ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2012-ENST-0057

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Eléctronique et Communications »

présentée et soutenue publiquement par

Carina SCHMIDT-KNORRECK

le 04.10.2012

Architectures Radio-Logicielles appliquées aux Réseaux Véhiculaires

Directeur de thèse : **Raymond KNOPP**

Jury

M. Jaques PALICOT, Professeur, SUPELEC, Cesson-Sévigné	Président
M. Andreas HERKERSDORF, Professeur, Technische Universitaet Muenchen, Allemagne	Rapporteur
M. Guido MASERA, Professeur, Politecnico di Torino, Italy	Rapporteur
M. Renaud PACALET, Directeur d'Etudes, Télécom ParisTech, Sophia Antipolis	Examinateur

TELECOM ParisTech
école de l'Institut Télécom - membre de ParisTech

T H È S E

To Daniel

Abstract

Today, wireless communication applications have become a major part in our life. Smartphones, for instance, support more and more applications allowing us to surf on the web or to talk to our friends at any place and at any time. New products compete by providing more applications in only one device that is smaller, lighter, cheaper and of higher performance than similar products on the market. Dealing with these requirements of reconfigurable radio architectures is a very challenging task. One solution can be found in the context of Software Defined Radio (SDR). Under its umbrella, flexible hardware platforms that support a wide range of different wireless communication standards are designed. The OpenAirInterface ExpressMIMO platform developed by Eurecom and Télécom ParisTech is a very flexible SDR platform whose baseband processing engine is split over different DSPs to enable a fast and easy component replacement and component upgrade.

The main objective of this thesis is to propose the first prototype of a receiver chain for the ExpressMIMO platform in general, and in particular to assess the applicability of the platform for latency critical standards. An interesting representative of such a standard is IEEE 802.11p. The work presented in this thesis is thus settled in the automotive context where efficient physical layer implementations of the IEEE 802.11p standard required for Car-to-Car and Car-to-Infrastructure communication are still an open research topic. The first contribution is a complete design of the IEEE 802.11p receiver implemented for the ExpressMIMO platform and was therefore serving as a proof of concept in general and in particular for standards operating on short data sets. Our results prove, that an efficient receiver design is feasible for various modulation schemes when applying a centralized control flow on the platform. Different design bottlenecks have been identified and solutions to overcome these limitations are suggested.

The combination of Car-to-Car and Car-to-Infrastructure communication with information about traffic jams or merchandising applications within only one device enables various new applications for future cars. Therefore we investigate on a possible multimodal execution of IEEE 802.11p and ETSI DAB. For the design of an appropriate scheduler it is of main importance to have first key figures at hand. We provide these figures based on a detailed runtime performance evaluation and enhance our obtained results by the derivation of scheduling guidelines and the presentation of a first scheduler prototype.

Our analysis reveals that the Front-End Processing (FEP) engine is heavily charged and that the required configuration time outreaches the pure executing time for short vectors when considering an FPGA target. To meet this challenge we introduce an Application Specific Instruction-set Processor (ASIP) as the solution of choice when dealing with strong latency requirements. The presented solution is not only compared to the programmable DSP engine but also to different solutions from academia. For design comparison we mainly focus on architectural differences and the runtime performance in terms of processing time.

To complete the receiver chain we finally present a first Preprocessor prototype. The Preprocessor connects the external A/D, D/A converters with the remaining baseband engine and is responsible

among others for I/Q imbalance correction and sample rate conversion. In this context we present a generic, flexible and hardware optimized Sample Rate Converter (SRC) operating on fractional ratios with a resolution of 1 Hz between the sampling frequencies. The design supports up to four different receive and up to four different transmit channels and is based on bandlimited interpolation.
Our results are finally generalized to ease the deployment of future standards on the ExpressMIMO platform.

Résumé

Aujourd'hui, les applications de communication sans fil font partie de notre vie quotidienne. Les smartphones, par exemple, supportent de plus en plus d'applications qui nous permettent de surfer sur le web et de parler à nos amis en tout lieu et à tout moment. Principalement, les nouveaux produits se concurrencent sur le nombre de leurs applications. Les autres facteurs determinants sont la taille, le poids et la performance qui sont comparés au produits concurrents qu'on trouve déjà sur le marché. Répondre aux contraintes des architectures reconfigurables n'est pas toujours une tâche aisée. Des solutions existent dans le domaine de la radio logicielle (Software Defined Radio, SDR) où des plateformes flexibles qui prennent en charge un large éventail de différentes standards de communication sans fil peuvent être conçus. La plateforme OpenAirInterface ExpressMIMO qui est développée par Eurecom et Télécom ParisTech est une plateforme radio logicielle très flexible: le traitement des operations dans la bande de base est réparti sur différents DSPs pour permettre un remplacement ou une mise à jour des composants simple et rapide.

L'objectif principal de cette thèse est de proposer le premier prototype d'un récepteur pour la plateforme ExpressMIMO, et d'évaluer le potentiel de la plateforme pour les standards ayant des latences critiques en particulier. Un cas intéressant d'un tel standard est la norme IEEE 802.11p qui spécifie la communication entre plusieurs véhicules (Car-to-Car communication) ainsi que la communication entre les véhicules et l'infrastructure (Car-to-Infrastructure communication). Le travail présenté dans cette thèse se focalise donc en partie sur le domaine de l'automobile où les implémentations efficaces de la couche physique du standard IEEE 802.11p est encore un sujet de recherche ouvert. La première contribution proposée est la conception complète d'un récepteur qui est basée sur le standard IEEE 802.11p et qui a été implémentée sur la plateforme ExpressMIMO. Ce system a donc servi de démonstrateur à la fois pour valider l'ensemble de la plateforme et en particulier l'utilisation de standards qui emploient des vecteurs de petite taille. Nos résultats prouvent qu'une conception efficace du récepteur est réalisable pour des schémas de modulation différents lorsqu'un un contrôle centralisé existe sur la plateforme. Des goulots d'étranglement ont pu être identifiés et des solutions ont été proposées pour surmonter ces limitations.

La combinaison dans un seul dispositif de la communication inter-véhicules avec l'information sur les embouteillages ou sur la proximité de commerces permet diverses applications nouvelles pour les futures automobiles. Par conséquent, nous étudions une possible exécution multimodal du IEEE 802.11p et du ETSI DAB. Pour la conception d'un ordonnanceur de tâches approprié, il est d'une importance principale àvoir un permier aperçu chiffré des performances. Nous fournissons des chiffres qui sont basés sur une évaluation détaillée des performances d'exploitation et nous utilisons ces résultats pour déduire l'ordonnancement optimal et nous présentons un premier prototype d'ordonnanceur.

Notre analyse, lors des expérimentations sur une cible FPGA, révèle que le Front-End Processing (FEP) DSP est lourdement chargé et que le temps de configuration requis dépasse le temps d'execution dans le cas d'operations sur des vecteurs de petite taille. Pour relever ce défi, nous proposons un Application Specific Instruction-set Processor (ASIP) comme solution lorsque les

contraintes de latence sont fortes. La solution présentée est non seulement comparée au DSP programmable, mais aussi à d'autres solutions du monde universitaire. Pour la comparaison des conceptions nous nous concentrons principalement sur les différences dans les architecture et la performance d'exécution en termes de temps de traitement.
Pour compléter la chaîne de réception, nous présentons enfin un premier prototype de Préprocesseur DSP. Le Préprocesseur connecte les convertisseurs A/D et D/A avec les autres composants de la bande de base. Il est également reponsable, entre autres, de la correction du déséquilibre entre les voies I et Q et du ré-échantillonnage. Dans ce contexte, nous présentons un convertisseur générique et flexible pour le ré-échantillonnage (Sample Rate Converter, SRC) qui travaille sur des rapports fractionnaires de fréquence d'échantillonnage avec une résolution de 1 Hz entre les fréquences. La conception prend en charge jusqu'à quatre chaînes différentes en transmission et en récpetion et est basée sur une interpolation à bande limitée.
Nos résultats sont finalement généralisés et nous montrons que notre approche facilite le déploiement de futures standards sur des plateformes tel que ExpressMIMO .

Acknowledgements

During the past years, I had the chance to work with and to meet a lot of fantastic people. All of you influenced my life and taught me a lot. Not only about telecommunications and engineering but also about life. My thanks go to all of you, because without you, I would not be who I am today!

First of all I would like to thank my supervisor Raymond Knopp for his support and for giving me the chance to become a part of his great team. It was a big pleasure for me to work with you!
Further thanks go to the whole LabSoC team. You guys are amazing! Thanks for all your support and your friendship throughout the past years. It was a pleasure for me to meet all of you and I hope that we can keep in touch :-)
At the end I spent six fantastic years at Eurecom. First as a student, then as a PhD student. Eurecom is an incredible place in France and I enjoyed so much meeting people from all over the world and to learn from them. I do not want to start writing names of the friends I made the past years, because I'm very afraid that I will miss someone. And I'm quite sure that you know who I mean.

Finally I would like to thank my dear husband Daniel. We started our french and even our PhD journey together and all the challenges of the past years made our relationship stronger and stronger. You are my love, my soulmate and the best friend I ever had! Thanks to your support, I was able to finish this work and I hope, that I was such a tower of strength for you than you were for me!

Thank you!

Table of 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.

A/D	Analog to Digital
ADL	Architecture Description Language
AGU	Address Generation Unit
ALU	Arithmetic Logic Unit
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction-set Processor
ASP	Application-Specific Processor
ASPE	Adaptive Stream Processing Engine
ASSP	Application-Specific Signal Product
AVO	Arithmetic Vector Operation
C2C	Car-to-Car
C2I	Car-to-Infrastructure
CHDEC	Channel Decoder
CIC	Cascaded Integrator Comb
CIF	Common Interleaved Frame
CISC	Complex Instruction-Set Computer
CSS	Control Sub-System
CWA	Component-Wise Addition
CWL	Component-Wise Look-Up Table
CWP	Component-Wise Product
CWS	Component-Wise Square
CWSM	Component-Wise Square of Modulus
D/A	Digital to Analog
DAB	Digital Audio Broadcasting
DDR	Double Data Rate
DDS	Direct Digital Synthesizer
DEINTL	Deinterleaver
DFT	Discrete Fourier Transform
DQPSK	Differential QPSK
DSP	Digital Signal Processor
DSRC	Dedicated Short Range Communication
DXP	Deep eXecution Processor
ETSI	European Telecommunications Standards Institute
FCC	Federal Communications Commission
FDD	Frequency Division Duplex

FEP	Front-End Processor
FFT	Fast Fourier Transform
FIB	Fast Information Block
FIC	Fast Information Channel
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
freeRTOS	free Real-Time Operating System
FU	Functional Unit
GP	General Purpose
GPP	General Purpose Processor
HDL	Hardware Description Language
HR	Hardware Radio
HW	Hardware
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse Fast Fourier Transform
IIR	Infinite Impulse Response
INTL	Interleaver
IPC	Interpolation Control
ISA	Instruction-Set Architecture
ISR	Ideal Software Radio
ISI	Inter Symbol Interference
ITS	Intelligent Transport System
LISA	Language for Instruction-Set Architectures
LTE	Long Term Evolution
LTS	Long Training Symbol
LUT	Look-up Table
MAC	Media Access Control
MIMD	Multiple Instruction Multiple Data
MIMO	Multiple Input Multiple Output
MIO	Input / Output Data Space (FEP)
MIPS	Millions of Instructions Per Second
MMSE	Minimum Mean Square Error
MOPS	Millions of Operations Per Second
MOV	Move
MPDU	MAC Protocol Data Unit
MSC	Main Service Channel
MSS	Memory Sub-System
NCO	Numerically Controlled Oscillator
NoC	Network on Chip
OFDM/A	Orthogonal Frequency Division Multiplexing / Multiple Access
OS	Operating System
PE	Processing Entity
PER	Packet Error Rate
PD	Pre-Distortion
PHY	Physical
PLL	Phase-Locked Loop
PM	Program Memory
PP	Preprocessor

PSD	Power Spectral Density
PU	Processing Unit
RF	Radio Frequency
RISC	Reduced Instruction-Set Computer
RTEMS	Real-Time Executive for Multiprocessor Systems
RTL	Register Transfer Level
RTV	Roadside To Vehicle
RX	Receiver Chain
SC	Synchronization Channel
SC-FDMA	Single Carrier Frequency Division Multiple Access
SCR	Software Controlled Radio
SDMA	Space Division Multiple Access
SDR	Software Defined Radio
SEQ	Sequencer Unit
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction stream Multiple Tasks
SNR	Signal to Noise Ratio
SISO	Single Input Single Output
SoC	Software on Chip
SRC	Sample Rate Converter
STA	Synchronous Transfer Architecture
STS	Short Training Symbol
SU	Storage Unit
SW	Software
TDD	Time Division Duplex
TMP	Temporary Memory
TTA	Task Triggered Architecture
TWD	Twiddle Factor Memory
TX	Transmitter Chain
UC	Microcontroller
UCM	Microcontroller Program Memory
USR	Ultimate Software Radio
USRP	Universal Software Radio Peripheral
VANET	Vehicular Ad Hoc Network
VCO	Voltage Controlled Oscillator
VLIW	Very Long Instruction Word
VECS	Vector Sum
VECSI	Vector Shift
VMM	Vector Max / Min
VTV	Vehicle to Vehicle
WAVE	Wireless Access in Vehicular Environments
W-CDMA	Wideband Code Division Multiple Access
WIF	Wireless Innovation Forum
WIM	Window Invalid Mask
WLAN	Wireless Local Area Network
WSU	Wireless Safety Unit

Chapter 1

Introduction

1.1 Motivation and Problem Statement

Today, wireless communication applications have become a major part in our life. Almost every day we check our emails either on smartphones or on personal computers via Wireless Local Area Network (WLAN) connections. Besides we communicate via our mobile phones or we consult navigation systems or online maps to find our way in case we have lost our bearings. Especially for the young generation it is impossible to imagine living in a world where they cannot be connected to their friends at any place and at any time. More and more companies have recognized this trend and seek to bring new products to the market that integrate more applications in only one device, that is smaller and lighter, that costs less and that has a higher performance than competing ones.

Another interesting market for wireless communication devices can be found in the automotive industry. It is a well known fact that the demographic change leads to a rising percentage of old people, especially in Europe. In countries like Germany where there is no age limit for car driving, there is a high need for new safety applications like speed measurements, obstacle warnings or distance measurements to the car driving in front. Two key terms in this context are Car-to-Car communication (C2C) and Car-to-Infrastructure (C2I) communication which also include the provision of non-safety applications like toll collection, tourist information or mobile internet. Standards of interest are IEEE 802.11p which is an enhancement of the well known IEEE 802.11a standard used for WLAN connections, and ETSI DAB (Digital Audio Broadcasting). To combine these two standards, two approaches are imaginable. Either they are implemented individually and come with their own receivers and transmitters that have to be integrated in the car, or both of them are combined in only one device. As it is the case for the mobile phone market, these devices should be small, cheap, of high performance and should be easily adaptable to future standards. Especially the latter is very time consuming and costly when integrating separate standard technologies in a car. Therefore, a single architecture that is capable to process whatever wireless communication standard is the preferable solution, especially as there is a high interest in combining IEEE 802.11p with LTE (3GPP Long Term Evolution) in vehicular system in the near future.
To deal with these increasing requirements for reconfigurable radio architectures is a very challenging task. One solution can be found in the context of Software Defined Radio (SDR). A major aim of SDR is to provide flexible platform solutions supporting a wide range of different wireless communication standards in a multimodal fashion. This approach does not only come with the advantage of a faster development and a faster deployment of new standards but also with the automatic adoption to the surroundings.

As initially stated, we exemplify the execution of latency critical standards on the ExpressMIMO platform by means of the IEEE 802.11p standard. We further investigate in the combination with a DAB receiver. As IEEE 802.11p has been in draft version till July 2010, efficient physical layer receiver implementations are still an open research topic. And to our knowledge, so far little effort has been spent on the description of a possible multimodal SDR platform processing of these two standards of interest.
The chosen target platform is the OpenAirInterface ExpressMIMO platform developed by Eurecom and Télécom ParisTech. In contrast to other SDR platforms, the baseband processing functions are split over several independent Digital Signal Processor (DSP) engines or hardware accelerators like Channel De/Encoder, (De)Interleaver or Front-End Processor (FEP) which can be executed in parallel. This enables not only a higher performance of the whole design but further allows an easy component replacement in case future updates become necessary. The platform is capable to process up to eight different channels simultaneously (four channels in transmission, four in reception) by reusing the existing programmable resources. Main design challenge is the synchronization of these resources by providing a maximum accuracy and by meeting all the real-time requirements. The platform can further be emulated with the Library for ExpressMIMO baseband, called *libembb*, enabling an easy receiver validation and verification in a pure software environment.
At the very beginning of this thesis, the work on this platform was still ongoing. The presented receiver is thus the very first complete design that has been developed and evaluated on this target platform and that was emulated with the help of libembb. It was therefore serving as a first proof of concept of the whole design. Executing standards that operate on very small vector lengths, like IEEE 802.11p, require a fast baseband processing engine. So choosing this standard as a first use case permitted us to evaluate the current platform design to find bottlenecks and possible solutions to overcome them.
To pave the way for a complete receiver chain, the Preprocessor prototype had to be designed. The latter connects the external Analog to Digital (A/D) and Digital to Analog (D/A) converters with the entire platform and embeds among others a Sample Rate Converter (SRC) for sample rate adjustment and I/Q imbalance correction.

To achieve all these contributions, the basic objectives have been grouped in five different tasks which are more detailed in the following:

1. The first step is the **emulation of the IEEE 802.11p receiver with the help of the Library for ExpressMIMO baseband (libembb)** to (1) validate the chosen algorithms in a pure software environment, (2) to verify if the platform functionality is sufficient for the IEEE 802.11p receiver design and (3) to obtain first performance figures based on the pure processing time of the DSP engines. This task is an iterative one. In case it already turns out at this step, that real-time processing is not possible even when neglecting the communication overhead, the algorithms have to be reworked and verified again.

2. After a successful completion of the first task, the development continues with **the implementation of the IEEE 802.11p receiver and its performance evaluation on the ExpressMIMO platform**. This proof of concept comprises a cycle accurate simulation in Modelsim and the receiver validation on the real hardware platform.

3. Once the work on the IEEE 802.11p receiver is finalized, we focus on the question **how DAB and IEEE 802.11p can be executed simultaneously on the ExpressMIMO platform**. Due to different standard properties, this task is very challenging. For the implementation

of a scheduler prototype we recall the performance figures obtained with libembb to derive basic guidelines for an efficient standard scheduling.

4. The **identification of design bottlenecks and the provision of possible solutions** is related to the tests on the hardware platform as mentioned in the second task. Based on the obtained results, possible algorithmic and design improvements are identified and possible solutions are provided and implemented.

5. To complete the IEEE 802.11p receiver chain, the final task includes the **implementation of a Preprocessor DSP engine prototype**. Here we mainly focus on the SRC as the most critical part of the Preprocessor in terms of area and space consumption.

1.2 Outline and Contributions

The work presented in this thesis is structured as follows:

1. First an overview of the basic terminology when talking about SDR applications is given in Chapter 2. Besides a presentation of latest SDR systems from academia and industry, a detailed description of the OpenAirInterface ExpressMIMO platform is provided. This description includes a detailed overview of the architecture as well as a presentation of the basic design methodology.

2. The IEEE 802.11p receiver is presented in Chapter 3. IEEE 802.11p is an enhancement of the well studied IEEE 802.11a standard commonly used in WLAN systems. In contrast to the latter, the IEEE 802.11p bandwidth has been reduced by one half, from 20 MHz to 10 MHz, to obtain OFDM (Orthogonal Frequency Division Multiplexing) symbols that are longer in time domain. This results not only in systems with large delay spreads to avoid Inter Symbol Interference (ISI) but also in stronger latency requirements which is a major design challenge as it requires a very fast baseband engine. Although the standard has been in draft form till July 2010, there are already few academic and some industrial transceiver solutions available. These solutions are mostly limited to the automotive context so that an efficient transceiver design on SDR platforms that are limited to wireless communications standards in general is still an open research topic. Implementing the IEEE 802.11p receiver on the ExpressMIMO platform comes therefore with the advantage that there are no limitations in possible combinations with other standards like LTE. Especially the combination with the latter is of high interest for future applications. Besides, the strong latency requirements of IEEE 802.11p make this standard the ideal first use case for the platform as it allows us to identify possible design bottlenecks. The obtained results are further extended by a possible combination of IEEE 802.11p and DAB. The combination of C2C and C2I communication with information about traffic jams or merchandising applications within only one device enables a lot of different future car applications. Preferred target technology are flexible SDR platforms allowing the execution of these different standards at low costs. The work on this task is still an open research topic as the necessary scheduler design is very challenging. For its design it is very important to have first key figures at hand.
   The results of this work have been obtained within the DeuFrako project PROTON (Programmable telematics on-board unit) / PLATA (PLAteforme Télématique multistandard pour l'Automobile) [1]. The implementation of the DAB receiver was in the responsibility of our german project partners.

Major contributions presented in this chapter are:

- an efficient physical layer implementation of the IEEE 802.11p receiver prototype for the ExpressMIMO platform (without the Preprocessor) that has been validated on the platform itself
- a Matlab prototype of the IEEE 802.11p receiver for algorithmic validation
- an IEEE 802.11p receiver emulation prototype based on the Library for ExpressMIMO baseband. The presented receiver is the first complete design implemented with the help of this library
- the identification of possible design improvements as well as their implementation. These are especially of interest for standards operating on short data sets.
- the derivation of a low latency scheduler design to execute multiple platform DSP engines simultaneously
- a detailed comparison of IEEE 802.11p and DAB
- a runtime performance comparison of IEEE 802.11p and DAB based on their emulation prototypes
- a derivation of basic guidelines for an efficient IEEE 802.11p and DAB scheduling on the ExpressMIMO platform as well as an implementation of a first scheduler design in software

Results have been presented and / or published

(a) at the Acropolis Summer School 2012 on Cognitive Wireless Communications (Poster Presentation)

(b) at the 7th Karlsruhe Workshop of Software Defined Radios [2]

(c) at the 15th EUROMICRO Conference on Digital System Design (DSD'12) [3]

3. One design bottleneck that has been identified in the previous chapter is the need for an optimized FEP design for standards operating on short data sets when Field Programmable Gate Arrays (FPGAs) are chosen as target technology. Limitations of this design were related to the huge communication overhead when compared to the pure execution time for short vector lengths. The FEP contains a vector processing unit and a DFT (Discrete Fourier Transform) / IDFT (Inverse Discrete Fourier Transform) unit and allows to implement different air-interface algorithms like channel estimation or synchronization. To overcome the observed limitations, the vector processing unit has been replaced by an Application Specific Instruction-set Processor (ASIP) solution. For development we have chosen the Language for Instruction-Set Architectures (LISA) that has gained commercial acceptance over the past years. To evaluate the proposed architecture, we compare it to the programmable DSP solution as well as to two recent ASIPs from academia. The comparison is based on the actual processing related to the cycle counts.
In Chapter 4 two different ASIP solutions are presented. The first one is based on the old FEP specification that has been reworked to gain a higher performance of the design. Major contribution is the second version of the ASIP that has been extended by general purpose instructions and whose internal latencies have been decreased significantly.

The obtained results of this work have been accomplished in collaboration with RWTH Aachen, Germany, in context of the cluster of ICT research network of excellence NEWCOM++ [4] and in context of the European FP7 project ACROPOLIS (Advanced coexistence technologies for radio optimization and unlicensed spectrum) [5].

