, "ASIP architecture for multi-standard wireless terminals," in *Design, Automation and Test in Europe - DATE*, vol. 2, Mar. 2006, pp. 1 –6.

[LIZMP05] D. Lo Iacono, J. Zory, E. Messina, and N. Piazzese, "Block processing engine for high-throughput wireless communications," in *2nd International Symposium on Wireless Communication Systems,*, 5-7 2005, pp. 118 – 122.

[LL07] Y.-W. Lin and C.-Y. Lee, "Design of an FFT/IFFT processor for MIMO OFDM systems," *IEEE Transactions on Circuits and Systems I*, vol. 54, no. 4, pp. 807 –815, April 2007.

[LLBL05] C. Laot, R. Le Bidan, and D. Leroux, "Low-complexity MMSE turbo equalization: a possible solution for EDGE," *IEEE Transactions on Wireless Communications*, vol. 4, no. 3, pp. 965 – 974, May 2005.

[LLiCC06] J. Lee, H. Lee, S. in Cho, and S.-S. Choi, "A high-speed, low-complexity radix-2/sup 4/ FFT processor for MB-OFDM UWB systems," in *IEEE International Symposium on Circuits and Systems*, 2006.

[LMP94] C. Liem, T. May, and P. Paulin, "Instruction-set matching and selection for DSP and ASIP code generation," in *Proceedings of the European Design and Test Conference, EDAC*, Feb. 1994, pp. 31–37.

[LNPM05] Y.-C. Liang, S. Naveen, S. K. Pilakkat, and A. K. Marath, "Reconfigurable signal processing and hardware architecture for broadbandwireless communications," *EURASIP Journal on Wireless Communications and Networking*, pp. 323 – 332, 2005.

[LU95] R. Lackey and D. Upmal, "Speakeasy: the military software radio," *IEEE Communications Magazine*, vol. 33, no. 5, pp. 56 –61, May. 1995.

[LWB$^+$08] T. Limberg, M. Winter, M. Bimberg, R. Klemm, E. Matus, M. Tavares, G. Fettweis, H. Ahlendorf, and P. Robelly, "A fully programmable 40 GOPS SDR single chip baseband for LTE/WiMAX terminals," in *34th European Solid-State Circuits Conference, ESSCIRC,*, Sep. 2008, pp. 466 –469.

[Ma99] Y. Ma, "An effective memory addressing scheme for FFT processors," *IEEE Transactions on Signal Processing*, vol. 47, no. 3, pp. 907 –911, Mar. 1999.

[Mel02] J. Melby, "JTRS and the evolution toward software-defined radio," *MILCOM*, vol. 2, pp. 1286 – 1290, Oct. 2002.

[men] Mentor Graphics. [Online]. Available: http://www.mentor.com

[Mit95] J. Mitola, "The software radio architecture," *IEEE Communications Magazine*, vol. 33, no. 5, pp. 26 –38, May 1995.

[MKKP07] N.-I. Muhammad, K. Khalfallah, R. Knopp, and R. Pacalet, "Reconfigurable DSP architectures for sdr applications," in *14th IEEE International Conference on Electronics, Circuits and Systems, 2007. ICECS 2007.*, 11-14 2007, pp. 971 –974.

[MKP09a] N.-I. Muhammad, R. Knopp, and R. Pacalet, "On the hardware design of front-end processings in the SDR systems," in *SDR'09, Software Digital Radio, Technical Conference and Product Exposition,*, Dec 2009.

[MKP09b] ——, "Variable length DFT memory organization schemes for communication systems applications," in *12th SAME Forum,*, Sep. 2009.

[MRP$^+$08] N.-I. Muhammad, R. Rasheed, R. Pacalet, R. Knopp, and K. Khalfallah, "Flexible baseband architectures for future wireless systems," in *11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools, 2008. DSD '08.*, 3-5 2008, pp. 39 –46.

[MSL$^+$06] E. Matu, H. Seidel, T. Limberg, P. Robelly, and G. Fettweis, "A GFLOPS Vector-DSP for broadband wireless applications," in *IEEE Custom Integrated Circuits Conference, 2006. CICC '06.*, Sep. 2006, pp. 543 –546.

[MSM05] M. Morelli, L. Sanguinetti, and U. Mengali, "Channel estimation for adaptive frequency-domain equalization," *Wireless Communications, IEEE Transactions on*, vol. 4, no. 5, pp. 2508 – 2518, sep. 2005.

[MW00] Y. Ma and L. Wanhammar, "A hardware efficient control of memory addressing for high-performance FFT processors," *IEEE Transactions on Signal Processing*, vol. 48, no. 3, pp. 917 –921, Mar. 2000.