Major contributions presented in this chapter are:

- a first ASIP implementation based on the old FEP specification
- a second ASIP implementation based on the new FEP specification
- a thorough comparison of the latest ASIP version with the programmable FEP DSP engine as well as with different ASIP solutions from academia

Results have been presented and / or published

(a) at workshops of the ACROPOLIS project

(b) within an official research deliverable of NEWCOM++ [6]

(c) at the DASIP'12 conference [7]

4. In the remainder of this thesis we focus on the design of a first Preprocessor DSP engine (Chapter 5) which is the only missing DSP engine in the IEEE 802.11p receiver chain. The Preprocessor connects the A/D, D/A converter interface with the remaining baseband engine and is responsible among others for I/Q imbalance correction, sample synchronous interrupt generation, framing and sample rate conversion. Most critical is the Sample Rate Converter (SRC) whose behavior can change dynamically at runtime. The SRC deals with the relation between the sampling rate at the A/D, D/A converters side and the baseband side. Processing the converters with a fixed master clock comes with the advantage of a low phase noise. In the past, one SRC was dedicated to each standard of interest but for the ExpressMIMO platform, this approach is too space consuming. That is why one fractional SRC architecture capable to process up- and downsampling is preferred.
The obtained results of this work have been accomplished within the cluster of ICT research network of excellence NEWCOM++.

Major contributions presented in this chapter are:

- a design of a fractional SRC able to process up to eight different channels (four in reception and four in transmission). All channels are executed on the same parameterizable hardware architecture. To guarantee a continuous filter processing, context switches between them happen instantaneously without any delay.
- an implementation of a first Preprocessor prototype for proof of concept and to complete the IEEE 802.11p receiver chain.
- an implementation of different C-models for design validation and verification.

Results have been presented and / or published

(a) at the NEWCOM++ Winterschool on "Flexible Radio and Related Technologies", 2009

(b) at the 6th Karlsruhe Workshop of Software Defined Radios [8]

(c) in the FREQUENZ journal [9]

5. The report finally concludes with the derivation of guidelines for a further standard deployment on the ExpressMIMO platform in Chapter 6

Chapter 2

SDR Baseband Processing

The need for the design of flexible SDR platforms for the automotive context is the main motivation for the work presented in this thesis. This chapter enhances some basic terminology and presents the different possible solutions to be considered when talking about SDR systems. These solutions motivate the implementation of the chosen target platform presented in Section 2.3. Besides architectural details of the OpenAirInterface ExpressMIMO platform we further introduce the related software emulation environment and describe the basic development methodology of transceiver applications.

2.1 Software Defined Radio

During the past years two major trends could be observed in the domain of wireless communications. Not only that the number of wireless communications standards was increasing rapidly, more and more standards have also been merged in more sophisticated devices like mobiles phones that include GPS and internet. To keep these devices as small as possible one preferable solution is the design of a global system that can also easily be updated to future standards. Costs are reduced not only because of the decreased manufacturing costs but also due to the decreased (re)design time in case of standard upgrades.
One solution to this challenging problem has been proposed by J. Mitola in [10] where he introduced the term of *Software Defined Radio*. Based on his initial definition, SDR has further been defined by the Wireless Innovation Forum (WIF, former SDR Forum) [11] and the EU Reconfiguration Radio Colloquium [12]. According to these sources, the ideal interpretation of SDR is a system where (1) the wideband digitization occurs next to the antenna and (2) where the actual transceiver application is running either on a desktop PC or on a very fast General Purpose Processor (GPP). Till now, common processors are still not fast enough to provide an efficient implementation of such a system [13]. Therefore today's approaches focus on baseband design for flexible hardware platforms, multiprocessor systems or Network on Chips (NoCs). The candidates for such designs are manifold and comprise among others Application-Specific Integrated Circuits (ASICs), ASIPs, DSPs or FPGAs. The final design choice depends on the application to be designed and related to that on different criteria like power efficiency, flexibility, reconfigurability and usability. Flexibility in SDR systems is usually related to reconfigurability and thus representing a system that is able to change its behavior dynamically in case the surrounding environment is changing. Or more specifically, the wireless network as well as its equipment can dynamically adapt to environment changes. Reconfigurability can be increased by two different factors: (1) a higher programmability of the design typically achieved by using FPGAs or microcontrollers

and (2) by a modular design approach. The more modular an architecture is, the easier it is to replace components in case future standards or improvements of current standards require design upgrades.

SDR systems have first been developed for military targets in the context of projects like SPEAKeasy [14] and JTRS [15]. Since then, SDR has moved towards civil and daily life applications like the automotive context on which we will mainly direct our attention in this thesis. However, an exact definition of SDR can still not be found in the literature. In 2005, the Federal Communications Commission (FCC) defined SDR as "a radio that includes a transmitter in which operating parameters of frequency range, modulation type or maximum output power (either radiated or conducted), or the circumstances under which the transmitter operates in accordance with Commission rules, can be altered by making a change in software without making any changes to hardware components that affect the radio frequency emissions." [16]. In contrast, WIF has split SDR in several so-called *tiers*. Their separation is performed by means of capabilities and thus the extend of (re)configurability of a certain system.

- **Tier 0 - Hardware Radio (HR)**
  Tier 0 is the simplest example of an SDR system. For its implementation only hardware components are used which results in a complete redesign in case of modifications.
- **Tier 1 - Software Controlled Radio (SCR)**
  In this tier, the control functions are implemented in software, resulting in a limited number of programmable functions like interconnections.
- **Tier 2 - Software Defined Radio (SDR)**
  Designs related to this tier are usually split into two parts: the control part implemented in software and the flexible hardware part. For the design of the latter, dedicated hardware blocks such as ASICs, FPGAs, DSPs, etc can be used to increase the flexibility of the design. A well-known tier 2 application are SDR platforms.
- **Tier 3 - Ideal Software Radio (ISR)**
  At this tier, the entire system is programmable. Analog conversion is performed only at the antenna, speaker and microphones.
- **Tier 4 - Ultimate Software Radio (USR)**
  Tier 4 has been defined for comparison purposes only. Here the system is fully programmable and switches between the air-interfaces can be performed in milliseconds.

The work presented in this thesis can be assigned to tier 2 as the chosen target architecture is a flexible SDR platform. To motivate this design choice an overview of different SDR systems is given in Section 2.2.

2.2 Related Work

Today's wireless communication systems all follow a common processing flow illustrated in Fig. 2.1. After receiving the signal samples from the Media Access Control (MAC) layer, the transmitter chain, TX, embraces source encoding, channel encoding, digital modulation and D/A conversion before the signal is transmitted through the Radio Frequency (RF) back-end. In the other direction, the receiver chain, RX, is composed of an A/D converter, digital demodulation, channel decoding and source decoding before the samples are given to the MAC layer.

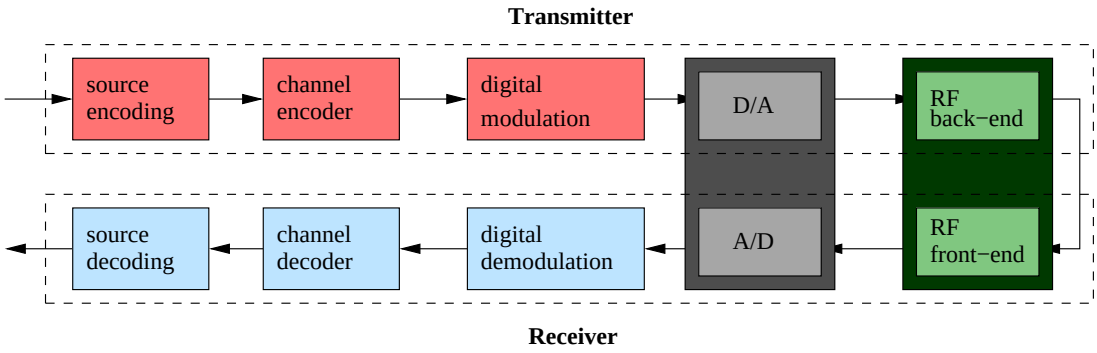


Figure 2.1: Overview of a Wireless Communication System

The required basic building blocks for a possible mapping of these different tasks on a flexible SDR platform being capable to execute different wireless communication standards simultaneously by reusing the same HW resources is shown in Fig. 2.2. These blocks comprise different kinds of processors and a main CPU responsible for the SW control. Sample processors are fast processing engines that operate in the order of ns. Once started they perform autonomously and my influence the scheduling decisions of the main CPU by timing and synchronizations events that may occur after a predefined number of generated output samples. One example to which we also refer to in this thesis is a preprocessing unit including a SRC. The tasks of block processors instead operate in the order of 10s of μs and are regularly scheduled by the main CPU which consequently results in higher SW dependencies when compared to the sample processors. Common usage are among others vector processing units required for the computation of different air-interface operations (e.g. channel estimation, synchronization, etc.). Although different approaches are possible for these kind of hardware accelerators, we will mainly focus on two different types in this thesis: (1) basic processing units that can be combined with microcontrollers and (2) ASIPs where the decoding of the program instructions is merged with the processing unit.

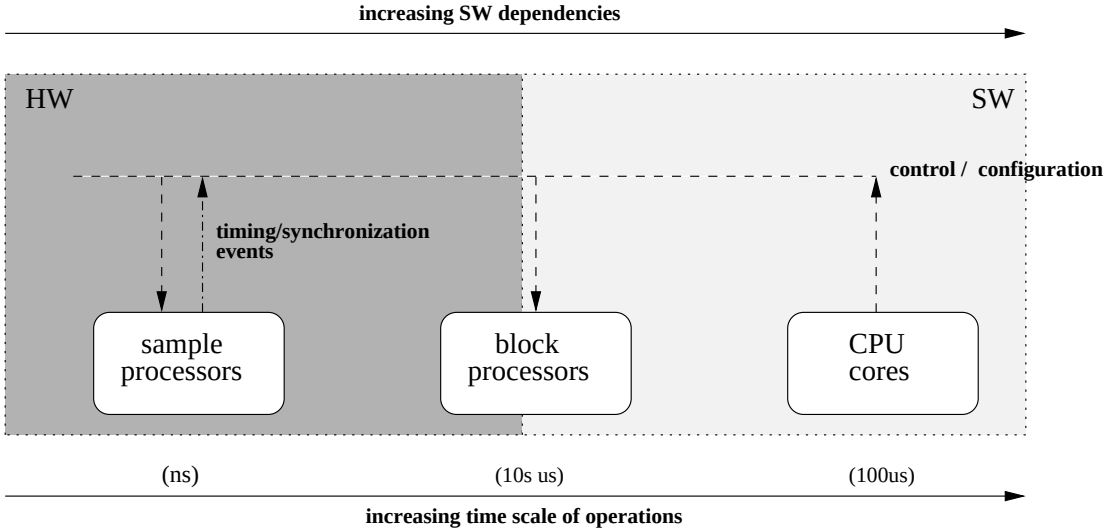


Figure 2.2: Basic Building Blocks for Hardware Mapping

With each new upcoming standard, the computational complexity of the existing designs is increasing. Therefore there is a high need for flexible architectures that meet the requirements of all current wireless communication standards and that are very efficient when dealing with the increasing performance requirements of new ones. This is especially challenging for standards with short data sets. In general, five keywords summarize the main challenges in the design of flexible SDR systems:

1. **Flexibility**, which stands for the efficient integration of new features in the existing design as well as dynamic switching between standards at runtime.

2. **Portability**, which measures how easy the design components can be transferred from one system to another.

3. **Scalability**, which defines how easy a design can deal with the requirements of future upcoming standards.

4. **Performance**, which stands for the efficiency of the application implementations.

5. **Programmability**, which is important when realizing a multimodal processing design.

Further criteria are the amount of required computations, power consumption, number of standards being supported and computational latency.

During the past years, an increasing number of different commercial SDR solutions has already been presented. These solutions have been designed using different technologies like flexible hardware platforms, microprocessor systems or NoCs. Pioneer in the domain of commercial SDR design is VANU SDR [17] whose work has started in the the last 1990s at MIT for a GSM softbasestation in context of the SpectrumWare project.
In the following, an overview of different commercial solutions is given. The list does not demand to be complete, but shall rather illustrate the different design strategies.

- **USRP2 in combination with GNU Radio:** GNU radio [18] is a free software development toolkit for SDR applications. It includes over 100 different signal processing blocks and is the primary platform using PC drivers for the Universal Software Radio Peripheral (USRP) developed by Ettus Research [19]. The motherboard contains among others A/D and D/A converters, an FPGA for sample rate conversion and Multiple Input, Multiple Output (MIMO) connectors while the daughterboards contain the RF equipment. The connection to the PC is established via a Gigabit Ethernet. The whole signal processing is done by running GNU radio on the desktop PC. The main drawback of this design is the huge latency for data processing due to GNU radio.

- **Deep eXecution Processor (DXP):** The DXP ([20], [21]) is a low power Single Instruction, Multiple Data (SIMD) processor that was initially developed by Icera before the latter was taken over by NVIDIA. It has been designed for wireless communication algorithms and control code whereas the implementation of the physical layer is a GNU radio based software code running on the DXP. The data path of the DXP has a depth of four and the configuration is done by pointing to a configuration map that contains different configurations and parameters required for processing. Per clock cycle, one of these configurations can be provided. Up to now, there is no information about architectural details available.

- **Intel (former Infineon) MuSIC 1 chip:** The MuSIC 1 chip presented in [22] is based on an SIMD chip that has been designed for mobile phones. Thus it comes with a small overall design and very low power consumption. The processing is split over four SIMD cores operating at a frequency of 300 MHz. Each of them contains four processing elements and can operate independently to implement hardware multitasking. Besides, programmable hardware accelerators are included for filtering and channel de/encoding. As all components access the same memory, a multilayer bus bridge is used for simultaneous memory access support. Apart from that the chips come with a digital interface to the RF front-end.

- **ST-Ericsson EVP:** The EVP [23] is already available in silicon since 2007 and was planned to be used in cellular systems starting from 2008. Like the MuSIC 1 chip it belongs to the SIMD class. The EVP is a high performance vector processor whose achieved frequency is of about 300 MHz. It operates on 256 bit vectors with a size of 16 bit and supports Very Long Instruction Words (VLIW) to enable a parallel vector processing. The architecture is divided into four basic blocks which are: Program Control Unit, Vector Data Computation Unit, Scalar Data Computation Unit and Address Computation Unit. The Vector Data Computation Unit operates on integer, fixed-point and complex data types. Its functions include different arithmetic and logical operations. In addition, an interleaver for shuffling and a channel decoder are part of the design. A major drawback of the EVP is the low performance for complex bit based operations.

- **Systemonic HiperSonic 1:** The Hipersonic 1 [24] is an Application Specific Signal Product (ASSP) baseband design which is split in a hard-wired logic for computationally intensive processing and a configurable SIMD/VLIW based DSP unit called *onDSP* which is used for DSP intensive algorithms. The HiperSonic 1 has been designed for IEEE 802.11a and HIPERLAN/2 based 5 GHz WLAN applications.

- **Freescale MSC8156 high-performance DSP:** The MSC8156 [25] is a programmable DSP unit build on the StarCore technology. It is a multi-accelerator platform for (Inverse) Fast Fourier Transform (FFT/IFFT) computations, turbo decoding and Viterbi decoding. It further includes serial rapid I/O interfaces, a PCIexpress interface and Double Date Rate (DDR) controllers for high-speed. Other modules such as interleavers or channel decoders are not included in the design.

- **Sandbridge SB3011 Platform:** The Sandbridge SB3011 platform is presented in [26]. It includes four DSPs each running at a minimum frequency of 600 MHz at 0.9 V. The chip is fabricated at a 90 nm technology. Each DSP can process multiple operations per cycle including data parallel vector operations. Up to 32 independent instructions can be executed simultaneously. The interconnection is established via a unidirectional ring network. Further the platform contains among others an ARM926EJ-S processor and different interfaces like USB or Ethernet. This DSP is used for front-end processing only.

A major drawback of SIMD designs is the low performance channel decoder whose improvement is still an open research topic. When talking about VLIW processors instead, the major challenge is to cope with the resulting large program executables or the excessive register file write ports resulting in a high power dissipation. For this reason, EVP, onDSP and the Synchronous Transfer Architecture (STA) provide specialized instructions and multiple register files. STA has has been proposed by researchers of TU Dresden in [27] as an extension of the concept of the Task Triggered Architecture (TTA, [28]). Using STA, a network is formed in which each functional unit is connected to other functional units. The delay between two nodes is one clock cycle. Based

on the STA design, Register Transfer Level (RTL) code and simulation models can be generated automatically.
Another design methodology with the aim of building networks are NoCs. One example for a NoC based platform is MAGALI [29] which is developed by CEA, France. This NoC supports different OFDMA/MIMO standards and offers a high reconfiguration speed of the system. One major drawback of using NoCs is the power consumption which still has to be optimized. Furthermore these systems may include possible deadlocks or livelocks in their network.

2.3 OpenAirInterface ExpressMIMO Platform

The OpenAirInterface ExpressMIMO platform ([30], [31]) has been developed by Eurecom and Télécom ParisTech. It potentially supports a wide range of different standards like GSM, UMTS, WLAN, DAB, LTE as well as their multimodal processing and Time / Frequency Division Duplex (TDD / FDD) modes. The platform is capable to process up to eight different channels simultaneously (four in reception, four in transmission) by reusing the same HW resources. As each channel may support a different wireless communication standard, the main design challenge is the synchronization of these resources by providing a maximum accuracy and by meeting all the real-time requirements.
ExpressMIMO is used for experimental purposes only. Therefore the chosen target technology are FPGAs which come with a reduced design time, higher runtime flexibility, simple ease of use and lower costs for small quantities when compared to other solutions. Nevertheless ASICs are considered in a future version once the whole baseband design has been validated.

In contrast to the previously presented solutions, the current design of the ExpressMIMO platform is split over two different FPGAs from Xilinx: (1) a Virtex 5 LX330 for the baseband processing and (2) a Virtex 5 LX110T for interfacing and control (Fig. 2.3). To simplify testing on the platform, the two FPGAs can run stand-alone if required. Another difference is that the baseband processing being responsible for the signal processing of the transceiver is split over different DSP engines that are explained more detailed in Chapter 2.3.2.2. The underlying hardware architecture further allows to process four receive and four transmit channels in parallel by using the same resources.
The interface and control FPGA transfers the signal coming from / going to the MAC layer and contains the main CPU (SPARC LEON3 processor) being responsible for the main control flow of the system. The two FPGAs are connected via an AMBA / AVCI DSP bridge while the different DSPs on the baseband side are connected via an AVCI crossbar. As only seven DSPs plus the VCI RAM and the main CPU are connected with each other, the performance of this crossbar is sufficient for the design of the ExpressMIMO platform.
The available memory space is distributed in a non-uniform way. Each DSP engine has its own memory space that is also mapped onto a global memory map. This global map is provided to the main CPU and to the DSPs and is consulted in case of DMA transfers between the DSPs or between the two FPGAs. For internal processing, the DSPs apply a local addressing scheme. In addition, an external DDR memory is available for mass storage on the baseband side and a DDR2 memory (size 16 MByte) contains the LEON3 program code and can be used for mass storage on the control side.

Currently the whole design is running at a frequency of 100 MHz but the target is to increase this frequency to the maximum possible one of the main CPU (133 MHz) in the future.

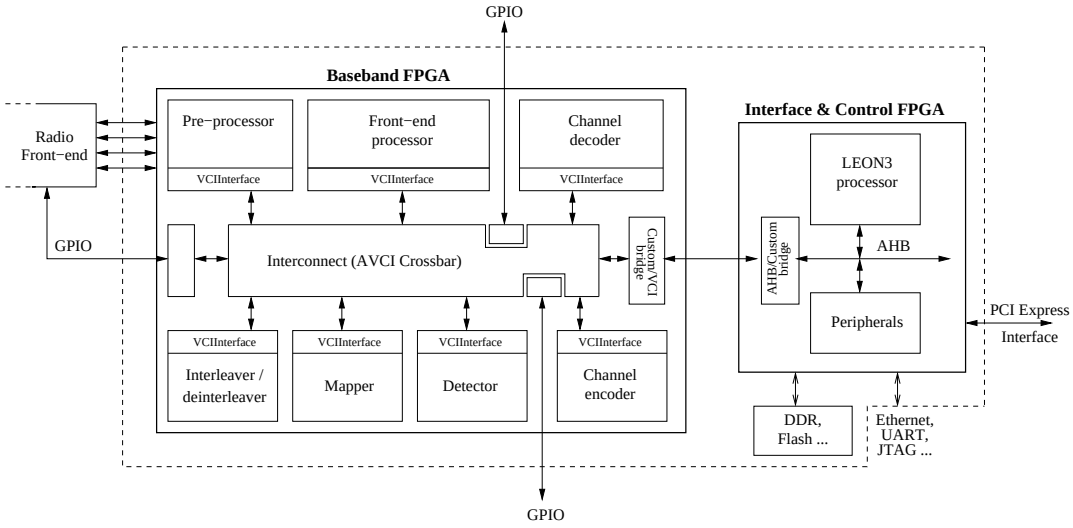


Figure 2.3: Baseband Architecture of the ExpressMIMO Platform

2.3.1 Control

The interface and control FPGA connects the ExpressMIMO platform with the external host PC by a JTAG and a PCIexpress connection (8-way when connected to a desktop PC, 1-way when connected to a laptop). The FPGA is further connected to a DDR2 memory available for mass storage of samples. Main component on the FPGA is the 32 bit SPARC LEON3 processor from Gaisler Aeroflex [32] that serves as main CPU for the baseband processing. In the future it is considered to replace it by a multiprocessor solution. An interesting candidate is the Xilinx Zynq [33] which includes a ARM Cortex A9. In contrast to LEON3 running at a maximum possible frequency of 133 MHz, Xilinx Zynq can be processed up to 800 MHz.
Currently, all DSPs are controlled by the LEON3 processor who can program them by writing into or reading from the memory-mapped control registers and the memory-mapped local memories inside the DSPs. Data transfers between DSPs and from / to LEON3 can be established by either writing directly at corresponding global memory addresses or by DMA transfers. Observed programming latencies are related to the bridge connecting the two FPGAs. To minimize these latencies, it is planned to investigate in the effects of a distributed control flow on the platform. From the software point of view, the platform includes three different kinds of possible execution nodes: (1) the main CPU LEON3, (2) the microcontroller (UC) that can be included in each of the DSPs and (3) the DSPs itself. It is obvious that when splitting the control flow the design of the C application code running on LEON3 will become more challenging. But on the other side a distributed control flow will result in a more efficient transceiver processing, especially when executing multiple standards in parallel.

2.3.1.1 Choice of the Operating System for LEON3

The SPARC LEON3 processor is supported by various Operating Systems (OS) like eCos, RTEMS (Real-Time Executive for Multiprocessor Systems) and freeRTOS (free Real-Time Operating System) which are all free and VXWorks which is not free. The main similarity between them is that they all use function calls (or static links) instead of system calls to reduce their internal latencies. For single processor systems, all of them achieve a very good performance which would make

them ideal candidates for the current version of the ExpressMIMO platform. However, a future version of ExpressMIMO will include a multiprocessor system. To avoid a time-consuming software redesign it is therefore recommended to choose a free OS with multiprocessor support.
A disadvantage of RTEMS is that it needs to run one instance of the OS per processor in the system. FreeRTOS has not multiprocessor support at all and the eCos multiprocessor support is still limited [34]. Therefore we decided to opt for MutekH ([35], [36]) which was originally designed to support multiprocessor heterogeneity of nowadays platforms. In contrast to the mentioned OS, MutekH provides a shared memory multiprocessors support and has been designed with strong multiprocessor support in mind. It further provides optimized function calls by using an appropriate set of inline functions. This reduces the latency of calls to the kernel which are frequent in parallel applications that are split in multiple threads to take advantages of several processors. For SPARC processors, unlike other kernels, MutekH uses the flat function call convention. This improves the interrupt latency and makes the function call time far more deterministic. Usually, SPARC comes with 32 general purpose registers that are always visible by the program. 24 of them are organized in a register window that is split over three different groups of eight registers. They are stated as `out`, `local` and `in`. The visible window per time instance is determined by the so-called *Current Window Pointer*. Using `save` and `restore` instructions that can be found at the beginning and at the end of each function, this pointer is moving. The register windows are overlapping, so the `out` registers are renamed when `save` is called and become the `in` registers. In addition to that, the Window Invalid Mask (WIM) register indicates if a window is invalid which results in copying the whole stack to the memory. All the mentioned processing operations sometimes result in a huge overhead which is very critical when processing standards with strong latency requirements. Therefore, MutekH has been optimized by a flat registers model where the compiler does not use `save` and `restore` instructions. The extra register windows which are not needed by the regular code can then be used to implement really fast interrupts context switching for free. All of these improvements reduce the latency significantly and make the ExpressMIMO platform also suitable for the processing of standards with short data sets. For multimodal processing, MutekH supports POSIX threads so that different transceivers can be executed on LEON3 simultaneously.

2.3.2 Baseband Design and Emulation

2.3.2.1 Generic DSP Shell

The architecture of the different DSP engines is based on a standardized DSP Shell (Fig. 2.4) which is composed of

- **a Control Sub-System (CSS)**
  The CSS is common to all DSP engines and is specialized through parameters. It optionally contains a local 8 bit UC (6502) and a 64 bit DMA engine as well as a set of control and status registers plus several arbiters and FIFOs for input-output requests and responses. Furthermore, the CSS acts as a gateway with the surrounding host system by using two 64 bit wide AVCI compliant interfaces. The first one is a slave (target) interface through which read and write requests to the internal control and status registers and to the Memory Sub-System (MSS) are received. The second one is a master (initiator) interface required by the DMA to perform data transfers between the MSS and external memory areas. In addition, a set of input and output interrupt lines is used for signaling and synchronization with the host system.
  The architecture of the UC inside the CSS is based on a Complex Instruction-Set Computer

(CISC) with 6 internal registers. Its address bus has a width of 16 bit and the reserved UC address space in the MSS has a size of 2 kB.

- **a Processing Unit (PU)**
  The PU is custom defined and depends on the functionality of the DSP. It is the main component of each DSP engine. The instructions required for the PU processing are received through the CSS and are stored in the control registers. So programming a DSP just means writing the parameters into the right registers.

- **a Memory Sub-System (MSS)**
  Like the PU, the MSS is custom defined and depends on the functionality of the DSP. The MSS contains the address space for the program and data memory of the UC with a size of 2 kByte and the input-output data space with a variable memory size. To increase the maximum achievable frequency after place and route, the number of registers before and after the actual RAM inside the MSS is variable and may differ between the different DSP engines.

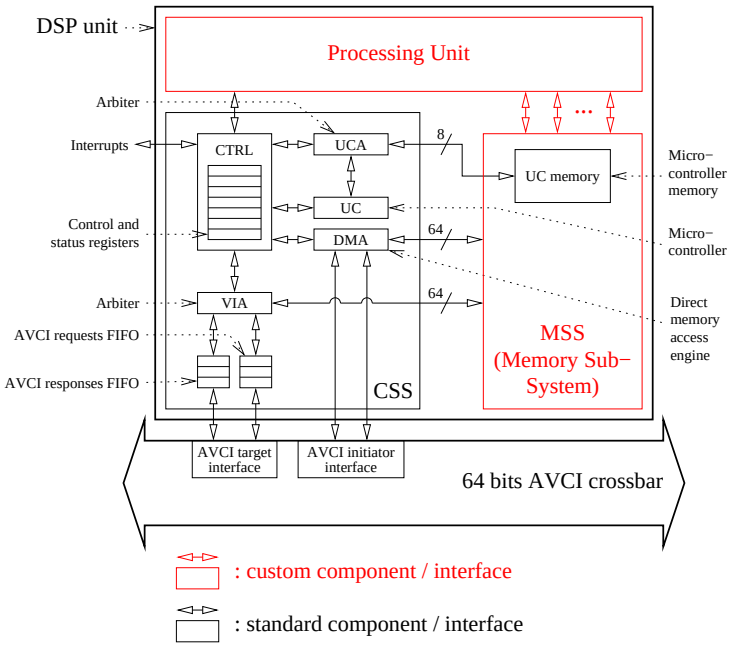


Figure 2.4: OpenAirInterface Standardized DSP Shell

From the LEON3 point of view, all DSP engines are seen as a memory block mapped onto the global memory map. The size of each of these memories is set to 1 MByte and is aligned on a 1 MByte boundary. The UC and the DMA access memory spaces inside this memory but without having access to the global memory map.
For the time being, the UC has not been integrated in the CSS yet. The current version of the receiver is thus orchestrated by a centralized control flow where the whole transceiver program is running on the main CPU.
In the future a global control flow including the UC will be applied to reduce the interrupt rate and the communication overhead to the main CPU. Currently, the latter starts a DSP by writing a value in the so-called `igost` (Ip GO and STatus) register. Once the operation is finished an interrupt

is raised. Each DSP unit has three different interrupt lines used for signaling to the host system when the scheduled task is finished: (1) UIRQ (UC), (2) DIRQ (DMA) and (3) IIRQ (PU). As an alternative, the main CPU can poll the `ibsy` flag of the `igost` register to get to know about the end of the PU processing. An important CSS feature is that one new command can already be prepared in the command registers. Once this happens the `ipend` flag is set to one to indicate that no more command can be prepared. The same rules apply as well when programming the DMA engine included in the CSS. In this case, the register `dgost` provides the main CPU with the status information.

2.3.2.2 Overview of the different DSP engines

In general, the baseband design takes place between the RF front- and back-end and the decoded signal samples. It represents the implementation of the physical layer while MAC layer operations are performed on the host PC. As mentioned earlier the baseband processing of the Express-MIMO platform is structured in independent DSP engines which allow an easy upgrade to future standards. Other advantages include the effective use of spectrum, mobility, increased network capacity, maintenance of cost reduction and a faster development of new services. The DSPs have been designed in such a way that they support the most computationally intensive tasks in an efficient way. Prior to that, a detailed analysis of the commonalities between the standards has been carried out to make sure that the platform supports all current wireless communications standards by minimizing the resource consumption without the lack of high accuracy. The final designs are programmable, reconfigurable at runtime and can be processed in parallel which is of a significant importance for multimodal applications.
In the context of different studies throughout the past years, seven different DSPs have been identified:

- **Preprocessor (PP):** The Preprocessor connects the external RF with the baseband system. The four A/D and four D/A converters (AD9832) provide 2x14 bit at 128 Ms/s in TX and 2x12 bit at 64 Ms/s in RX. Besides, the Preprocessor is used for basic signal processing functions including sample rate conversion, an NCO (Numerically Controlled Oscillator), I/Q imbalance correction as well as framing, (re)synchronization and sample synchronous interrupt generation. More details about the functionality of this DSP are provided in Chapter 5.

- **Front-End Processor (FEP):** The FEP is responsible for the different air-operations like channel estimation, synchronization, etc. A detailed analysis of the required operations and a first FEP design have already been carried out in [37] and have further been detailed and optimized in the past years. The resulting design contains a vector processing unit as well as a DFT/IDFT unit. Supported input and output data types are integers of 8 or 16 bit or complex values with a size of 16 or 32 bit.
  The FEP comprises five vector operations. The two input vectors are denoted as $X[i]$ and $Y[i]$, the result vector is denoted as $Z[i]$.

  - **Component-Wise Addition** (CWA): $Z[i] = X[i] + Y[i]$
  - **Component-Wise Product** (CWP): $Z[i] = X[i] \odot Y[i]$
  - **Component-Wise Square of Modulus** (CWSM): $Z[i] = |X[i]|^2$
  - **MOVe** (MOV): copies a vector from one MSS location to another
  - **Component-Wise Look-up Table** (CWL): $Z[i] = Y[X[i]]$

Input vectors can further be modified by applying force to zero, negate or absolute value operations to the real and imaginary part while the output vector $Z[i]$ can be rescaled or saturated. In addition to $Z[i]$ the FEP can provide some more results (sum / max, min / argmax, argmin of $Z[i]$) if required. These values are further denoted as `SMA values`. Another important feature of the FEP is its flexible Address Generation Unit (AGU) that can be used for address skipping or address repetition and that allows an easy realization of circular buffers inside the FEP MSS. The latter is split in four different banks each with a size of 16 kB. For vector operations, the two input vectors and the output vector have to be stored in different memory banks. More details about this DSP are provided in Chapter 4 where an ASIP implementation of the FEP is presented.

- **(De)Interleaver ((DE)INTL):** This DSP is a block (De)Interleaver with a throughput of one sample per cycle. Its MSS is split over three different memories: input and output memory space have a size of 64 kB, the permutation table memory has a size of 128 kB. Further operations supported are puncturing, value repetition and value insertion by using the zero or one forcing option. All operations can either operate on bit or on byte. The basic functionality of the (De)Interleaver is illustrated in Fig. 2.5. The address of the output buffer is directly correlated to the address of the permutation buffer containing the related input buffer address.

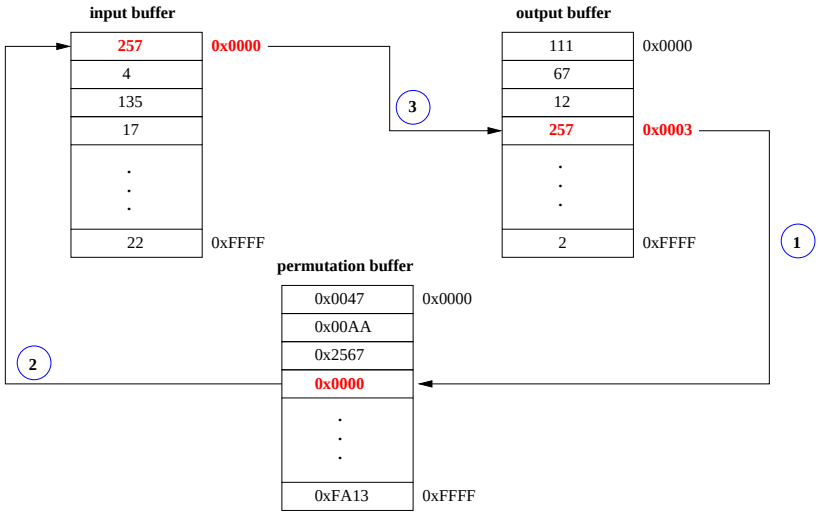


Figure 2.5: Illustration of the basic (De)Interleaver Functionality

- **Channel Decoder (CHDEC):** The Channel Decoder implements trellis based decoding algorithms - more specifically a Viterbi (< 256 states, traceback algorithms) and 8-state Turbo decoders (max-log-map / sliding window algorithm) for binary convolutional codes to cover almost all current systems. There are no restrictions concerning the choice of the generator polynomial. Accepted code rates are 1/2 and 1/3. The size of the traceback window is 5 x k with k as the constraint length. Supported constraint lengths are 7 and 9 for the Viterbi decoder and 4 for the Turbo decoder. For the latter the number of iterations can be programmed from 1 to 8. To increase the performance of standards with short data sets, a tail-biting option has been added to the Viterbi decoder. The MSS of the Channel Decoder is split over three different sections: (1) input data memory (32 kB), (2) output data memory (16 kB) and (3) intermediate data memory (40 kB).

- **Channel Encoder:** The Channel Encoder contains a convolutional encoder, block cyclic codes and m-sequences. For the time being this DSP is not included in the design of the ExpressMIMO platform.
- **Mapper and Detector:** These DSPs perform a set of different modulation schemes which are BPSK, QPSK, 8PSK, 16-QAM, 32-QAM, 64-QAM and 256-QAM. The input memory of the mapper has a size of 8 kB, the output memory a size of 16 kB. Each input symbol is considered as an address of a Look-Up Table (LUT) with a size of 4 kB from where the related output value is read.

All DSPs and the VCI RAM are connected via a generic Advanced Virtual Component Interface (AVCI) crossbar ([38], [39]). The VCI RAM is used for temporary sample storage on the baseband side. It is mapped onto the global memory map and has a size of 16 kB. The resource allocation of all connected devices is handled by a Round Robin policy.

2.3.2.3 Processing Times

The processing time of all DSPs and DMA transfers is deterministic and can be precalculated if required. Tab. 2.1 illustrates how to compute these times for the DSP engines and DMA transfers being considered in the remainder of this report.

Operation	**Number of Cycles**
FEP - DFT/IDFT	$T = 2 + (13 + \frac{L}{8}) * (\lfloor \frac{n+1}{2} \rfloor)$ $L = 2^n$ components vector
FEP - Vector Operations	$T = \lfloor \frac{L+1}{2} \rfloor + 11 + x + y$ $x = 4$ for CWL $y = 1$ if SMA value computations $x = y = 0$ when others
DE(INTL)	number of samples + 16
CHDEC (Viterbi)	number of samples + 16
DMA: LEON - DSP	$\frac{\text{number of bytes}}{4} + 24$
direct: LEON - DSP	7
direct: DSP - LEON	10
DMA: DSP - DSP	$\frac{\text{number of bytes}}{8} + 24$
direct: DSP - DSP	18

Table 2.1: ExpressMIMO Cycle Counts

Memcopy transfers denoted as `direct` correspond to transfers where LEON3 reads / writes directly in the baseband memory locations by using the global memory map.

2.3.2.4 Receiver Emulation using the Library for ExpressMIMO baseband (libembb)

The emulation environment of the ExpressMIMO platform called library for ExpressMIMO baseband (*libembb*) allows an easy validation and verification of the design in a pure software environment. It is developed by the System on Chip Laboratory of Télécom ParisTech and is an open-source C++ library that has already been applied in different European projects like SACRA [40] or PLATA [1]. The functions included in libembb are bit-accurate and represent all functions on the baseband side. The API of libembb provides basic commands for the main CPU and the local UCs as well as synchronization and signaling including error messages. In the future the

design will be extended by a cycle accurate SystemC model.
Currently, two different implementations are provided: (1) a C++ emulation layer and (2) C-language hardware dependent drivers. In case of the so-called `synchronous application`, no parallelism is supported. The application is designed with the libembb C-API and the code that is run in emulation and on the hardware target is the same as the parallelism of the different DSPs on the platform is not yet exploit. In contrast parallelism has been added for the `asynchronous application`. The emulation code running on the desktop PC is now multi-threaded and can be used unmodified for hardware processing where it exploits the parallelism of all resources.
Fig. 2.6 illustrates this general processing flow.

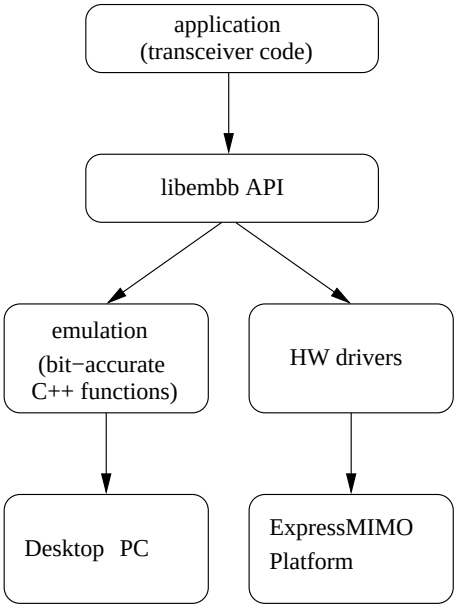


Figure 2.6: libembb Processing Flow

2.3.3 Development Methodology

The transceiver design methodology applied for any design developed for the ExpressMIMO platform can be divided in several steps.
Step one is the **development of a purely functional model** which is the common starting point for all transceiver designs. The goal of this step is to analyse the algorithmic part of the transceiver, to identify the required resources, the data flow and data dependencies. Thus, it is already possible to identify bottlenecks when processing several transceivers in a multimodal way on the platform. The considered models are typically sequential and do not yet fully exploit the parallelism of the target platform. For the design of the ExpressMIMO platform, the presented libembb library is used for the functional model design.
Step two is the **cycle accurate HW/SW co-simulation**. This step allows to fully exploit the parallelism on the platform. A common approach is the HW/SW co-simulation in discrete event simulators such as Modelsim. The parallelism on the platform includes simultaneous processing of the DSPs, data transfers using the DMAs as well as the preparation of commands in the standardized DSP shell. Results of this step are cycle accurate performance figures of the developed

transceiver to get to know the actual performance of the design. Unfortunately the usage of Modelsim is only appropriate for standards with short data sets as the initialization time of a standard like DAB for example is already in the order of 10^5 cycles.
The final step is the **transceiver validation on the hardware platform** where the design is tested and validated on the real hardware platform. For this step first known snapshots are applied before the signal received through the RF is decoded.

2.4 Conclusions

In this chapter, an introduction to the basic terminology of SDR has been given. Based on the presented existing industrial and academic solutions, the need for flexible SDR platform design was enhanced and explained more detailed using the example of the OpenAirInterface ExpressMIMO platform. As the further chapters are all related to this target platform we presented a detailed description of the architecture and introduced the library for ExpressMIMO baseband (libembb) used for transceiver emulation in a software environment.

Chapter 3

IEEE 802.11p Receiver for the ExpressMIMO Platform

In the automotive context, SDR platforms are of high interest as they allow to combine Car-to-Car and Car-to-Infrastructure communication with information about traffic jams or merchandising applications within only one device. In best case, future upgrades of such platforms only result in a software update and do not require a time consuming hardware redesign. This chapter focuses on the implementation of an IEEE 802.11p receiver for the ExpressMIMO platform. In contrast to other standards, the data sets of IEEE 802.11p are very short and thus require a very fast baseband processing. This makes this standard the ideal first use case for the platform to identify bottlenecks in the platform design and to obtain first performance figures.
After a short description of the IEEE 802.11p standard and a presentation of existing transceiver solutions, the main part of this chapter focuses on the different receiver implementations for the ExpressMIMO platform. These include a Matlab model, an emulation prototype obtained with libembb and the prototype running on the real hardware. In the context of a case study we further concentrate on the combination of C2X communication and TPEG information which is still an open research topic. For this purpose we have a close look at the differences between the two invoked standards, IEEE 802.11p and ETSI DAB. Considering the applied centralized control flow, resource management and thus the scheduling are assumed to happen in the main CPU. For the design of an efficient scheduler it is very important to have first key figures at hand. These figures are obtained by a performance comparison of the two standards using libembb. Based on these results we derive first guidelines for an efficient scheduling on the ExpressMIMO platform and present a first scheduler prototype. Our gained experience is summarized at the end of this chapter to provide guidelines for a future standard deployment.

3.1 Motivation

Currently, experts focus on the design for C2C and C2I communication also known as Vehicle-to-Vehicle and Vehicle-to-Infrastructure communication. The basic concept of C2C communication is the following: once one car sends messages to many others via a wireless communication channel the cars spontaneously form an ad hoc network which is known as Vehicular Ad Hoc Network (VANET). VANETS extend the drivers view of the road which may be limited due to darkness or obstacles and take into account that the driver may need some time to react to an unexpected event. Possible use cases focus on the reduction of traffic jams and accidents and include collision prevention, monitoring of hazardous vehicles, accident warnings, active navigation, etc.

C2X (X = Car, Infrastructure) communication is part of future Intelligent Transport Systems (ITS). An excellent overview of ITS is given in [41]. In this document not only various scenarios are presented but also the frequency allocation differences between several countries are enhanced. Possible applications in ITS are not only safety applications, but also traffic jam avoidance, toll collection, tourist information, mobile internet, etc. In general they can be divided in non-safety and safety applications where the higher priority is given to the latter. To distinguish between the different ITS applications, [42] proposed a set of important criteria for C2X communication which are usability, robustness, cost, efficiency, scalability and development effort.
In August 2008, the Commission of European Communities decided that the 5.875 - 5.905 GHz frequency band is dedicated for safety related applications of ITS [43]. The division of this frequency band is defined by the European Telecommunications Standards Institute (ETSI) in [44]. It is subdivided in several channels with a width of 10 MHz that can be combined to achieve higher data rates. A standard of main interest in this context is WLAN IEEE 802.11p ([45], [46]) which is an enhancement of the well-known IEEE 802.11a standard [47]. In contrast to the latter the bandwidth of IEEE 802.11p has been reduced from 20 MHz to 10 MHz. This results in OFDM symbols that are longer in the time domain and thus in systems with large delay spreads to avoid ISI. ISI is of major importance for vehicular use cases where the channels are strongly time-varying. So a reliable reception of the transmitted signal can still be guaranteed. The IEEE 802.11p standard is also known under the name Wireless Access in Vehicular Environments (WAVE) which has its origin in 1999 when the US Federal Communication Commission allocated 75 MHz of the Dedicated Short Range Communication (DSRC) spectrum exclusively for C2X communication. A good overview of DSRC is provided in [48]. As the standard has been in draft form till July 2010 an efficient transceiver design is still an open research topic. This task is quite challenging as compared to other standards, IEEE 802.11p transceivers come with very strong latency requirements and thus require a very fast baseband processing engine.

An important ITS project is the German SimTD project [49], where C2X communication is implemented on the physical and on the MAC layer. In the context of SimTD, real experiments are performed in the region around Frankfurt am Main in Germany. Test tracks include the highway as well as some parts of downtown Frankfurt. The goals of SimTD are manifold. Besides the definition of scenarios and their identification in real experiments, experts mainly focus on the implementation of C2X functions to improve the road safety. Standards of interest are IEEE 802.11p and GPRS and UMTS that have been integrated in case WLAN is not available.

3.1.1 Related Work

The IEEE 802.11p standard has been in draft form till July 2010, so efficient physical layer implementations for SDR platforms are still an open research topic. Up to now, most of the published papers focus on theoretical and performance aspects. These studies are beyond the scope of this report but will nevertheless be mentioned for the sake of completeness.
A general performance evaluation of the IEEE 802.11p standard including the MAC layer has been derived during analysis and simulation in [50] or [51]. Others papers focus on measurements that have been taken under real conditions to find out how reliable the packet transmission is by determining the Packet Error Rate (PER). In [52] a measurement study has been carried out on a C2I trail on an Austrian highway using an IEEE 802.11p prototype. Their main observations are that shadowing effects because of trucks lead to a strongly fluctuating performance, especially for long packet lengths and high vehicle speeds. Apart from that, they state that "the maximum data

volume that can be transmitted when a vehicle drives by a roadside unit is achieved at low data rates of 6 and 9 Mbit/s." In addition, [53] presents several measurements related to IEEE 802.11 a/b/g for vehicular environments and proves that the vehicle distance and the line of sight are of main importance for the performance as well. Besides an application level measurement study using the LinkBird-MX v3 unit produced by NEC has been carried out in [54].
To sum up, all the mentioned papers prove that (1) the number of packets to retransmit increases with a larger packet length and (2) that the communication ranges are reduced when a higher modulation order is applied.
Other studies focus on the different types of possible channels and the identification of possible scenarios or on the improvement of existing channel estimators that are also related to IEEE 802.11a for vehicular environments. [55] presents six small-scale fading models for real measurements using vehicles and analyzes their PER. These models are called (1) VTV (Vehicle to Vehicle) way Oncoming, (2) VTV Urban Canyon Oncoming, (3) RTV (Roadside to Vehicle) Suburban Street, (4) RTV Expressway, (5) VTV Expressway same direction with wall, and (6) RTV urban canyon. A detailed description of the scenarios can be found [56]. This article is based on the work presented in [55] and gives the results for the different scenarios with the help of a channel emulator. Apart from that, the differences between IEEE 802.11a and IEEE 802.11p in vehicular scenarios are enhanced. Their conclusion is that IEEE 802.11a performs worse because of "the high delay spread of the vehicular channels that introduce interference among symbols in case of 802.11a".

In the context of this report, we mainly focus on the physical layer implementation of the IEEE 802.11p receiver. Different software implementations of the standard have recently been described in [57] which focuses on the transceiver design, [58] where the physical layer of the whole IEEE 802.11p transceiver chain has been implemented as a Simulink version or [59] where an existing IEEE 802.11a Matlab simulation environment has been updated to deal with IEEE 802.11p. Besides [60] focuses on the simulation of the IEEE 802.11 physical layer implementation using the NS2 simulator. The latter is a network simulator that has originally been developed for networking research.