[ope] Open air interface. [Online]. Available: http://www.openairinterface.org/

[OS89] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1989.

[Pro95] J. G. Proakis, *Digital Communications.* McGraw Hill Book Co., Singapore, 1995.

[Pul08] D. Pulley, "Multi-core DSP for base stations: Large and small," in *Asia and South Pacific Design Automation Conference, ASPDAC*, Mar. 2008, pp. 389 –391.

[RS04] G. Rauwerda and G. Smit, "Implementation of a flexible RAKE receiver in heterogeneous reconfigurable hardware," in *IEEE International Conference on Field-Programmable Technology*, Dec. 2004, pp. 437 – 440.

[san] Sandbridge technologies. [Online]. Available: http://www.sandbridgetech.com/

[SIH$^+$91] J. Sato, M. Imai, T. Hakata, A. Alomary, and N. Hikichi, "An integrated design environment for application specific integrated processor," in *IEEE International Conference on Computer Design: VLSI in Computers and Processors,*, Oct. 1991, pp. 414 –417.

[SKKP10] C. Schmidt-Knorreck, R. Knopp, and R. Pacalet, "Hardware optimized sample rate conversion for software defined radio," in *WSR - 6th Karlsruhe Workshop on Software Radios*, Mar. 2010.

[SLHC10] S.-L. Su, Y.-C. Lin, C.-C. Hsu, and G. Chuang, "A DFT-based channel estimation scheme for IEEE 802.16e OFDMA systems," in *The 12th International Conference on Advanced Communication Technology (ICACT)*, vol. 1, Feb. 2010, pp. 775 –779.

[STB09] S. Sesia, I. Toufik, and M. Baker, *LTE, The UMTS Long Term Evolution: From Theory to Practice.* A John Wiley and Sons, Ltd, Publication, 2009.

[Tut99] W. Tuttlebee, "Software-defined radio: facets of a developing technology," *IEE Personal Communications*, vol. 6, no. 2, pp. 38 –44, Apr. 1999.

[UAB05] I. Uzun, A. Amira, and A. Bouridane, "FPGA implementations of fast Fourier transforms for real-time signal and image processing," *IEE Proceedings on Vision, Image and Signal Processing*, vol. 152, no. 3, pp. 283 – 296, Jun. 2005.

[vB09] C. van Berkel, "Multi-core for mobile phones," in *Design, Automation Test in Europe Conference Exhibition - DATE,*, Apr. 2009, pp. 1260 –1265.

[vBHM$^+$04] K. van Berkel, F. Heinle, P. Meuwissen, K. Moerman, and M. Weiss, "Vector processing as an enabler for software-defined radio in handsets from 3G+WLAN onwards," in *Software Defined Radio Technical Conference, Arizona - USA*, Nov 2004.

[vBHMKM05] K. van Berkel, F. Heinle, P. Meuwissen, and M. W. K. Moerman, "Vector processing as an enabler for software-defined radio in handheld devices," *EURASIP Journal on Applied Signal Processing*, vol. 16, pp. 2613 – 2632, 2005.

[VSI] VSIA. VSI Alliance. [Online]. Available: http://www.vsi.org/

[VW08] T. Vogt and N. Wehn, "A Reconfigurable ASIP for Convolutional and Turbo Decoding in an SDR Environment," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 10, pp. 1309 –1320, Oct. 2008.

[WF96] M. H. Weiss and G. P. Fettweis, "Dynamic codewidth reduction for VLIW instruction set architectures in digital signal processors," in *Proc. IWISP, Manchester*, 1996, pp. 517–520.

[WIF] Wireless innovation forum. [Online]. Available: http://www.wirelessinnovation.org

[Win86] S. Winograd, "On Computing the DFT," *Math. Computat.*, vol. 32, pp. 175–199, 1986.

[WVVW$^+$02] M. Wouters, G. Vanwijnsberghe, P. Van Wesemael, T. Huybrechts, and S. Thoen, "Real time implementation on FPGA of an OFDM based wireless LAN modem extended with adaptive loading," in *Proceedings of the 28th European Solid-State Circuits Conference,ESSCIRC*, Sep. 2002, pp. 531 – 534.

[xil] Xilinx. [Online]. Available: http://www.xilinx.com/

[YGC06] D. M. Y. Guo, J. Zhang and J. R. Cavallaro, "An efficient circulant MIMO Equalizer for CDMA Downlink: Algorithm and VLSI Architecture," *EURASIP Journal on Applied Signal Processing*, 2006.