In addition to these academic publications, different commercial products supporting IEEE 802.11p became available during the past years:

- The **LinkBird-MX v3 unit produced by NEC** [61] embeds a LINUX machine which is based on a 64 bit MIPS processor working at 266 MHz. It can be configured either for reception or transmission and further contains two DCMA-82-N1 Mini-PCI cards with Atheros 802.11 radio chips.

- **NXP and Cohda Wireless** developed a flexible SDR implementation of WAVE called **MK3** [62]. It includes among others a GPS module, a CAN bus interface and Ethernet and is based on the NXP MARS platform which has been developed for the automotive context. The MARS platform consists of a combination of Tensilica Vector DSPs and hardware accelerators and can run several automotive standards simultaneously. While the physical layer and the real-time portions of the MAC layer are part of the MARS platform, the remaining MAC layer and the network layer are running on ARM11 processors.

- Another solution is the combination of the **WSU (Wireless Safety Unit) platform from DENSO and the Openwave Engine developed by BMW** [63]. Besides the physical layer implementation of IEEE 802.11p this transceiver supports the required MAC protocols for

US, Europe and Japan. Furthermore it includes CAN2.0, Ethernet and a 400 MHz power PC that can process one or two standards in parallel. The platform has been presented in [64].

- [65] provides an implementation of an **IEEE 802.11p frame-encoder based on GNU radio** [18] which has been **combined with USRP2** [19] and compares it to a former implementation based on a modified Atheros chipset. Although the results presented in this paper mainly focus on the comparison the authors have carried out, they further illustrate that the GNU radio receiver is fully compliant with the IEEE 802.11a/g/p standards.

Using the ExpressMIMO platform instead of any of the mentioned solutions has one important advantage. As this platform is not limited to the automotive context but potentially supports a wide range of different wireless communication standards as well as their multimodal processing, the integration of new standards results in best case only in a software update and not in a time consuming hardware redesign. This is important as despite the fact that experts currently focus on the combination of IEEE 802.11p with ETSI DAB, a combination of the first with LTE is strongly considered for future designs. A project that already examines how LTE and WLAN can be combined in an efficient way is CoCarX [66]. So it is only a matter of time till the work on first transceiver designs supporting both standards will start.

3.1.2 Contributions

The contributions described in this chapter are manifold. Main contribution is the presentation of an efficient physical layer implementation of the IEEE 802.11p receiver prototype for the ExpressMIMO platform. On the way to the final design first a Matlab model has been written to facilitate the receiver debugging before we implemented an emulation model of the receiver with the help of libembb. As the work on this emulation library was still ongoing by the time of our receiver development, the presented model is the first complete libembb design that has been implemented. Therefore it further served as a proof of concept for this library and as basis for a case study in which we focus on the multimodal execution of two different receivers on the ExpressMIMO platform. Based on our results, we derive first guidelines for an efficient scheduling and present a first scheduler prototype.

Apart from that the presented IEEE 802.11p receiver is the first prototype that is running on the ExpressMIMO platform and has been used in particular to identify possible bottlenecks when processing standards with short data sets on the platform. To reduce the latencies in the design, we further identified and implemented possible improvements when accessing the different resources on the hardware platform. These may be considered in a future version of the asynchronous application mode of libembb. In addition, we also derived a low latency design of a scheduler necessary when executing multiple DSPs in parallel.

3.2 Description of the IEEE 802.11p Packet Structure

Having a look at today's different wireless communication standards, one can distinguish between two different types: (1) frame based standards (e.g. LTE, DAB) and (2) packet based standards (e.g. WLAN). IEEE 802.11p is part of the second category. When developing a packet based transceiver for a multimodal system, one major disadvantage is that the arrival time of the next packet is not known in advance. This introduces an indeterminism requiring a flexible scheduler design in case multiple standards are processed simultaneously as we will detail later in this Chapter.

IEEE 802.11p is an OFDM standard, which means that its high data rate signal is split over several independent signals with lower data rates. These signals are transmitted over orthogonal frequencies to avoid ISI. Compared to other strategies, OFDM is easy implementable, has a lower ISI and offers a higher spectral efficiency due to the dense subcarrier spacing. On the other side, the Peak-to-Average power ratio of the transmitted signal is high. Thus the provision of a very accurate synchronization procedure to detect the beginning of the packet at the receiver side is unavoidable. The IEEE 802.11p OFDM symbols are composed of 80 sub-carriers. Please note that in the remainder of this document, one sub-carrier may also be denoted as a complex 'sample' with a width of 32 bit where real and imaginary part both have a size of 16 bit. Per OFDM symbol, 16 sub-carriers represent the guard interval which separates two neighbor OFDM symbols to avoid them to interfere with each other. These guard intervals are build using a cyclic prefix technique meaning that the guard interval is identical with the last part of the OFDM symbol. The remaining 64 sub-carriers contain 4 comb pilots needed for channel estimation / compensation, 12 nulled carriers and the transmitted information.

Once an OFDM symbol is received, the DFT transforms it to the frequency domain. In the transmitter the sub-carriers have been rearranged to match the inputs of the IDFT as shown in Fig 3.1. This has to be reverted on the receiver side.

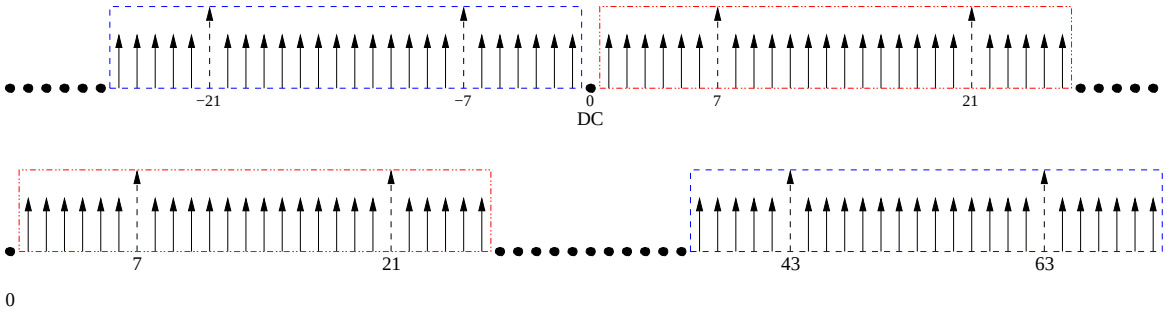


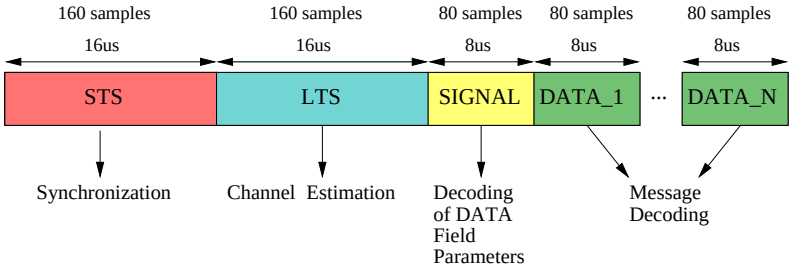
Figure 3.1: IEEE 802.11p OFDM Symbol Carriers before and after Reordering

The packet structure shown in Fig 3.2 is similar to the one of IEEE 802.11a. Each packet consists of a constant and a variable part. For a channel spacing of 10 MHz, the constant part has a length of 40 μs. It is composed of the Preamble and the SIGNAL Field:

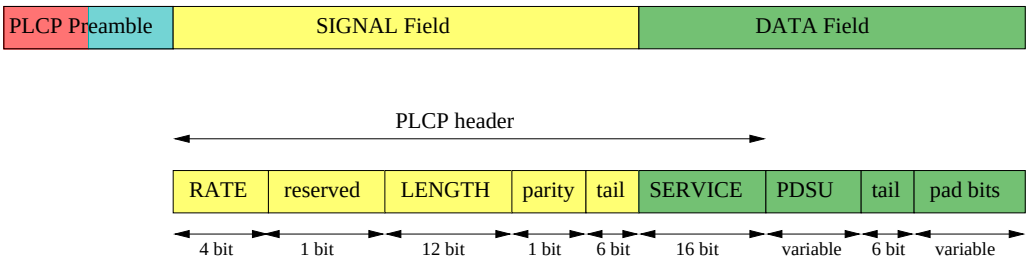
- **Short Training Symbol (STS):** The STS is part of the Preamble and is formed by 10 repetitions of the same 16 samples sequence. Each sequence has a length of 16 μs. The STS is required for the packet synchronization where the actual beginning of the packet is detected.

- **Long Training Symbol (LTS):** The LTS is part of the Preamble as well and consists of a guard interval of 32 samples and two identical OFDM symbols. These contain the block pilots that are needed for channel estimation. Two symbols are provided to improve the quality of the calculated channel estimate. The duration of the LTS is similar to the one of the STS (= 16 μs).

- **SIGNAL field:** The SIGNAL field specifies how to decode the transmitted message. It is BPSK modulated with a code rate of 1/2 and contains all required parameters for the subsequent DATA field detection. More specifically, the output of the Viterbi Decoder contains the information about RATE and LENGTH. RATE identifies the modulation scheme and the code rate of the DATA field, LENGTH corresponds to the the number of octets in the MPDU (MAC Protocol Data Unit) requested by the MAC layer. Further parameters have to be calculated or retrieved from LUTs based on these two values. These parameters are the code rate, the data bits per OFDM symbol (N_{dbps}), the number of transmitted symbols (N_{sym}), the number of data bits (N_{data}), the number of padding bits (N_{pad}), the number of carrier bits per symbol (N_{cbps}) and the parameter K_{mod} used for normalization in the mapper. An overview of the modulation dependent parameter values is provided in Table 3.1.



(a) IEEE 802.11p Packet (Channel Spacing of 10 MHz)



(b) IEEE 802.11p Packet with a Detailed View on the SIGNAL and the DATA Field

Figure 3.2: IEEE 802.11p Packet Structure

In contrast to the constant part of the packet, the **DATA field** consists of a variable number of OFDM symbols. Its length is not known before the decoding procedure of the SIGNAL field is finished. The contained LENGTH parameter can vary between 1 and 4095 which results in a DATA field length of 1 to 1366 OFDM symbols. All decoded SIGNAL field parameters apply on the whole DATA field and may not change before the next packet is received. The time between the end of one packet and the reception of the following one is at least 10 μs. Table 3.2 shows the modulation parameters for the applied 10 MHz channel spacing.

	Code Rate	N_{bpsc}	N_{cbps}	N_{dbps}	Data Rate (Mb/s)
BPSK	1/2	1	48	24	3
BPSK	3/4	1	48	36	4.5
QPSK	1/2	2	96	48	6
QPSK	3/4	2	96	72	9
16-QAM	1/2	4	192	96	12
16-QAM	3/4	4	192	144	18
64-QAM	2/3	6	288	192	24
64-QAM	3/4	6	288	216	27

Table 3.1: Modulation dependent Parameters decoded in the SIGNAL Field

Parameter	Value
Number data sub-carriers	48
Number pilot sub-carriers	4
Subcarrier frequency spacing	10 MHz/64
Packet length	1 - 1366 DATA symbols
Modulation Schemes	BPSK, QPSK, 16/64-QAM
Code rates	1/2, 2/3, 3/4
Data rates	3, 4.5, 6, 9, 12, 18, 24, 27 Mb/s

Table 3.2: IEEE 802.11p Specification Parameters (10 MHz Channel Spacing)

3.3 IEEE 802.11p Receiver Algorithms

This section describes the different receiver algorithms applied to decode the IEEE 802.11p packet. To improve the overall performance of the design, the FEP operations are performed on the whole OFDM symbol including all 64 sub-carriers. The removal of the nulled carriers and the reordering of the remaining ones are done by the Deinterleaver.

3.3.1 Packet Synchronization

A disadvantage of OFDM systems is their high sensitivity to timing and frequency synchronization errors. Therefore the packet synchronization algorithm used to detect the beginning of the packet has to be chosen properly to lower the PER. As the moment in time a packet arrives is not known in advance, the incoming samples have to be analyzed continuously making this algorithm the most latency critical one in the whole design. This is in opposite to frame based standards where the beginning of the frame has to be detected only once at the beginning of the receiver processing. Possible techniques for packet synchronization are auto- or cross-correlation. The preamble of the IEEE 802.11p receiver is suitable to both but [67] has shown that the cross-correlation performs slightly better. To speed up processing, we decided to combine an energy detector with an overlapping DFT-based correlator. The energy detector comprises only one function that can be expressed as

$$E(X) = \sum_{i=0}^{255} |r_x[i]|^2 \tag{3.1}$$

with r_x as the received signal.

The operations to be performed in the FEP are listed in Table 3.3. The number of cycles are calculated using the equations provided in Chapter 2.3. To recall: *direct* DMA transfers denote memcopy operations where LEON3 directly accessed the related memory regions without invoking the DMAs.

Function	DSP / DMA	samples	bytes	cycles
sum computation	FEP	256	1024	140
fep2leon	direct	1	4	11

Table 3.3: Energy Detection Operations

The subsequent packet synchronization is performed over the known reference STS denoted as STS_{ref}. As the size of the DFT window has been set to 256 due to the STS length of 160, the STS has to be zero extended before its DFT can be computed. This can be done offline before the receiver is started. The DFT window is shifted by the size of one OFDM symbol (80 samples). Maximum possible is a shift of 96 samples which corresponds to the window size minus the size of the STS. So it can be guaranteed that there is always one window which contains the whole received STS.
The received signal after the DFT, R_x, can be expressed as

$$R_x = (e^{j\Phi_n} H \times X_n) + Z_n \tag{3.2}$$

X_n are the signal components, Z_n the noise components and $e^{j\Phi_n} H$ the channel based error component. The packet synchronization algorithm can then be stated as

$$q_k = IDFT(R_x * DFT(STS_{ref})^*) \tag{3.3}$$

In case the result of this step is beyond a predefined threshold, the resulting timing offset i_s can be computed as

$$i_s = argmax(|q_k|^2) \tag{3.4}$$

i_s corresponds to the maximum absolute value of the cross-correlation of the symbol timing estimate. The operations to be performed for the packet synchronization are listed in Table 3.4.

Function	DSP / DMA	samples	bytes	cycles
DFT	FEP	256	1024	143
CWP	FEP	256	1024	139
IDFT	FEP	256	1024	143
energy/max/argmax	FEP	256	1024	143
fep2leon	direct	1	4	11

Table 3.4: Packet Synchronization Operations

The energy detection and the packet synchronization are performed by the FEP only, as depicted in Fig. 3.3. The comparison to an energy threshold and thus the decision of whether the receiver proceeds to channel estimation is currently in the responsibility of the main CPU but may be delegated to the UC or to a microprocessor or sequencer on the baseband side in a future version of the receiver prototype.

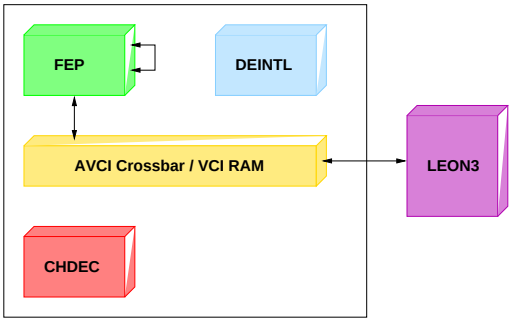


Figure 3.3: Preamble Data and Control Flow

3.3.2 Channel Estimation

In vehicular systems, the channels are strongly time-varying due to correlated fading caused by the multipath propagation. This results in a variation of the amplitude and / or the relative phase in the received signal. Therefore the channel estimation is very important. Like IEEE 802.11a, IEEE 802.11p defines two different pilot patters: block and comb pilots. The block pilots are carried by the LTS while the four comb pilots are included in each of the OFDM symbols of the SIGNAL and the DATA field. They can be found at positions -21, -7, 7 and 21. Specific for IEEE 802.11p is, that the polarity of the comb pilots changes between the OFDM symbols. The sequence defining this polarity can be generated from the scrambling sequence (generator polynomial $S(x) = x^7 + x^4 + 1$) when the all one initial state is applied. In addition, all 1s have to be replaced by -1s and all 0s by 1s. The first element of the obtained sequence defines the polarity of the comb pilots in the SIGNAL field, the subsequent ones are used for the DATA field.
A good paper about the different types of channel estimators is [68]. It proves that the best performance is achieved by the comb-type channel estimator, followed by the block-comb-type channel estimator. The first one uses the four comb pilots and obtains the values for the remaining carriers via interpolation techniques. As interpolation in general comes with a considerable computation effort we decided to rely on the block-comb-type channel estimator. Applying this kind of channel estimator, a first channel estimate is calculated based on the block pilots and the four comb pilots are used for a subsequent amplitude and phase correction. To do so it is assumed that the channel does not change during the reception of one packet. Compared to the comb-type channel estimator, the block-comb-type channel estimator reduces the computational complexity while still achieving a good performance.

The channel estimate based on the LTS block pilots denoted as $\hat{H}$, is calculated once for the whole packet:

$$\hat{H} = DFT(LTS_{received}) \times DFT(LTS_{reference})^* \tag{3.5}$$

Like for the packet synchronization, the computations listed in Table 3.5 are carried out by the FEP.

Function	DSP / DMA	samples	bytes	cycles
FFT	FEP	64	256	39
CWP	FEP	64	256	43

Table 3.5: Channel Estimation Operations

3.3.3 SIGNAL and DATA Field Detection

SIGNAL and DATA field detection are carried out by FEP, Deinterleaver and Channel Decoder (Fig. 3.4).

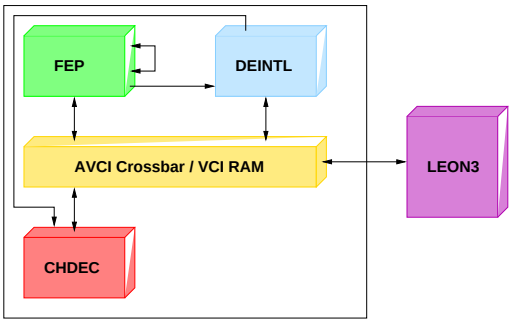


Figure 3.4: SIGNAL and DATA Field Data and Control Flow

The SIGNAL field is included in the constant part of the packet. Its detection procedure has to be finished before starting the DATA field detection, as the parameters describing the latter are to be extracted from the SIGNAL field. As stated in Chapter 3.2 these parameters comprise the number of OFDM symbols contained in the DATA field, the modulation scheme (BPSK, QPSK, 16-QAM, 64-QAM), the code rate (1/2, 3/4, 2/3), the number of sub-carriers, etc. While for the SIGNAL field detection, the three DSPs have to be executed one after another, they can be executed in parallel for the DATA field detection but by operating on different OFDM symbols. By the time the results presented in this report have been obtained, the Channel Decoder operated on the complete received message before the result was transferred to the MAC layer. In the future, tail-biting will be included so that this DSP can operate on smaller vectors and can be executed in parallel to the FEP and to the Deinterleaver. The main difference of the DATA field OFDM symbols when compared to the SIGNAL field is that the latter is always BPSK modulated with a code rate of 1/2 while the first can be modulated with four different modulation schemes, each of them exhibiting two different code rates.

3.3.3.1 Channel Compensation (FEP)

The channel compensation comprises the multiplication with the channel estimate as well as further corrections based on the comb pilots. The latter can be expressed as

$$R_{d,n} = (A(H)e^{-j\hat{\Phi}_n} \times R_x) \tag{3.6}$$

with

$$A(H)e^{-j\hat{\Phi}_n} = \frac{1}{4} \sum_{i=-21,-7,7,21} R^*_{p,i} R_{n,i} \tag{3.7}$$

and

$$\hat{A}(H) = \sum_{i=-21,-7,7,21} |\hat{H}_i|^2 \tag{3.8}$$

$R_{n,i}$ are the received and $R^*_{p,i}$ the known comb pilots. $A(H)e^{-j\hat{\Phi}_n}$ is used to correct the phase offset. This approximation still contains an unknown energy level term stated as $\hat{A}(H)$. Its correction is only necessary for 16/64-QAM demodulation where the calculation of the remaining bits is

based on $R_{d,n}$. One drawback identified in the design is that the result of the sum is always stored as a complex 64 bit value which is distributed over two consecutive 32 bit memory entries in the FEP MSS. Currently this result is modified in the main CPU to a complex 32 bit value that can be used for further processing in the FEP.
The operations to be performed for the channel compensation are listed in Table 3.6.

Function	**DSP / DMA**	**samples**	**bytes**	**cycles**
CWP	FEP	64	256	43
fep2fep (4 times)	direct	4*1	4*4	76
dot product	FEP	4	16	44
fep2leon (2 times)	direct	2*1	2*4	22
leon2fep	direct	1	8	64
CWP	FEP	64	256	43

Table 3.6: Channel Compensation Operations

For the SIGNAL field, the real part of $R_{d,n}$ directly serves as input for the Deinterleaver. No additional data detection is required. To sum up, the FEP operations needed for the SIGNAL field detection are summarized in Table 3.7.

Function	**DSP / DMA**	**samples**	**bytes**	**cycles**
FFT	FEP	64	256	39
Channel Compensation	*FEP*			
fep2deintl	FEP DMA	64	64	32

Table 3.7: SIGNAL Field Detection Operations (FEP)

3.3.3.2 Data Detection (Decoding, FEP)

The data detection for IEEE 802.11p can be done by the FEP and does not require the Mapper DSP engine.

- For **BPSK**, only the real part of $R_{d,n}$ serves as input to the Deinterleaver.
- For **QPSK**, real and imaginary part of $R_{d,n}$ are both significant.
- For **16/64-QAM**, the missing bits are calculated as a function of $R_{d,n}$ as stated in [69].
  For 16-QAM, only two remaining bits have to be calculated

$$R_{d2,n} = \frac{2}{\sqrt{10}}(\hat{H} \times \hat{H}^*) * \hat{A}(H) - abs(R_{d,n}) \tag{3.9}$$

  while for 64-QAM, four bits are missing

$$R_{d2,n} = \frac{4}{\sqrt{42}}(\hat{H} \times \hat{H}^*) * \hat{A}(H) - abs(R_{d,n}) \tag{3.10}$$

$$R_{d3,n} = \frac{2}{\sqrt{42}}(\hat{H} \times \hat{H}^*) * \hat{A}(H) - abs(R_{d2,n}) \tag{3.11}$$

The root term in the equations is already known after the SIGNAL field detection (parameter referred to as K_{mod}).
The result of the multiplication of the root term with $\hat{A}(H)$ and $\hat{H} \odot \hat{H}^*$ does not change during the whole DATA field detection and can be computed once after the SIGNAL field detection is finished. The parameter K_{mod} is used for normalization in the transmitter where it is multiplied with the output values to achieve the same average power for all schemes. K_{mod} depends on the modulation scheme. For BPSK it is set to 1, for QPSK to $1/\sqrt{2}$ for 16-QAM to $1/\sqrt{10}$ and for 64-QAM to $1/\sqrt{42}$.

$R_{d,n}$, $R_{d2,n}$ and $R_{d3,n}$ can directly be written into the Deinterleaver MSS. The required bit reordering as illustrated in Fig. 3.5 is performed by the Deinterleaver.
Only the abs term has to be calculated at runtime while the other factor can already be prepared once K_{mod} is known. The required functions for 16/64-QAM are listed in Table 3.8.

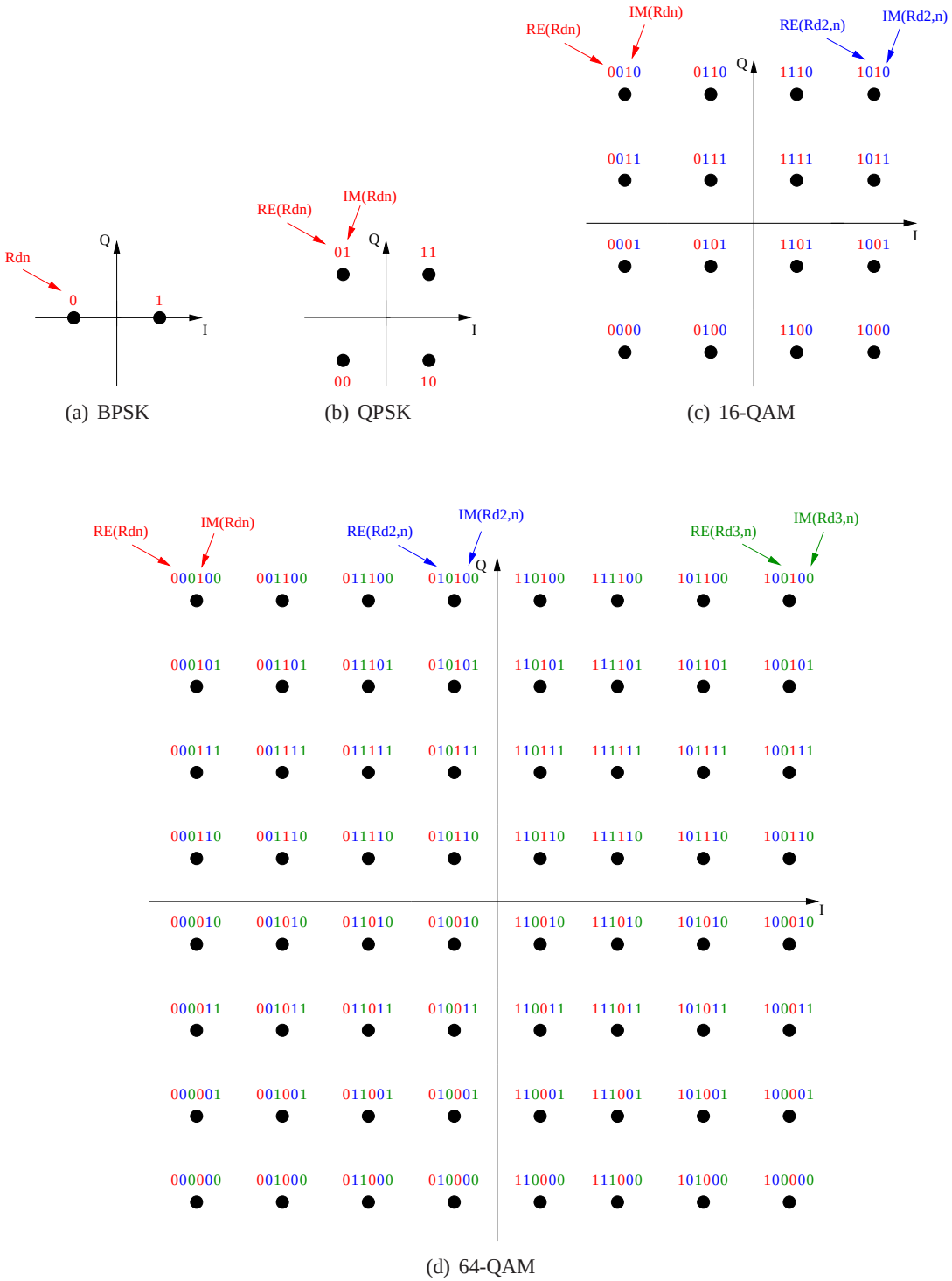
Function	DSP / DMA	samples	bytes	cycles
16-QAM				
fep2leon	direct	4*1	4*4	44
fep2fep	direct	64	256	129
CWP	FEP	64	256	43
leon2fep	direct	1	4	8
CWP	FEP	64	256	43
additional 64-QAM				
leon2fep	direct	1	4	8
CWP	FEP	64	256	43

Table 3.8: Data Field Initialization Operations

The data detection operations for the different modulation schemes are listed in Table 3.9. For BPSK and QPSK, the result of the channel compensation is taken.
The resulting DATA field detection operations are listed in Table 3.10. The copy operations of the result are split over several transfers as the results are stored in different banks of the FEP MSS.

Function	DSP / DMA	samples	bytes	cycles
16-QAM				
type change	FEP	64	256	43
absolute value	FEP	64	256	43
CWA	FEP	64	256	43
64-QAM				
type change	FEP	64	256	43
absolute value	FEP	64	256	43
CWA	FEP	64	256	43
type change	FEP	64	256	43
absolute value	FEP	64	256	43
CWA	FEP	64	256	43

Table 3.9: Data Detection Operations



(a) BPSK (b) QPSK (c) 16-QAM

(d) 64-QAM

Figure 3.5: Bit Constellations for the IEEE 802.11p Data Detection

Function	DSP / DMA	samples	bytes	cycles
FFT	FEP	64	256	39
Channel Compensation	*FEP*			
Data Detection	*FEP*			
BPSK				
fep2deintl	FEP DMA	64	64	32
QPSK				
fep2deintl	FEP DMA	64	128	40
16-QAM				
fep2deintl (2 times)	FEP DMA	2*64	2*128	80
64-QAM				
fep2deintl (3 times)	FEP DMA	3*64	3*128	120

Table 3.10: DATA Field Detection Operations

3.3.3.3 Deinterleaver

The Deinterleaver operates on a block of 8 bit samples and reverts the two permutations applied by the transmitter to reduce the impact of burst errors by distributing the affected bits over the received sub-carriers. It is stated in the standard specification [45] that the first permutation in the transmitter ensures that adjacent bits are modulated onto non-adjacent sub-carriers while the second permutation in the transmitter ensures that the adjacent bits are mapped onto less and more significant bits of the constellation.
The first permutation of the Deinterleaver is defined by

$$i = s * floor(j/s) + (j + floor(16 * j/N_{cbps})) * mod(s), \quad j = 0, 1, ..., N_{cbps} - 1 \tag{3.12}$$

with $s = max(N_{bpsc}/2, 1)$.
The second permutation is defined by

$$k = 16 * i - (N_{cbps} - 1)floor(16 * i/N_{cbps}), \quad i = 0, 1, ..., N_{cbps} - 1 \tag{3.13}$$

The deinterleaving is performed before the Channel Decoder where the number of bits is decreased by one half. The size of the permutation table per OFDM symbol is thus $2*N_{dbps}$. For our receiver, the final permutation tables do not only consider the two permutations defined above. Instead, they also remove the nulled carriers, bring the carriers in the right order, reorder the bits as required by the data detection and insert zeros is case the code rate is not set to 1/2. In the transmitter, higher rates (2/3 and 3/4) are achieved by puncturing where some of the encoded bits are omitted in the transmitter. To revert this effect, the Deinterleaver can be used as well.
For an efficient generation of the permutation tables we provide a C code that generates them for receiver emulation and for the real hardware processing. The difference between the two modes is that in the first case a simple 16 bit array is sufficient to include the tables in the design while for the latter, a 32 bit representation is required so that the tables can be DMA transferred to the baseband side before the receiver is started.

3.3.3.4 Channel Decoder

The Channel Decoder detects the codeword that has been transmitted based on the result provided by the Deinterleaver. This means that the codeword is mapped back to the corresponding information bit sequence. In the transmitter a convolutional encoder is used. The generator polynomials

of the encoder are $g_0 = 133_8$ and $g_1 = 171_8$ and the constraint length K is set to 7. Per symbol, the Channel Decoder takes $2 * N_{dbps}$ input symbols a 8 bit to compute N_{dbps} output bits. At the time when we obtained the results presented in this thesis, the Channel Decoder had been executed only once after all Deinterleaver operations has been finished. The new version instead will contain tail-biting.

3.3.3.5 Descrambling and CRC Check

Before the output of the Channel Decoder is forwarded to the MAC layer implemented on the host PC, the sequence has to be scrambled and CRC checked. In the transmitter, the bits were scrambled to randomize the data pattern. The work of a timing recovery circuit is thus simplified and dependencies between the signal's power spectrum and the transmitted data are eliminated. The generator polynomial for the IEEE 802.11p (de)scrambler is $S(x) = x^7 + x^4 + 1$. When descrambling the starting point of the sequence are the six LSB of the SERVICE field (Fig 3.2). The sequence is then periodically repeated.

3.4 System Presentation and Receiver Versions

3.4.1 Required Resources on the ExpressMIMO platform

As shown previously, only VCI RAM, FEP, Deinterleaver and Channel Decoder are needed to decode the packets of the IEEE 802.11p receiver (Fig. 3.6). The Preprocessor will be included in a future version of the design. After each interrupt, the Preprocessor will copy 640 complex samples a 32 bit into the circular input buffer memory being part of the FEP MSS. This corresponds to a memcopy operation of eight OFDM symbols. Including the Preprocessor will not change the presented results in this chapter as the copy operation will be handled by a local UC on the baseband side and not by the main CPU.

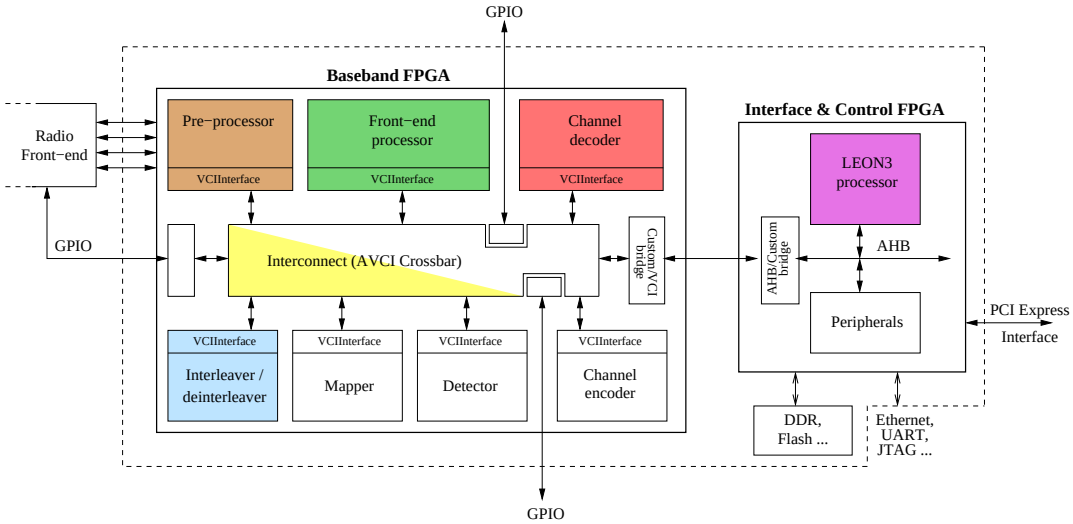


Figure 3.6: Baseband Architecture of the ExpressMIMO Platform

3.4.2 Matlab Prototype of the IEEE 802.11p Receiver

A first version of the IEEE 802.11p receiver has been implemented in Matlab for fast algorithm validation. To generate the test signals, a Matlab transmitter code provided by the Telecommunications Research Center Vienna [70] has been used. In addition, different real snapshots provided by BMW have also been tested to validate the chosen algorithms of our design.

3.4.3 Emulation of the IEEE 802.11p Receiver

The emulation prototype of the IEEE 802.11p receiver has been designed with a sequential execution in mind. Thus it does not fully exploit the possible concurrency of the DSPs on the platform. Currently, the receiver emulation is considered untimed. That is why concurrency is not yet meaningful. Instead, receiver emulation is important to identify the required DSP functions and the way how they are programmed (= how the control register parameters have to be set) in a pure software environment which speeds up the design flow significantly.
To enable a simple integration of other standards in case of multimodal processing and to simplify updates due to changes in the baseband and thus in libembb, we included an additional layer between the receiver code and libembb called `expressmimo_emu` (Fig. 3.7). So the receiver calls only the functions defined in `expressmimo_emu` which then calls the libembb routines.

Figure 3.7: Emulation Codestructure

In the following a code example for a DFT operation is given. The function call in the receiver code looks as follows:

```
emm_fep_fft(256, DFT_ADDR_in, DFT_ADDR_out, 0);
```

The first value indicates the size and the last value if a DFT or an IDFT has to be performed.
The function definition can be found in `expressmimo_emu`:

```
void emm_fep_fft (uint32_t size, uint32_t src, uint32_t dst,
                  uint32_t inverse)
{
  FEP_CONTEXT *ctx = &ctxf;
  FEP_FFT;
  fep_start(ctx);
}
```

The `FEP_CONTEXT` is an enumerate that contains pointer to all the control and status registers of the FEP. `FEP_FFT` is a macro where all the necessary parameters using libembb routines are set:

```
#define FEP_FFT { \
  uint32_t src_addr_index, dst_addr_index; \
  uint32_t src_memquarter, dst_memquarter; \
```

```
  src_addr_index = get_addr_index_fft(offset_src); \
  dst_addr_index = get_addr_index_fft(offset_dst); \
  src_memquarter = get_fep_mss_bank(offset_src); \
  dst_memquarter = get_fep_mss_bank(offset_dst); \

  fep_set_l (ctx, size); \
  fep_set_i (ctx, inverse); \
  fep_set_bx(ctx, src_addr_index); \
  fep_set_bz(ctx, dst_addr_index); \
  fep_set_qx(ctx, src_memquarter); \
  fep_set_qz(ctx, dst_memquarter); \
  fep_set_wx(ctx, 3); \
  fep_set_wz(ctx, 3); \
  fep_set_op(ctx, FEP_OP_FT); \
}
```

Due to the applied local addressing scheme of the whole FEP MSS, the FEP MSS bank and the related offset inside the bank have to be identified before setting the parameters.
The emulation prototype of the IEEE 802.11p receiver supports all different modulation schemes and code rates. It has been annotated by cycle counts and extended by the generation of trace files for an efficient receiver evaluation. Apart from that it automatically generates files that can be used to plot (intermediate) results in Matlab or Octave. All these enhancements allow an easy validation of the receiver and the identification and implementation of necessary algorithmic improvements in a pure software environment.

3.4.4 Hardware Prototype of the IEEE 802.11p Receiver

In the future design flow, the code written for emulation can directly be compiled for the hardware platform even for standards with short data sets. Currently each parameter setting function (e.g. `fep_set_l(ctx, size)`) first reads the related register value from the FEP control registers, modifies the bits that corresponds to the parameter in the main CPU and writes the register value back. As this is done for each of the parameters, this procedure is very time consuming and ends up at an average programming time of around 425 ns per parameter. If one imagines that a standard FEP operation requires at least 14 parameters it is obvious, that this procedure is too time consuming in case of strong latency requirements. To satisfy these strong real-time constraints, the emulation code has therefore been revised and optimized several times before being ported on the ExpressMIMO platform. The improvements included the choice of an appropriate OS for the main CPU (more specifically LEON3), a flexible scheduler to execute the different DSP engines simultaneously, grouping of DATA OFDM symbols and the generation of command words offline before the receiver is started.

3.4.4.1 C Code Optimization

Performance improvements can already be obtained by optimizing the C code running on the main CPU to decrease the overhead due to functions calls after compilation. Modifications include a higher number of inline functions and macros as well as a limited number of parameters set dynamically at runtime.

3.4.4.2 Interrupt Handler

The interrupt handler provided by MutekH is very fast and efficient. The time measured on the platform from the moment an interrupt is raised on the baseband side till the main CPU reacts and continues with the next assembly command takes only about 2.3 μs. This time is negligible for common standards but not for standards like IEEE 802.11p as the overhead caused by the interrupt handler results in a significant performance drop (recall: the duration of one OFDM symbol is 8 μs). An alternative to interrupts is to poll the status registers of the DSP engines. In this case the time the main CPU takes to continue with the next assembly command is only about 436 ns.

3.4.4.3 Command Preparation at Runtime

The standardized DSP shell supports the preparation of the next command while the PU is still busy. As soon as a command is prepared, the pending flag in the control registers is set to one. It depends on the data dependencies of the code where this feature can be used. In case of the energy detection, for instance, command preparation is not possible as the result of the FEP operation is required for the decision of how to proceed in the program flow. Instead it can be observed for the packet synchronization that this feature lowers the communication overhead significantly. While the DFT is running, the CWP command can already be programmed. And while the CWP is running, the IDFT command can be prepared, etc. The communication flow is thus executed in parallel to the DSP processing and the receiver performance of these operations is only limited by the processing time of the DSP and not by the communication overhead any more.

3.4.4.4 Scheduling

When compared to other wireless communication standards, the IEEE 802.11p standard is extremely latency critical due to the short time available till the acknowledgement packet has to be sent. That is why we abandoned the overhead that the usage of POSIX threads would have entailed. Instead, we relied on a very simple thread model where one thread is assigned to each concurrent platform entity, such as the Processing Units and the DMA engines.

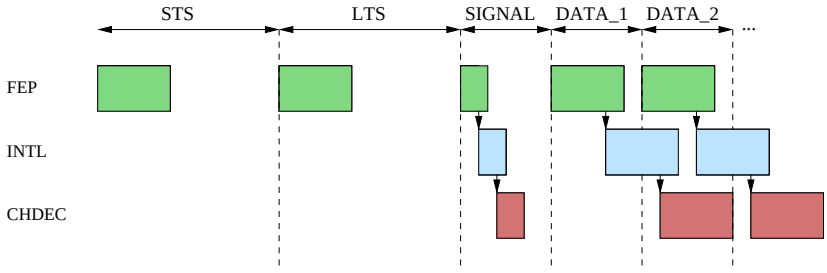
Per thread, two different data items are checked and updated:

- a *position pointer* to the next code function that shall be executed. This pointer is updated when the control flow moves on to the next potentially blocking activity on the platform which is either a DSP or a DMA operation.
- a *condition pointer* to a memory location indicating if the task is runnable or not (e.g. busy flag of the DSP).

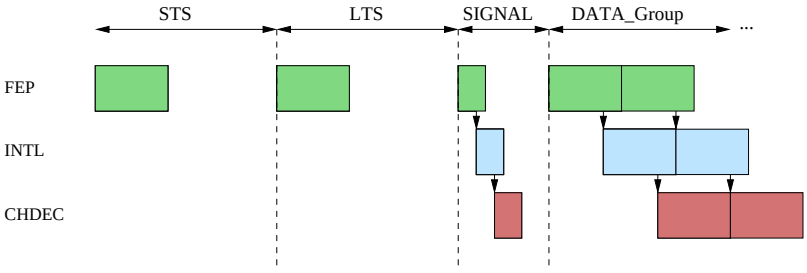
The question whether a thread can be executed depends on the status of the DSP engine. If the DSP is ready, the thread is runnable immediately while it is blocked when the DSP is in busy state. For the moment the scheduler is based on a Round Robin policy but it could be an interesting task to experiment with different scheduling policies in the future.
Fig. 3.8 illustrates the scheduling of the different DSP engines. Once, the results of a DSP are available, they are copied into the MSS of the subsequent DSP. During this time, both DSPs are busy. In contrast to the SIGNAL field, the structure of the DATA field allows to decode different data symbols in parallel. The processing flow depends on the number of DATA OFDM symbols available in the FEP MSS:

1. Assuming the case that the Preprocessor provides one DATA OFDM symbol as soon as it is received, the scheduler has to wait till the next symbol is available before it can schedule the next IEEE 802.11p task to the FEP (Fig. 3.11(a)).

2. When more DATA OFDM symbols are available in the FEP MSS, the FEP can be started again once it finished the decoding procedure of one symbol (Fig. 3.11(b)). The time the different DSP engines are busy will increase as the scheduler has to take care of the FEP and the Deinterleaver tasks in parallel.



(a) DSP Engine Scheduling when only one DATA OFDM Symbol is available



(b) DSP Engine Scheduling when several DATA OFDM Symbols are available

Figure 3.8: IEEE 802.11p DSP Processing and Scheduling

3.4.4.5 Symbol Grouping

Another possible optimization in the design flow is the grouping of DATA OFDM symbols for the FEP so that the FEP can operate on larger vectors. It is obvious that the required communication overhead is strongly correlated with the number of DATA OFDM symbols that are grouped. The more symbols are grouped, the less the communication overhead. The choice of the maximum group size depends on the number of OFDM symbols supplied by the Preprocessor per *acquisition cycle*. An acquisition cycle corresponds to the time till X samples are stored in the output FIFO of the Preprocessor MSS. Currently the maximum group size has been set to eight. The operations to be performed in the FEP for each OFDM symbol comprise DFT, channel compensation and data detection. Compared to the case when symbol per symbol is processed, 28 FEP commands can be saved for the considered maximum group size.
The actual group size applied at runtime is determined automatically. It is set to the minimum between the number of OFDM symbols available in the FEP MSS and the maximum group size. This ensures, that the FEP is always processing and not blocked in case not enough OFDM symbols are provided. Table 3.11 recalls the FEP operations presented in Table 3.6, Table 3.9 and

Table 3.10 and shows which operations are currently grouped.
To continue the processing over the grouped output vectors of the FEP, the Deinterleaver permutation table generation has been modified to enable a Deinterleaver processing of the whole grouped result vector. Per possible group size one permutation table is provided due to the different sizes of the input vectors. At runtime, the decision which permutation table to take is chosen dynamically. In a future version of the receiver, the same will apply for the Channel Decoder.

Function	**DSP / DMA**	**grouping**
FFT	FEP	no
CWP	FEP	yes
fep2fep (4 times)	direct	no
dot product	FEP	no
fep2leon (2 times)	direct	no
leon2fep	direct	no
CWP	FEP	yes
16-QAM		
type change	FEP	yes
absolute value	FEP	yes
CWA	FEP	yes
64-QAM		
type change	FEP	yes
absolute value	FEP	yes
CWA	FEP	yes
type change	FEP	yes
absolute value	FEP	yes
CWA	FEP	yes
DMA transfers to DEINTL		
fep2deintl	FEP DMA	yes

Table 3.11: DATA OFDM Symbol Grouping

3.4.4.6 Command Preparation before starting the Receiver

An efficient way to optimize the centralized control flow is to prepare the commands and to store them in a memory on the control side before starting the receiver. When applying a centralized control flow, the DSP engines are all programmed by LEON3. Per parameter, the programming time takes around 425 ns. In case only one parameter has to be written, the communication overhead is negligible, but usual FEP commands, for instance, comprise at least 14 different parameters. This results in a total programming time of at least 6 μs per operation. When preparing the commands in advance, the required programming time at runtime is significantly reduced to 70 ns per 32 bit command register. In case of the FEP which has six of these registers, the programming time is therefore decreased from around 6 μs to 420 ns. Fortunately most of the IEEE 802.11p commands are static and can easily be prepared. In case a parameter is set dynamically at runtime, the performance gain depends on the number of parameters to be set and how they are distributed over the 32 bit command registers. For the IEEE 802.11p receiver, at maximum one parameter is set dynamically per operation. The related timing overhead is negligible as only one additional assembly command is required for this operation. For the command preparation, the same macros than for the receiver emulation are used.

3.5 Results

The presented results have been obtained with the emulation prototype of the IEEE 802.11p receiver and by a cycle-accurate HW/SW co-simulation. Prior to that, the whole receiver chain has been validated on the hardware platform itself for a reference frequency of 100 MHz. The results have been retrieved using the JTAG and the PCIexpress connection. To achieve a higher performance this frequency will be increased in the near future. The maximum achievable frequency considering the FPGA target is determined by the main CPU which can be processed up to 133 MHz.
To get exact figures about the receiver performance, different test signals have been generated for validation. First, test signals generated with the Matlab reference model have been used. These signals are based on the example provided in the annex of the standard specification and can be generated for any packet length, code rate and modulation scheme. Second, our receiver has been validated by testing different snapshots provided by our PLATA project partner BMW. These have been generated with the Densobox, NEC Linkbird and a SimTD motorcycle. All of them were QPSK modulated.

3.5.1 Remarks

- For the scheduler a Round Robin policy is applied.
- Results of the Channel Decoder are provided under the assumption that the tail-biting is already included in the current design. For the tests on the hardware platform the Channel Decoder is invoked only once, once the Deinterleaver results of all DATA symbols are available.
- The receiver is considered to be real-time capable if the processing time of the constant part is below 40 μs and if FEP, Deinterleaver and Channel Decoder take less than 8 μs per OFDM symbol when performing the DATA field detection.

3.5.2 Resource Consumption Results obtained with libembb

The emulation prototype of the IEEE 802.11p receiver gives a first insight in the resource consumption of the different DSPs with the aim to answer the following questions:

1. Which DSP is used most of the time?
2. How much processing time is required for DMA transfers?
3. Considering only the processing time, can the receiver be executed in real-time on the platform? If not, where are the bottlenecks?
4. Would it be possible to execute the receiver together with other transceivers? If not, where are the bottlenecks?

Thanks to the emulation environment libembb, these results can already be obtained at an early design stage and allow an easy improvement of the standard implementation in a pure software environment.

First results of the IEEE 802.11p receiver have already been presented in [2] but since then the design has further been optimized. Basically, the processing time of the receiver without considering the communication overhead can be split into two parts: (1) busy time of the DSPs and (2)

dead time. The latter is the time when no new receiver task can be scheduled as the end of the Preprocessor acquisition cycle is not yet reached. For multimodal processing, the dead time is of major importance as during these time slots tasks of another standard can be scheduled without decreasing the performance of the blocked one.
For our receiver we assumed that the Preprocessor collects 640 complex samples before they are transmitted to the FEP. This corresponds to an acquisition cycle duration of 64 μs. In case the beginning of the packet corresponds to the first sample in the 640 sample window, the constant part and three DATA OFDM symbols can be processed till the system has to wait for the next samples. There are two options to identify the time slot available to process another standard. Either to validate all possible scenarios for the 640 sample window or to rely on an earliest deadline first policy. The first is not an appropriate approach as the evaluation of all possible scenarios is a very time consuming procedure. Furthermore the results have to be reworked completely in case the reference frequency is modified. An example analysis using this method has been illustrated in the Deliverable D2.4 of the PLATA project. In case of the earliest deadline first policy, the deadline should be based not on every operation but on a group of operations. Suitable groups for IEEE 802.11p are the constant part and the OFDM symbol groups of the DATA field. This means that the processing of the constant part has to be finished within 40 μs while the size of the time slot available for the DATA field detection depends on the group size that is set dynamically at runtime.

Table 3.12 illustrates the execution and the memcopy runtime for the constant part for the applied frequency of 100 MHz. All times represent the busy times of the DSPs meaning that DMA transfers between the DSPs are considered twice as both DSPs are busy during that time. Applying a higher frequency than the chosen one (e.g. 1/2 faster) would result in a reduced processing time of 1/2 as no communication overhead is considered in this context. The results in the table illustrate that the total processing time is far below the available 40 μs. So around 25 μs are left for the communication with the main CPU.

Task	**DSP**	**memcopy**	**total**
energy detection	1.40 μs	0.11 μs	1.51 μs
packet synchronization	5.65 μs	0.11 μs	5.76 μs
channel estimation	0.82 μs	-	0.82 μs
Signal Field (FEP)	1.69 μs	1.94 μs	3.63 μs
Signal Field (DEINTL)	0.64 μs	0.98 μs	1.62 μs
Signal Field (CHDEC)	0.64 μs	0.63 μs	1.27 μs

Table 3.12: Task Runtime for the DSP Engines - Constant Part

In Fig. 3.9 illustrates the distribution of the processing time over the different resources. The FEP is required most of the time, followed by the DMA transfers on the baseband side. Deinterleaver and Channel Decoder operate only on vectors with a size of 48 samples and thus need only a small amount of the overall processing time.

For the DATA field detection, the actual processing time depends on the modulation scheme, the code rate, the symbol grouping and the number of symbols contained in the field, N_{sym}. Table 3.13 lists the different processing times for a group size of 1 (first value) and a group size of 8 (second value) for the different modulation schemes and code rates. For the latter, the average processing time per symbol is given. It can be seen that the FEP and the Deinterleaver execute their tasks within the required 8 μs while for 64-QAM with code rate 3/4 this is almost not possible. So we

see already at this step that either an optimization of the Channel Decoder or a higher frequency is necessary (when applying a frequency of 133 MHz, the requirements are fulfilled for all DSP engines).

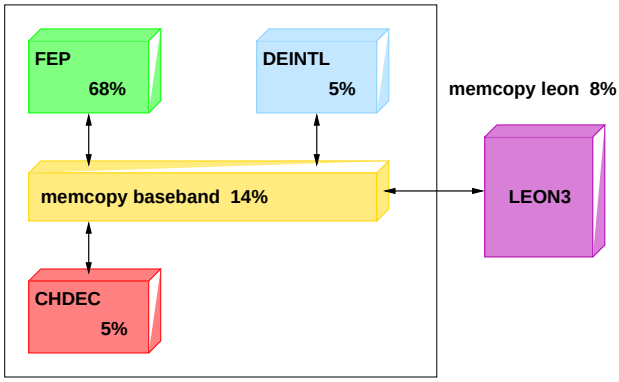


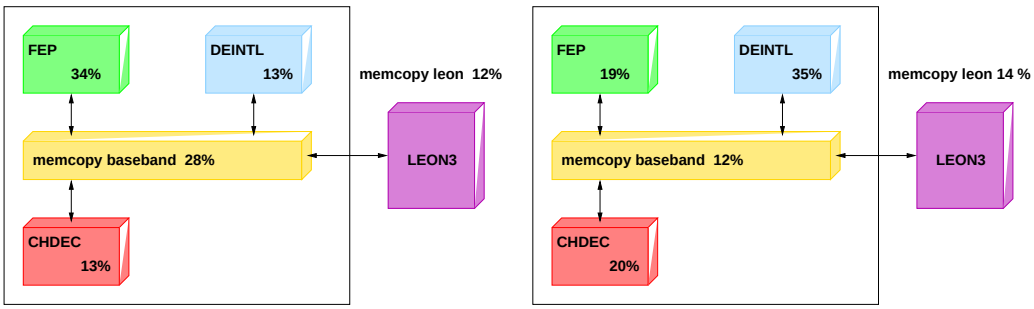
Figure 3.9: Runtime Distribution - Constant Part

DSP	**BPSK rate 1/2**	**BPSK rate3/4**	**QPSK rate 1/2**	**QPSK rate 3/4**
FEP	3.07 μs / 2.56 μs	3.07 μs / 2.56 μs	3.15 μs / 2.64 μs	3.15 μs / 2.64 μs
DEINTL	1.62 μs / 0.85 μs	1.95 μs / 1.184 μs	2.28 μs / 1.51 μs	2.94 μs / 2.17 μs
CHDEC	1.27 μs / 0.92 μs	1.76 μs / 1.36 μs	2.14 μs / 1.79 μs	3.01 μs / 2.66 μs
DSP	**16-QAM rate 1/2**	**16-QAM rate 3/4**	**64-QAM rate 2/3**	**64-QAM rate 3/4**
FEP (init)	2.67 μs	2.67 μs	3.18 μs	3.18 μs
FEP	4.84 μs / 3.83 μs	4.84 μs / 2.83 μs	6.53 μs / 5.02 μs	6.53 μs / 5.02 μs
DEINTL	4.08 μs / 2.17 μs	5.4 μs / 4.42 μs	7.12 μs / 5.93 μs	7.78 μs / 6.59 μs
CHDEC	3.88 μs / 3.53 μs	5.62 μs / 5.27 μs	7.36 μs / 7.01 μs	8.23 μs / 8.01 μs

Table 3.13: Task Runtime for DSP Engines (including memcopy) per DATA OFDM Symbol

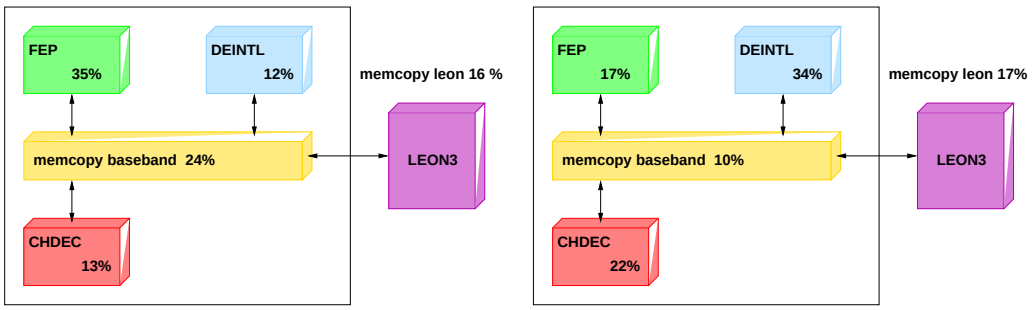
The results in the following present the runtime distribution for the minimum and the maximum group size. For the Deinterleaver processing, the worst case is assumed when the permutation table is copied from the VCI RAM into the Deinterleaver MSS for each DATA symbol group. This effect mainly occurs for short packet lengths where the group size is not always constant while for a long packet length it can be assumed, that the permutation table is copied only three times (in worst case): for the first / last group and for the remaining groups with the length of the max group size. Not copying the permutation table reduces the busy time of the Deinterleaver by around 17 %.
Fig. 3.10 and Fig. 3.11 show the runtime distribution results for the minimum case (BPSK, coding rate 1/2) and the maximum case (64-QAM, coding rate 3/4). While for BPSK the FEP takes most of the resources, it is the Deinterleaver for 64-QAM. The reason for that can be found in the longer permutation table required for 64-QAM. Recalling Table 3.1, the number of data bits per symbol, N_{dbps} is 24 for the BPSK example and 216 for the 64-QAM example resulting in permutation table lengths of 48 and 512 entries for a group size of one. The results further illustrate that the resource distribution is almost similar for the two different group sizes. But this will change most

probably once the communication overhead is considered as well. What changes already is the overall processing time that has been reduced by 17 % for BPSK and by 13 % for 16-QAM per group. It is worth to mention, that the results of these figures are based on a sequential processing flow. The possible parallelism of the different DSPs is not considered yet.
When changing the overall packet length and thus the number of DATA OFDM symbols in the DATA field, the processing time depends on the size of the groups. If N_{sym} is a multiple of eight, the overall processing time is achieved by multiplying the shown results by eight. The percentage values will not change. If N_{sym} can be split in groups of eight plus a group with size x, the overall processing time will be eigth times the values related to a group of eight plus the values for the group x. The probability of a percentages change will depend on the overall size of the packet but in any case the percentage values will be between the two illustrated cases in Fig. 3.10 and Fig. 3.11.



(a) BPSK, coding rate 1/2 (b) 64-QAM, coding rate 3/4

Figure 3.10: Runtime Distribution - Data Field (group size = 1)



(a) BPSK, coding rate 1/2 (b) 64-QAM, coding rate 3/4

Figure 3.11: Runtime Distribution - Data Field (group size = 8)

3.5.2.1 Case Study: Multi-Standard Processing of IEEE 802.11p and ETSI DAB

A) Motivation

The combination of C2X communication (IEEE 802.11p) and TPEG information (ETSI DAB [44]) brings not only safety but also commercial benefits. Imaginable use cases of the latter are for example entertainment and comfort applications like local information about restaurants, internet in the car, TV, etc. The standards supporting these new applications are less latency critical and

reliability is not of such importance than for safety applications.
The ETSI DAB standard used for TPEG information has been developed between 1987 and 2000. Although the standard was not very successful at the beginning, the work on it has been restarted in Germany in 2011 in the context of DAB+. The aim is to develop first applications that can reuse the already existing infrastructure.

The necessary scheduling of IEEE 802.11p and DAB including the resource management for multimodal processing is still an open research topic. For that reason there is not so many information available yet. In a recent paper ([71]), an SDR control framework to provide the required coexistence services and necessary interface was presented. This framework is based on an SDR technology demonstrator presented in [72]. The goals of this demonstrator are manifold. It is not only used to prove that different wireless communication standards can be dynamically installed or started but also that they can run in a coordinate fashion. This means that all standards are real-time compliant and that the hardware resources can be shared in an efficient manner. In [71], the presented scheduling is performed by putting the operation request from the standards into a common timeline which is executed in small slices that are called *scheduling window*. Their selection is not static as it depends on the number of active channels which are the wireless communication standards to be processed in parallel. Further they identify three different types of scheduling tasks:

- A **rigid operation** is an individual operation whose time slot has a fixed start and end time. Their length does not exceed the scheduling window and the different operations do not depend on each other.
- A set of multiple rigid operations that have boolean relations like `and` or `or` are called **flexible operations**. As an example they mention a primary uplink/downlink slot pair (TX1 + RX1) which is followed by two backup slot pairs as applied for instance in Bluetooth.
- The last type of scheduling tasks are **continuous operations** which may exceed the window length. So they have to be scheduled piecewise.

The work presented in this context is based on the first type of scheduling tasks.

B) ETSI DAB Receiver Mapping on the ExpressMIMO Platform

The DAB [44] prototype has been developed at BMW and TUM and has recently been presented in [73]. The standard defines four different transmission modes where mode I is the most commonly used. Table 3.14 lists the modulation parameters for the applied transmission mode I. In contrast to IEEE 802.11p, DAB is a frame based standard where each frame has a duration of 96 ms.

Parameter	**Value**
frame duration	96ms, 76 Symbols
symbol duration (total, useful, guard)	1.246ms, 1ms, 246us
null symbol duration	1297ms
transmission bandwidth	1.536 MHz
OFDM type	2048-FFT, 1536 used
modulation	D-QPSK
bitrate	2.4 Mbps

Table 3.14: ETSI DAB Specification: Transmission mode I

The DAB frame structure is shown in Fig. 3.12. Each frame has a length of 96 ms and consists of a Synchronization Channel (SC), a Fast Information Channel (FIC) and a Main Service Channel (MSC). While the SC is required for basic demodulator functions (transmission frame synchronization,...) the FIC is needed for a fast information access and contains information on the MSC data such as labels, type, length or protection level. For transmission mode I it is composed of 12 Fast Information Blocks (FIB), each with a size of 256 bit. Audio and data service components are carried by the MSC. For transmission mode I it is build of 4 Common Interleaved Frames (CIF) where each CIF is composed of 864 so-called capacity units that have a size of 64 bits.

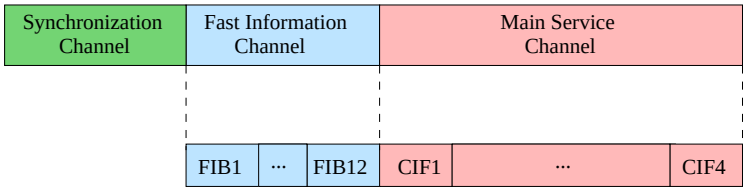


Figure 3.12: ETSI DAB Frame

The basic DAB receiver structure is sketched below. Going into too much details would go beyond the scope of this report. For more details about the different algorithms please refer to [74].

1. To lock to the DAB signal, an initial time synchronization (STI) and a coarse and fine frequency estimation (SFC, SFF) are performed. STI is executed only once by using a sliding window. Goal is to find the zeros that are separating the different frames. All these operations can be performed by the FEP, except the sliding window operation to determine the actual start of the frame and the $\arctan$ to find the actual frequency offset which are in the responsibility of the main CPU.

2. Once the synchronization is performed, changes caused by system variations have to be tracked. This is done by tracking time (TTI) and fine frequency offset (TFF). These operations are performed by the FEP as well, except for the determination of the frame start the actual frequency offset.

3. Next the fine frequency is corrected based on the estimate (MIX) and the OFDM symbols are demodulated (FFT, DEM). The DAB OFDM symbol consists of 1536 active carriers and is modulated using DQPSK (differential QPSK). For DQPSK demodulation, the carriers of the current OFDM symbol are just multiplied with the carriers of the previous one. The OFDM demodulation (FFT) is thus performed using a 2048-DFT. For both operations the FEP is needed.

4. For frequency deinterleaving (FDI) which is performed over one OFDM symbol the Deinterleaver is taken. Instead, the time deinterleaving to reduce the fading channel effects runs on the main CPU. The reason for that is that this operation is performed over the span of several frames (total depth of 384 ms). This size does not fit into the MSS of the Deinterleaver. The time deinterleaver is not needed when decoding the FIC.

5. Unpuncturing (PNT) is performed by the Deinterleaver. DAB supports a wide range of different code rates between 8/9 and 1/4.

6. For Viterbi Decoding (VIT), the Channel Decoder DSP is required. The constraint length is 7 and 8 bit for soft decision are used.

7. Finally, energy dispersal (EDI), audio decoding (MP2) and the extraction of additional information from the FIC are done in the main CPU. EDI is performed by the XOR of the current sequence with a pre-defined pseudo-random one.

To simplify the DAB receiver implementation we enhanced the additional layer called `expressmimo_emu` that has previously been presented by the missing DAB functions. The runtime results can be seen in Table 3.15. As the FEP memory is limited to 4x4 kSamples, the FEP context has to be saved and restored several times in the DDR2 on the control side. Fig. 3.13 illustrates the runtime distribution for one DAB frame of 96 ms. Most of the processing resources are due to the memcopy operations between the baseband engine and the DDR2. The overall processing time is about 8.22 ms and the longest single DSP call for a frame of 96 ms is a Deinterleaver task of 1.23 ms. So there are a lot of resources for IEEE 802.11p tasks available.

Task	DSP	Memcopy	total
SFF	0.1 ms (17.2%)	0.48 ms (82.8%)	0.58 ms
STI	0.33 ms (9.2%)	3.26 ms (90.8%)	3.59 ms
SFC	0.3 ms (21.6%)	1.09 ms (78.4%)	1.39 ms
TTI	1.06 ms (18.1%)	4.80 ms (81.9%)	5.86 ms
TFF	1.60 ms (13.0%)	10.7 ms (87.0%)	12.3 ms
FFT	2.54 ms (25.5%)	7.43 ms (74.5%)	9.97 ms
DEM	2.95 ms (14.3%)	17.6 ms (85.7%)	20.6 ms
FDI	4.37 ms (18.6%)	19.1 ms (81.4%)	23.5 ms
PNT	6.08 ms (56.8%)	4.65 ms (43.2%)	10.7 ms
VIT	1.52 ms (24.6%)	4.65 ms (75.4%)	6.17 ms

Table 3.15: Task Runtime for DSP Engines and Memcopy for one Second of Audio Data

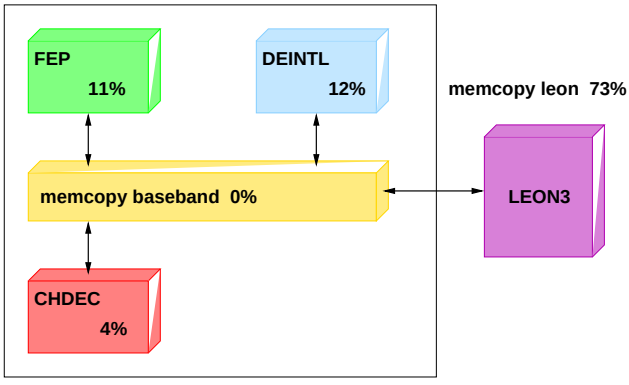


Figure 3.13: DAB Runtime Distribution for 1 DAB Frame (96 ms)

C) Runtime Distribution Comparison of ETSI DAB and IEEE 802.11p

The differences between the two standards are listed in Table 3.16. While the DAB receiver is operating on long vectors and has no latency requirements, IEEE 802.11p needs a very fast baseband engine to fulfill the strict latency requirements. The critical DSP in the design is the FEP. Although Deinterleaver and Channel Decoder also require quite some resources to process both

standards, usual tasks to be scheduled comprise only one operation. Instead, the FEP processing is split in a lot of different tasks that can be grouped to macro operations like channel estimation or data detection. The FEP DAB runtime distribution for a frame of 96 ms is given in Table 3.17. Most of the tasks are vector processing operations, that can be split easily at runtime if necessary. Critical are the DFT / IDFT operations requiring a processing time of 9.493 μs. A major challenge processing the two standards simultaneously is that the scheduler has to ensure, that these 27 operations can be scheduled without causing problems for the timing of the IEEE 802.11p receiver.

ETSI DAB	**IEEE 802.11p**
frame based standard	packet oriented standard
2048-OFDM	64-OFDM
OFDM symbol duration of 1 ms	OFDM symbol duration of 64 μs
DAB frame duration of 96 ms	variable packet length (between 48 μs and 10.968 ms)
deterministic timing and processing	processing time depends on packet size modulation scheme, interarrival time

Table 3.16: Comparison of ETSI DAB and IEEE 802.11p

Duration	**Number of Calls**
0.113 μs	443
1.813 μs	10
3.520 μs	15
6.933 μs	42
9.493 μs	27
13.760 μs	15

Table 3.17: FEP DAB Runtime Distribution for one Frame (96 ms)

Table 3.17 illustrates the FEP runtime distribution for the IEEE 802.11p receiver. The group size, denoted by N, strongly influences the number of tasks to be scheduled. Taking the worst case example of BPSK with a code rate of 1/2, the number of DATA OFDM symbols is 1366. For a group size of one, there are 2740 operations a 0.43 μs, but for a group size of eight (plus one group with a size of six), this can be reduced to 1376 operations. Comparing Table 3.16 with Table 3.17 it can be seen, that all entries in the IEEE 802.11p table are smaller than the second entry in the DAB table. To avoid a permanent scheduling of IEEE 802.11p tasks it is therefore recommended to group the FEP functions into execution groups. For the Deinterleaver and the Channel Decoder grouping of commands is only of interest when the DSP command can be combined with DMA transfers.

Duration	**const. part**	**init 16-QAM**	**init 64-QAM**	**BPSK/QPSK**	**16-QAM**	**64-QAM**
0.39 μs	2			N	N	N
0.43 μs	3	2	3	2	5	8
0.44 μs	1			N	N	N
1.39 μs	1					
1.4 μs	2					
1.43 μs	2					

Table 3.18: FEP Runtime Distribution IEEE 802.11p

D) Presentation of a Possible Scheduling on the ExpressMIMO Platform

The scheduler being developed for the ExpressMIMO platform has to be able to cope with different wireless communication standards that may have contrary latency requirements. Challenging in our case is the real-time processing of IEEE 802.11p despite of the simultaneous processing of DAB. Reconsidering the presented results, not much time is left for DAB FEP tasks once the beginning of a packet has been detected. Furthermore the whole scheduling is unpredictable as the arrival time of the next IEEE 802.11p packet is not known in advance.
The main goal of the scheduler is to keep the invoked DSPs busy most of the time to achieve the best cost-performance relation. Factors to be considered when designing the scheduler are (1) the data dependencies for both standards, (2) the processing time of the tasks to be scheduled and (3) the available memory space to avoid unnecessary memcopy operations. For (3) it is important to group IEEE 802.11p operations in so-called *Execution Groups* that comprise all the operations that have to be scheduled at once. To recall, possible groups can be the constant part of the packet and the different groups of the DATA field. In addition the DAB design has to make sure, that after each group, the DAB can use the resource it asked for without loosing any information. For the FEP, this is not a big problem. The MSS of the FEP is organized in four different banks of same size. While the IEEE 802.11p receiver uses all four banks, the DAB requires only three out of the four banks for its processing. So all known values, like the reference STS, as well as the circular buffer of the incoming samples of the IEEE 802.11p receiver can be stored in this unused memory bank.
Another challenge is that the scheduler has to be able to decide dynamically at runtime how to copy the data samples from one DSP to another. In case the next DSP is busy, the samples have to be stored into the VCI RAM to unblock the just executed DSP for the next task. If the next DSP is not busy, the samples can be copied directly (Fig. 3.14).

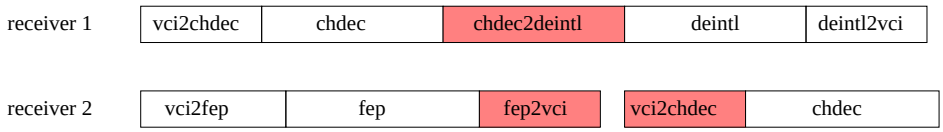


Figure 3.14: Flexible Memcopy Scheduling at Runtime

When scheduling DAB FEP tasks it is recommended to split them into smaller operations to guarantee the real-time behavior of the IEEE 802.11p receiver. In order to avoid unnecessary memcopy operations, the scheduler has to decide dynamically at runtime if a splitting of vector operations is necessary or not. The only operations that are not split are the DAB DFT / IDFT operations which take 9.493 μs. To ensure a high performance of both standards we therefore recommend to add a separate DFT / IDFT unit in the design.
For DAB, the long Deinterleaver tasks are only executed from time to time. In case IEEE 802.11p requires the same resource, its samples have to be buffered in the VCI RAM and executed at once as soon as the DAB Deinterleaver task is finished. To do so the FEP can already continue processing without being blocked till the IEEE 802.11p samples can be copied. The same is applied for the Channel Decoder. It is obvious that this only works if the main scheduler knows exactly at runtime how many data symbol groups are stored in the VCI RAM so that it can program the Deinterleaver and the Channel Decoder accordingly.
Within our collaborative work, a first scheduler has been implemented and tested in a software environment only. Nevertheless we enhance the following argumentation for the hardware pro-

cessing.
Both receivers are single-threaded. The parallelism of the different DSPs on hardware is realized within the execution groups by programming the different DSPs using a Round Robin scheduler. Each receiver task to be scheduled contains a flag stating if the task is `blocking` or `unblocking`. Blocking operations occur only within an execution group and state, that the scheduler is not allowed to give the DSP to another application. At the end of the execution group, the flag is set to unblocking to force the next action by the scheduler. The scheduling queues of the DSPs and the DMA engines have therefore a depth of only one command. We still have to include the deadlines of the macro operations. They have to be provided with the first command of an execution group and should represent the available processing time of the whole group and not of single tasks.
When applying a distributed control flow in the future, it is recommended to execute the content of the execution groups on the baseband side. In this case, the main CPU would only take care of the main scheduling of the execution groups and not of single tasks.

3.5.3 Runtime Performance Analysis - Hardware Results

The hardware results have been obtained via a cycle accurate HW/SW co-simulation using Modelsim. Besides the processing time that we already analyzed in the previous section, the presented results now include the communication overhead when a centralized control flow is applied and exploit a parallel processing of the different DSP engines on the ExpressMIMO platform.
The communication overhead can be observed when none of the DSPs is busy. By evaluating the relation between this factor and the processing time of the DSPs, clear statements about the receiver performance can be made.

3.5.3.1 Constant Part

Table 3.19 lists the DSP processing time and the communication overhead for the different algorithms implemented for the constant part. These are energy detection, packet synchronization, the calculation of the channel estimate and the SIGNAL field detection. The communication overhead of the first two entries contains the programming time of the operations as well as the value comparison to a known threshold in the main CPU or, more specifically, the LEON3 processor. The latter takes 350 ns once the result stored in the FEP MSS has been copied to LEON3.
The SIGNAL field detection requires three different DSP engines: FEP, Deinterleaver and Channel Decoder. The busy times including the DMA transfers are summarized in Table 3.19. The DMA transfer of the results between the DSPs are therefore considered twice (once per affected DSP) meaning that sum of the busy times does not correspond to the overall DSP processing time of the SIGNAL field. The additional communication time required for the FEP processing is related to the channel compensation algorithm where the result of the FEP sum operation has to be transformed into a complex 32 bit value in LEON3. During this time, the FEP should not be interrupted by another task. Furthermore, the calculation of the parameters needed for the DATA field detection (number of DATA OFDM symbols, code rate,...) requires an additional processing time of 2.74 μs.
To sum up, the overall processing time of the constant part is about 23 μs plus the time required for an internal DMA transfer in the FEP MSS due to some internal restrictions. This additional processing time may vary and causes an additional worst case processing overhead of 26 μs when using the internal FEP DMA engine but can be reduced to 6.6 μs when using the MOV operation of the FEP instead. Not considering this variable transfer time, only 40% of the Signal Field pro-

cessing time is required by the communication overhead. This value is still not optimum but can only be decreased by a distributed control flow on the platform.

	total proc. time	**DSP proc time**	**communication overhead**
Energy Detection	2.82 μs	1.51 μs (53.5%)	1.31 μs (46.5%)
Synchronization	8.01 μs	5.76 μs (71.9%)	2.25 μs (28.1%)
Calculation Channel Estimate	1.65 μs	0.82 μs (49.7%)	0.83 μs (50.3%)
Signal Field	11.64 μs	5.26 μs (45.2%)	6.38 μs (54.8%)

Table 3.19: Runtime Performance Results

	DSP proc time	**communication overhead**
FEP	2.99 μs (58.3%)	2.14 μs (41.7%)
DEINTL	1.62 μs	-
CHDEC	1.27 μs	-

Table 3.20: DSP Busy Times (Constant Part) including the DMA Transfers between the DSPs

3.5.3.2 Variable Part (DATA Field)

The FEP operations of each DATA field OFDM symbol comprise two main tasks: channel compensation and data detection. The average processing time as a function of the group size for the data detection is illustrated in Fig. 3.15. BPSK and QPSK are not listed, as the result of the channel compensation can directly serve as input for the Deinterleaver. For 64-QAM, the computation of the remaining bits results in a higher processing time of the FEP than for 16-QAM as two more bits have to be calculated. Furthermore it can be observed, that for an increasing group size, a boundary value is reached which is equal to the pure processing time of the DSP plus the communication overhead plus some delays due to the scheduler.
Fig. 3.16 illustrates the performance loss when applying a Round Robin scheduler. For that the average processing time of the Deinterleaver for 16-QAM with a code rate of 3/4 is given. The dotted curve represents the ideal case where tasks of the FEP and the Deinterleaver are scheduled instantaneously while the other curve shows the results when a Round Robin scheduler is applied.
Finally Fig. 3.17 and Fig. 3.18 illustrate the overall processing time including the DMA transfers of the DATA symbols for the FEP and the Deinterleaver. As expected, BPSK and QPSK perform best as only $R_{d,n}$ has to be copied from the FEP to the Deinterleaver. A centralized control flow is possible for BPSK, QPSK and 16-QAM as the processing time of the required DSP engines is below 8 μs which corresponds to the duration of one OFDM symbol.
More detailed results concerning the DSP processing time and the communication overhead for a group size of eight are given in Table 3.21 and Table 3.22. For the FEP, the DSP processing time takes between 40.1% and 56.03%. Best performs 64-QAM as the additional operations related to the data detection can be programmed while the FEP is busy. Thus only the DSP processing time increases while the additional communication overhead remains almost unchanged. For the Deinterleaver, the processing time takes between 52.41% and 86.76%. The highest value is achieved for 64-QAM with a code rate of 3/4 as the Deinterleaver operates on the largest possible vector in this design with a size of 8*432 samples.

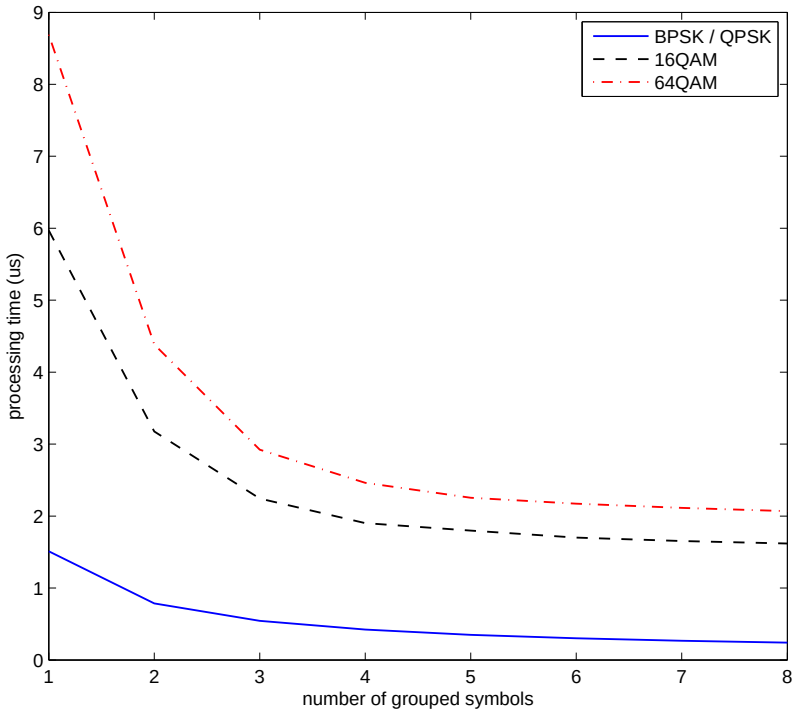


Figure 3.15: Average Processing Time Data Detection

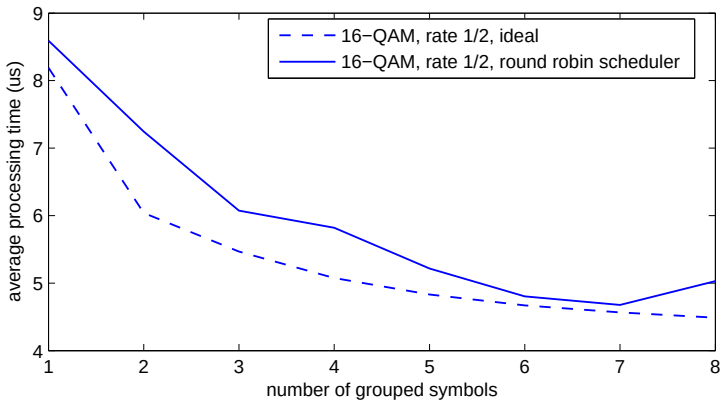


Figure 3.16: Round Robin Scheduler for 16-QAM

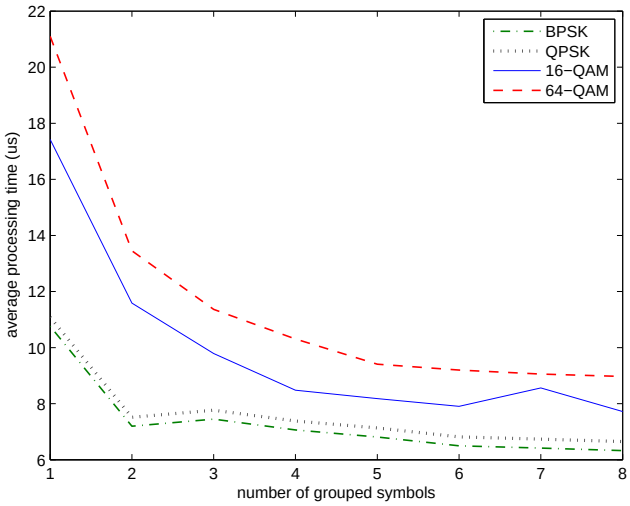


Figure 3.17: Average FEP Processing Time

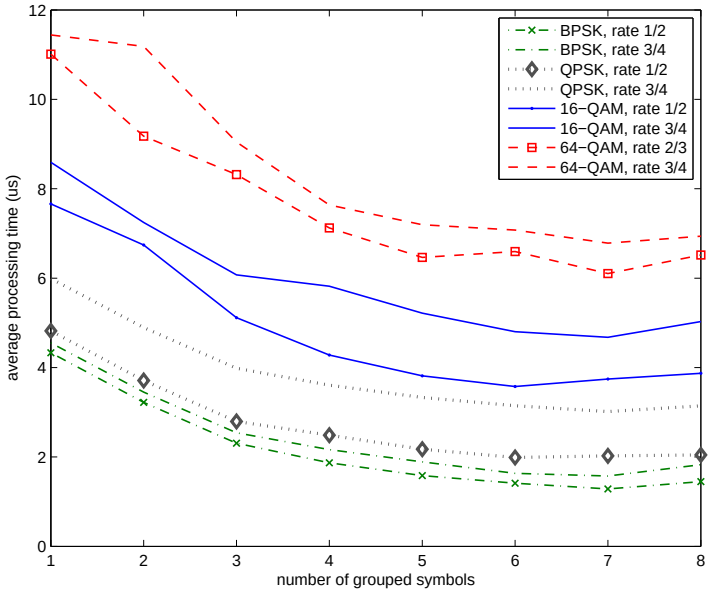


Figure 3.18: Average Deinterleaver Processing Time

	total proc. time	DSP proc time	communication overhead
BPSK	50.62 μs	20.5 μs (40.5%)	30.12 μs (59.5%)
QPSK	51.26 μs	21.14 μs (41.2%)	30.12 μs (58.8%)
16-QAM	61.74 μs	30.67 μs (49.7%)	31.07 μs (50.3%)
64-QAM	71.75 μs	40.2 μs (56.0%)	31.55 μs (44.0%)

Table 3.21: FEP Busy Times DATA Field (group size of eight)

	total proc. time	DSP proc time	communication overhead
BPSK (1/2)	11.6 μs	6.08 μs (52.4%)	5.52 μs (47.6%)
BPSK (3/4)	14.61 μs	8.48 μs (58.0%)	6.13 μs (42.0%)
QPSK (1/2)	16.36 μs	10.88 μs (66.5%)	5.48 μs (33.5%)
QPSK (3/4)	25.14 μs	15.68 μs (62.4%)	9.46 μs (37.6%)
16-QAM (1/2)	30.96 μs	22.64 μs (73.1%)	8.32 μs (26.9%)
16-QAM (3/4)	40.24 μs	32.24 μs (80.1%)	8 μs (19.9%)
64-QAM (2/3)	52.15 μs	43.36 μs (83.1%)	8.79 μs (16.9%)
64-QAM (3/4)	55.51 μs	48.16 μs (86.8%)	7.35 μs (13.2%)

Table 3.22: Deinterleaver Busy Times DATA Field (group size of eight)

3.6 Conclusions

In this chapter, we have presented a first receiver prototype for the ExpressMIMO platform. Chosen standard was IEEE 802.11p that is used for C2C and C2I communication. Its short data sets and strong latency requirements made these standard an ideal first use case to identify possible bottlenecks in the design. The different implementations for current version of the ExpressMIMO platform considering an FPGA target included a Matlab prototype, an emulation prototype using libembb and the prototype running on the real hardware.
Based on the obtained results we can state

- that the IEEE 802.11p receiver can be executed in real-time for BPSK, QPSK and 16-QAM. Assuming a higher target frequency like it is automatically the case when ASICs are considered, 64-QAM is real-time compliant as well.
- that a further reduction of the communication overhead can only be achieved by a distributed control flow using the UC or by a microprocessor or sequencer on the baseband side.
- that polling is the preferable solution to determine the end of the DSP processing when working with a standard with short data sets
- that commands have to be prepared for latency critical standards. For standards with long data sets like DAB or LTE, the real-time behavior is still guaranteed even if this recommendation is not considered. When processing a vector operation over a size of 4096 samples, for instance, the required processing time would be about 20 μs while the programming time of the DSP stays at a maximum of 360 ns.
- that further improvements of the FEP have to include a flexible type choice of the sum operation.

Apart from that this chapter presented a possible mapping of the DAB standard on the Express-MIMO platform. The results obtained using the libembb library have been compared to the IEEE 802.11p receiver results to get basic key figures that are important for the design of an appropriate scheduling algorithm. As critical DSP engine we have identified the FEP. Although it is not the computationally most intensive DSP, it has to execute most of the tasks including the latency critical ones. Therefore we strongly recommend either a faster and further improved design or the including of a second FEP in the baseband design. Based on the provided key figures we further derived guidelines for an appropriate scheduler and have presented a first prototype.

Chapter 4

ASIP Design for Front-End Processing

One of the key DSP engines on the ExpressMIMO platform is the Front-End Processor (FEP). To overcome performance limitations of the FPGA target when executing standards with short data sets as identified in the previous chapter, we replaced the vector processing unit of the manually designed programmable solution by a tool-based ASIP design. For development, the Language for Instruction-Set Architectures (LISA) that has gained commercial acceptance over the past years has been chosen.
To evaluate the proposed ASIP, it is compared to the programmable DSP solution as well as to two recent ASIPs from academia. The thorough comparison between them is carried out in terms of architectural differences and in terms of the runtime performance. For the latter, the processing time based on the actual cycle counts as well as the communication overhead on the ExpressMIMO platform is considered. In addition, we provide synthesis results for different target technologies.

4.1 Motivation

To recall Chapter 2.3.2.2, the FEP has been designed as a generic front-end for OFDM/A (Orthogonal Frequency Division Multiplexing / Multiple Access), SC-FDMA (Single Carrier FDMA), W-CDMA (Wideband Code Division Multiple Access) and SDMA (Space Division Multiple Access) air-interface operations. For the evaluation of the IEEE 802.11p standard, we considered a programmable DSP solution which is denoted as *C-FEP* in the following. This DSP is composed of a vector processing unit combined with a DFT / IDFT unit. In the previous chapter, we identified the need for a second FEP block or for an additional DFT / IDFT unit to increase the performance especially when processing standards with short data sets. Main drawback of the design was the huge communication overhead that leaded to a significant performance drop. It is worth to note, that these limitations are related to the FPGA target and not to the final ASIC one. For the redesign of the FEP we took the chance to collaborate with RWTH Aachen University (Germany) to evaluate the ASIP design methodology for ExpressMIMO platform design. Another aim of our collaboration was to overcome the C-FEP drawbacks for vector operation processing by removing the DFT / IDFT unit from the standardized DSP shell and by replacing the vector processing unit by an ASIP solution called *A-FEP*. Following this approach, the A-FEP can easily be embedded in the baseband processing engine of the ExpressMIMO platform and FEP tasks can be split and scheduled on the two FEP solutions simultaneously. For design evaluation, the A-FEP is not only compared to the C-FEP but also to other ASIP solutions from academia in terms of architectural differences and processing time.

But where is the main advantage of ASIPs when compared to other technologies? Important factors to be considered for SDR platform design are area and power consumption as well as the production costs. Major goal is to decrease the area and to minimize the power as much as possible by maintaining the performance. In [75], a detailed overview of the different System on Chip (SoC) implementation techniques is provided. Technologies of interest are

- **General Purpose Processors** that can be divided into two categories, GPP proper for general purpose applications and microcontrollers for industrial applications.
- **Digital Signal Processor** which are a subcategory of Application Specific Processor (ASP). DSPs are programmable microprocessors used for extensive numerical real-time applications that are specialized for the digital signal processing domain.
- **Application Specific Integrated Circuits** which are also a subcategory of ASPs. They are implemented in hardware, usually with a Hardware Description Language (HDL) like VHDL or Verilog.
- **Application Specific Instruction-set Processors** which are a subcategory of ASPs as well. They can be seen as a class of microprocessors coming with a specialized Instruction-Set Architecture (ISA).

The authors conclude, that ASIPs, tend to be suitable candidates as they are meant to fill the gap between GPPs and ASICs. Being tailored to a specific application, ASIPs offer a higher flexibility than ASICs by exhibiting a lower energy consumption than GPPs or DSPs at the same time. Or in other words, ASIPs allow to tradeoff the performance of ASICs against the flexibility of GPPs.
By additionally taking the advantage of high level tools, the prototyping is facilitated whereas the generated design is not hardware optimized and may not fit the dedicated resource (e.g. FPGA). On the other side, VHDL allows a resource-efficient FPGA design although the implementation requires a lot of time and resources. This drawback is overcome by tools like System Generator from Synopsis which speed up the VHDL design process by a high-level block design and by the support of fast design modifications.

4.1.1 Related Work

During the past years, lots of different solutions for flexible front-end processing as well as different ASIP design approaches and architectures have been proposed. The work presented in this section does not demand to be complete but shall give an overview of these different strategies. Usually, ASIP architectures are evaluated in terms of frequency, area, power consumption and the number of Millions of Operations or Instructions Per Second (MOPS / MIPS). Unfortunately, the latter does not provide any information about the processing time of the different air-interface operations like channel estimation or data detection. Therefore we opted for the processing time based on the cycle counts. Two recent ASIP solutions providing this information are the ASIPs developed by ETH Zürich [76] and by the Cairo University [77]. As these two solutions are mainly considered for comparison in the remainder of this chapter, a detailed architectural description is provided at the end of this section.

4.1.1.1 Front-End Processing Solutions

For the design of flexible front-end processing, [23] and [78] have endorsed the efficiency of a vector processing unit combined with SIMD operations. Although a higher performance can be obtained by exploiting instruction level parallelism using VLIWs or by task level parallelism.
In Chapter 2.2 an overview of commercial and academic SDR platform solutions has been provided, where flexible front-end processing was supported by SIMD and VLIW designs. Examples that have already mentioned comprise the EVP provided by STEricsson [79] which is a key DSP engine for 3G+/4G applications on low-power terminal architectures, the Systemonic HiperSonic1 [24], the Freescale MSC8156 high-performance DSP [25] and the Sandbridge SB3011 Platform [26].

Further solutions to be mentioned are

- **C66x Baseband DSP:** The C66x Baseband DSP is provided by Texas Instruments [80] and supports 2-way SIMD operations for 16 bit data and 4-way SIMD operations for 8 bit data. The instructions are based on 128 bit vectors.

- **SODA (Signal-Processor On-Demand Architecture):** SODA [81] is a fully programmable SDR architecture that consists of multiple processing entities (PE), a scalar control processor and a global scratch-pad memory that are all connected via a shared bus. The design of each PE includes an SIMD vector processing pipeline, a DMA engine, a scalar and an AGU pipeline as well as local memories. The architecture is further composed of an ARM Cortex-M3 processor being responsible for top level tasks, as well as a system bus that connects a global memory and four different processing elements where the latter support data-level parallelism. Achieved performance is 2 Mbit/s for W-CDMA and 24 Mbit/s for IEEE 802.11a (including Viterbi decoding).

- **ADRES:** ADRES [82] is a coarse-grained reconfigurable processor that supports instruction-level parallelism by tight coupled VLIW instructions. Its template consists of a set of different basic components whose types and interconnections are specified at design time.

- **SAMIRA:** SAMIRA [83] is based on the STA approach that has already been introduced in Chapter 2.2. It is a low-power high-performance floating-point vector DSP that supports instruction level parallelism in a VLIW fashion. Per cycle, each vector processing unit is able to compute eight single precision floating point operations. The whole design runs at a frequency of 212 MHz.

- **RaPID:** The datapath of the RaPID design consists of a variable number of linear array functional units (FUs) that can be Arithmetic Logic Units (ALUs), multipliers, registers or storage units. They are connected via a reconfigurable network. Purpose and quantity of the FUs are determined at design time. In [84] a 4 antenna OFDM receiver based on RaPID has been presented. Algorithms of interest in this paper are timing synchronization and the FFT of the four receive streams. These algorithms have also been mapped on ASIC, FPGA and DSP targets for design comparison. Main conclusion of the paper is that RaPIDs fill in the gap between ASICs and DSPs when considering the performance-costs relation.

- **Tensilica ConnX Baseband Engine:** Tensilica [85] is one of the big industrial provider for front-end DSPs. Its ConnX Baseband Engine [86] targets a high throughput for OFDM and MIMO applications and includes among others complex convolutions for synchronization,

FIR filters, FFT, complex vector multiplication, vector matrix operations, minimum search, support for searching and sorting functions, division operations, and so on. Basis of the design is the Tensilica Xtensa Processor featuring SIMD instructions. Per instruction, sixteen 18 bit multiplications, eight 20 bit additions or four 40 bit additions may be executed in parallel. Additionally, the baseband engine supports the execution of three instructions simultaneously (3-way VLIW architecture). Running at 400 MHz, the vector processing performance is similar to the one of the EVP but ConnX additionally comes with flexible SIMD processing, configurable hardware blocks and an extensible instruction-set.

- **Coresonic Solutions:** Another big industrial provider for front-end DSP solutions is Coresonic. In the following, some products are presented:

  - **Coresonic's LeoCore:** The LeoCore ([87]) is specifically designed for baseband processing applications. Its design is based on SIMT (Single Instruction stream Multiple Tasks) where parallel tasks are controlled by a single instruction flow. SIMT architectures obtain the same performance as SIMD / VLIW ones but with a smaller program size and a simplified control hardware.
  - **Coresonic's BBP1 processor:** The BBP1 processor ([88]) is a multi-standard baseband processor mainly designed for WLAN standards. Its baseband processing core comprises an ALU and a complex-valued MAC unit as well as data memories and specialized data processing blocks that can be programmed at runtime.
  - **Coresonic's BBP2 processor:** The BBP2 processor ([89]) is an improvement of the BBP1 processor by enhancing the first with the ability for multistandard processing. Its design includes two 4-way SIMD units operating on 16 bit complex vectors. Up to three different contexts for three different tasks are supported.

4.1.1.2 ASIP Design Approaches for Front-End Processing Solutions

One common approach when designing ASIPs is the usage of different Architecture Description Languages (ADL). In general, these languages can be grouped in three different categories ([90]):

1. **Instruction-set centric languages** mainly focus on the instruction-set of the processor. As stated in [90], they represent the programmer's view of the architecture and are mainly used to describe the instruction encoding, assembly syntax and behavior. Examples are nML [91] or ISDL [92].

2. **Architecture centric languages** mainly focus on the structural aspects for the architecture. In contrast to instruction-set centric languages, they represent the designer's view. Therefore, functional building blocks as well as interconnections are used to describe the hierarchy of the processor architecture. One example of such a language is MIMOLA [93].

3. **Mixed instruction-set and architecture oriented languages** are the combination of the two previous listed language types. Here, instruction-set description and structural aspects are combined which leads to ADLs being able to target all possible ASIP domains including architecture exploration and implementation as well as system integration and software development tool generation. Examples are EXPRESSION [94] and LISA [95], [96].

During the past years, lots of different ASIP solutions for front-end processing have been proposed, that were designed using the presented ADLs. Some of the architectures focus only on some air-interface algorithms while other designs are tailored to the processing of a specific group

of standards. Examples for the first group are [97] where channel equalization based on the TTA approach is presented, [98] which focuses on an ASIP for channel estimation, [99] which presents an ASIP for signal detection and coarse time synchronization using LISA or [100] which focuses on a flexible ASIP design for wireless communication standards. As a case study [100] presents the implementation of a HSDPA / WLAN equalizer and shows that the proposed architecture is capable to process the IEEE 802.11a standard in real-time.
One example where the processing is tailored to a specific group of standards is [101] where the proposed ASIP supports the execution of the IEEE 802.15.4a standard only.
In contrast to these ASIP solutions, the A-FEP being designed for the ExpressMIMO platform shall support

1. a wide range of different wireless communication standards and
2. a wide range of different air-interface operations

in a multimodal fashion including MIMO reception and transmission.

Most of the presented solutions are based on RISC (Reduced Instruction-Set Computer) architectures. In contrast to CISC architectures which combine arithmetic, logic and memory access instructions in only one instruction with a variable length, the length of the RISC instructions is fixed at design time [102]. This results in a simplification of the overall design process, especially the automatic code generation. Usually, RISC processors are based on a load-store policy using a set of general purpose registers. Programming is realized by the processor's instruction-set that is stored in the Program Memory (PM).

The presented A-FEP prototype is based on a RISC architecture and has been designed using LISA. Compared to ADLs like nML, ISDL or EXPRESSION, LISA comes with various advantages. It provides not only cycle-accurate processor models but also supports VLIW / SIMD / MIMD (Multiple Instruction, Multiple Data) instructions, is strongly C/C++ oriented so that the language is easier to learn, supports instruction aliasing as well as complex instruction coding schemes and allows to determine the abstraction level of the processor model.

4.1.1.3 ASIP Solutions for Design Comparison

In this section, the three solutions chosen for design comparison in Chapter 4.9 are presented. The first ones are the two ASIP solutions developed by ETH Zürich [76] which we denote as *ASPE A* in the following. Their architectures are based on the Adaptive Stream Processing Engine (ASPE) [103] which is a coarse-grained ASIP architecture being optimized for data processing. Main advantage when compared to other solutions are the lower costs based on the shortened design time and the limited runtime reconfigurability for bug fixes. Besides ASPE designs come with a better performance when compared to VLIW architectures due to the reduced load/store overhead. Each ASPE design is connected to a GPP taking care of the control and of performance uncritical tasks and includes three different building blocks whose quantity and type can be selected from a library at design time, depending on the target application.

1. **Functional Units** (FU) contain the arithmetic operations and can be combined to implement more complex ones like CORDIC. The number of internal pipeline stages is flexible and can be chosen at design time.
2. **Storage Units** (SU) are used for local data storage. They are connected to the FUs via a runtime configurable network.

3. **Sequencer Units** (SEQ) control the configurable network between FUs and SUs. They further support control related tasks like zero-overhead loops or a data dependent control flow.

The clock frequency is limited by the complex control network. In theory, each ASPE design should be able to schedule concurrent accesses of SEQs to the FUs. Therefore, the control network becomes a major design bottleneck in case only one SEQ is used.
In [76], first a Single Input Single Output (SISO) receiver tailored to the processing of the IEEE 802.11a standard is presented. Its ASPE A configuration is detailed in Table 4.1.

Ressource	Quantity	Comments
SEQ	1	Program Memory (512 words a 192 bit) - storage of the program control flow - storage of the 16 bit command words
FU	1	complex-valued multiply and accumulate unit
	2	complex-valued arithmetic logic units
SU	1	registerfile (16 registers)
	1	input data buffer (64x32 bit)
	6	data storage (256x32 bit)

Table 4.1: ASPE A Configurations for the IEEE 802.11a/n Receiver

In contrast, the presented 2x2 MIMO receiver is tailored to the IEEE 802.11n standard and enhances the described configuration by a second ASPE ASIP denoted as *ASPE B*. Tasks to be performed on ASPE B are MMSE (Minimum Mean Square Error) estimation and MIMO detection to achieve a higher performance of the overall design. These tasks are neglected for the comparison to the A-FEP.

The last ASIP solution chosen for comparison has been developed by the Cairo University and was presented in [77]. As this design only covers synchronization and acquisition of different OFDM standards like HIPERLAN/2, IEEE 802.11a or LTE, it is denoted as *Sync-ASIP* in the remainder of this chapter. The Sync-ASIP includes six 12 bit real adders, three 13 bit real multipliers, two 12 bit rounders, two 24 bit accumulators, ten 13 bit multiplexers and two 24 bit shifters that are distributed over three different pipeline stages. Like ASPE A, it supports more complex vector processing algorithms like CORDIC, maximum likelihood or correlation functions whereas the maximum vector length is set to 256. This corresponds to the maximum correlation length required for IEEE 802.16e and LTE. The MSS is accessed via a simple AGU and is build of 286 word dual-port banks a 24 bit. The instruction-set of the Sync-ASIP is composed of program flow instructions (conditional / unconditional jumps, move, ...), optimized instructions to facilitate the implementation of the synchronization tasks and vector instructions.

The synthesis results for the presented solutions are the following: For a 0.13 μm CMOS target process, the ASPE A SISO receiver configuration obtains a frequency of 160 MHz and requires a silicon-area of 1.9 mm^2. Instead the ASPE A MIMO receiver has been synthesized for a 0.18 μm CMOS target process. For a target frequency of 160 MHz the silicon area is 7.6 mm^2, although the ASIP can be executed up to 250 MHz.
Synthesizing the Sync-ASIP for the same target, the obtained frequency is 120 MHz and the area is 1.1 mm^2.

4.1.2 Contributions

All results presented in this chapter have been obtained in collaboration with the RWTH Aachen University (Germany). For the LISA design we used the Processor Designer from Synopsis (former Coware).
Throughout the collaboration, two different versions of the ASIP were developed:

1. The first version of the A-FEP, called A-FEP-V1, has been designed together with a colleague and was based on the FEP specification he derived during his PhD thesis. We will denote this first C-FEP design as C-FEP-V1. Soon after the results of our work were presented in his thesis report [37], the specification of the FEP has been reworked for design and performance improvements and some of the additional A-FEP-V1 features have been included in the current design of the C-FEP. For this reason and also to overcome the drawbacks of the first design (mainly the low frequency), we opted for a second ASIP version - although the first version was already very flexible. In this chapter we provide a short introduction and overview of the main results of this first A-FEP version and mainly focus on the second contribution.

2. The second contribution is a new ASIP design based on the new FEP specification. In contrast to the first version, the A-FEP also includes general purpose instructions and does not only replace the FEP but also extends it by UC operations. The UC is kept in the standardized DSP shell for the handling of DMA transfers but not for algorithmic processing. Compared to the first version of the A-FEP the second version comes with an enlarged instruction-set and obtains a higher frequency.

4.2 ASIP Design Methodology

The traditional process of embedded processors design can be split over four different phases [95]:

1. **Architecture Exploration Phase**
   In this phase, the micro-architecture is fixed and the instruction-set is defined based on a detailed HW/SW partitioning. There, the target application has to be analyzed to determine critical operations that may require a dedicated hardware support through specialized instructions. Furthermore the designer has to identify which instructions are required for processing and how the application functionalities have to be mapped onto the chosen processor architecture by achieving a maximum performance. The steps of this phase are performed in an iterative way and repeated till the best solution is found. This is very time consuming as every change in the architecture results in a manual redesign of the processor.

2. **Architecture Implementation Phase**
   In this phase, the previously designed processor is transformed in an HDL model using languages such as VHDL or Verilog. The outcome of this phase serves as input for synthesis.

3. **Software Application Design Phase**
   In this phase different software development tools necessary to program the processor are designed. These tools include C-Compiler / assembler / linker / debugger / simulation tools. In contrast to the hardware design where accuracy, resource or power consumption are important characteristics, main goal of the software design is to obtain fast simulation models. This results in a reimplementation of the tool-suite. The process is time consuming and comes with a high probability of consistency problems.

4. **System Integration and Verification Phase**
   In this phase, different co-simulation interfaces are developed so that the software simulator can be integrated in a system simulation environment. Modifications are time consuming as the interfaces have to be changed manually in case of architectural changes.

The two hardware and two software phases are usually executed in parallel by different groups of designers. This results in a potential inconsistency of the designs and thus in very long development periods.
The major advantage of the LISA processor design platform is that HDL code and software development tool generation are based on the same LISA model which reduces problems due to inconsistency significantly. In general, the software and hardware development processes are iterative and are executed in parallel. Based on the functional specification a first LISA description model is implemented. Through compilation either a software model or a synthesizable hardware model can be generated. While the first can further be validated and evaluated with the help of software development tools, the latter can be simulated by tools like Modelsim or can directly be synthesized. Based on the results of these two design flows, the functional specification and the LISA model are reworked. It is worth to mention, that the hardware and the software flow are analyzed in parallel so that design optimizations and bug fixes on the LISA description level directly influence both design flows.

4.3 Front-End Processing Algorithms

The FEP has been designed to deal with the different air-interface operations at the transceiver side including OFDM/A, SC-FDMA, W-CDMA and SDMA. The resulting set of operations to be executed at the transceiver side has been identified in [37] and comprises channel estimation, synchronization, carrier / coarse frequency offset estimation and data detection. In [23] it has been shown that these operations can be build from component-wise vector operations and a DFT / IDFT unit:

- component-wise vector addition: $Z[i] = X[i] + Y[i]$
- component-wise division: $Z[i] = X[i]/Y[i]$
- component-wise vector product: $Z[i] = X[i] \times Y[i]$
- dot product: $X.Y = \sum_{i=0}^{N-1} X[i] \times Y[i]$
- energy calculation: $E(X) = \sum_{i=0}^{N-1} |X[i]|^2$
- max/min, argmax / argmin operations
- DFT / IDFT

To increase the programmability and thus the flexibility of the design, both A-FEP versions do not contain the dot product and the energy calculation operations like stated above. Instead their computation is based on basic vector operations. An energy calculation can simply be computed by a vector square modulus and a vector sum. Similarly, the dot product of two vectors $X[i]$ and $Y[i]$ can be computed by a component wise vector multiplication and a vector sum.

4.4 First Version of the A-FEP (A-FEP-V1)

The first version of the A-FEP (A-FEP-V1) was implemented together with a colleague by the end of his thesis. As a detailed description of our work can already be found in [37] we will just provide an overview of the architecture and the main results in this section.

4.4.1 Functional Specification

Like it is the case for the current version of the A-FEP, the A-FEP-V1 is embedded in the former version of the standardized DSP shell whose architecture is slightly simplified and does not support any command preparation. The A-FEP-V1 replaces the vector processing unit of the former C-FEP version (C-FEP-V1) which implements directly the basic functions resulting from the air-interface analysis. These functions are component-wise addition, component-wise division, component-wise product, dot product, energy calculation and maximum / minimum, argmax / argmin calculations. To obtain the desired throughput of two vector elements per cycle, two or four complex input vector elements a 32 bit are read in and one or two result vector elements a 32 bit are written back. All real and imaginary parts are represented in Q1.15 format while the vector length differs between 1 and 8192 complex vector elements.
The C-FEP-V1 vector operations can either be performed on the whole vector or on a sub-band level. For the latter, a large vector is split into different sub-vectors of same size. Inside each sub-band, the parameters `skip` and `offset` may enable a skipping of addresses (Fig. 4.1). `Offset` is the distance between the start of the sub-band and the first vector element. Starting from this vector element, `skip` defines the distance between two consecutive vector elements till the end of the sub-band is reached.

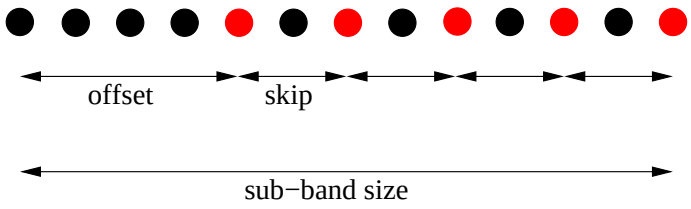


Figure 4.1: Definition of Skip and Offset within one Sub-band

The size of the input output data space of the MSS is 128 kB, the access delay is two cycles. Additionally an LUT can be stored at any memory location. This table contains the possible division factors $\frac{1}{x}$, where x can be any value in a Q1.15 format. Without optimizations, the required LUT would have a size of 64 kbit. To decrease this memory, we simplified the table by

1. storing only positive values. In case a negative value is required it is negated in the A-FEP-V1 pipeline.

2. ignoring most of the repeated values (e.g. between address 2^{10} and $2^{11} - 1$ only each 64th value is stored).

The resulting table has a size of 10.2 kbit.

4.4.1.1 ASIP Enhancements

In addition to the C-FEP-V1, the A-FEP-V1 has been enhanced by some features that have become part of the current version of the C-FEP. Besides the support of different data types which are 8, 16 or 32 bit integer values and pre- and post-processing value modifications including absolute value calculation, zeroing, negation and rescaling, another major enhancement of the A-FEP-V1 is the very flexible AGU. Thanks to programmable addressing schemes, vector elements can be read (written) from (to) non-contiguous addresses in the MSS. Furthermore address skipping, address repetition and periodic addresses as well as address wrapping to implement circular buffers inside the MSS are supported.

4.4.2 Architecture

4.4.2.1 Instruction Set

The instruction-set of the A-FEP-V1 consists of three different types of instructions, each having a size of 32 bit.

1. **Control Instructions:** The control instructions are ***NOP*** (no operation), ***JMP*** (jump to a given address in the program memory) and ***IRQ*** (interrupt request).

2. **Configuration Instructions:** The configuration instructions program the AGU and are only set in case of parameter changes.
   - ***agu_cfg_vector*** (basic AGU processing parameters like the vector size are set)
   - ***agu_set_vec0_addr*** (start address first input vector)
   - ***agu_set_vec1_addr*** (start address second input vector)
   - ***agu_set_res_addr*** (start address result vector)
   - ***agu_cfg_sub_vec_a*** (number of sub-vectors and their size)
   - ***agu_cfg_sub_vec_b*** (offset, skip)
   - ***agu_set_lut_addr*** (start address LUT)

3. **Arithmetic Instructions:** Arithmetic instructions are executed over multiple cycles whose actual number depends on the vector length. While executing these instructions, the Program Counter keeps its value and does not increment till the last result vector elements are in the pipeline.
   - ***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 a complex or a real vector)
   - ***vec_shift*** (vector shift)
   - ***vec_square*** (vector square)
   - ***vec_max_min*** (complex vector maximum / minimum in combination with argmax / argmin)

4.4.2.2 Pipeline Structure

The pipeline is split over six different stages.
In **Pre-Fetch (PFE)**, the Program Counter is incremented by one each cycle to fetch the next instruction from the Program Memory. The only exception are arithmetic instructions that are executed over multiple cycles. There the Program Counter is frozen.
Due to the two cycles delay when reading from the MSS, the **Fetch (FE)** stage is kept empty.
In the **Decode (DC)** stage, the instruction is loaded from the Program Memory. Depending on the instruction to be performed, the top level process is activated. In case of arithmetic operations, the signals to read the first input vector elements from the MSS are set.
Execute 1 (EX1) and **Execute 2 (EX2)** contain the ALU operations required for the arithmetic vector operation processing. In addition, EX1 contains several AGU functions as well as the LUT access. Per cycle, two or four input vector elements are read in EX1 to achieve a throughput of two vector elements per cycle. To reuse the existing resources, the vector elements are reallocated to the multiplier ports in case of `vec_abs_square`, `vec_sum`, `vec_max_min`, `vec_square` and `vec_div_s`. To give an example: The multiplier consists of eight 32 bit multipliers. In default case, the first one multiplies the real part of the first vector with the real part of the second one. In case of `vec_square` it has to multiply the real part of the first vector with itself. So a reallocation of the signals before starting the multiplier is necessary. In case the multipliers are not needed, the vector elements are sign extended. In EX2, first the vector elements are inverted if necessary which is the case for a subtraction for instance. The ALU consists of two adders that compute the sum of three 32 bit values and two adders that compute the sum of two 32 bit values. Furthermore it contains shift left / right and max / min / argmax / argmin operations and a truncation unit that is activated in case none of the other ALU functions is executed.
Finally, the results are truncated to 32 bit vector elements and written back in the **Writeback (WB)** stage. The necessary addresses are generated in WB as well.
A simplified architecture view of the whole pipeline is provided in Fig. 4.2.

4.4.3 Design Comparison

Compared to the C-FEP-V1, the A-FEP-V1 exhibits a higher programmability and thus a higher flexibility at runtime. The throughput of both designs is the same for common operations except for dot product and energy calculation. There the throughput of the A-FEP-V1 is decreased by a factor 2 as two vector processing instructions are required for the calculation of the result. In terms of memory, the input / output data space of both versions is the same. Only difference is that the MSS of the A-FEP-V1 includes a Program Memory for program code storage while the C-FEP includes some additional memory space required by the DFT / IDFT unit.

Synthesizing the A-FEP-V1 for the baseband engine FPGA of the ExpressMIMO platform (Xilinx Virtex 5 LX330, speed grade -2) the design obtains a maximum frequency of 78.6 MHz after place and route and needs 7493 function generators, 1874 CLB slices, 1394 DFFs or latches and 8 DSP48E slices.
In the future, an ASIC target may also be considered. Table 4.2 lists the results for a 65nm target library with low power and high voltage threshold. It is characterized for a typical manufacturing process at 1.2 Volts power supply and 25°C temperature.

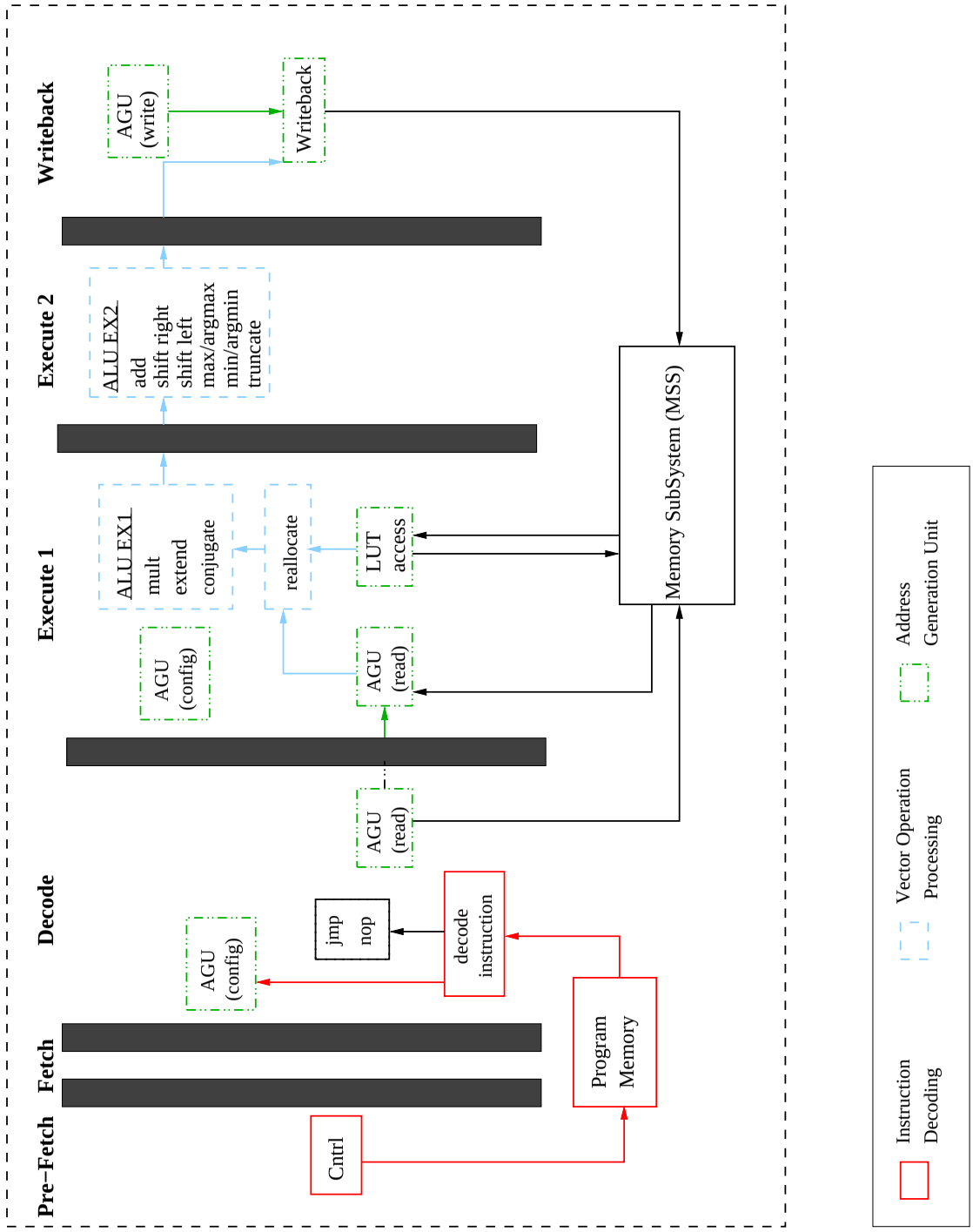


Figure 4.2: Pipeline Structure of the A-FEP-V1

Design	Target clock period (ps)	Target Freq. (MHz)	Silicon area (μm^2/FG)	Slack (ps)	Max Freq. (MHz)
	3300.00	303.03	107968	1585.00	204.71
C-FEP-V1	5400.00	185.19	97987	114.00	181.36
	5430.00	184.16	99497	0.00	184.16
	2000.00	500.00	91686	1095.00	323.10
A-FEP-V1	3095.00	323.10	93495	0.00	323.10
	3291.00	303.86	82166	0.00	303.86
	3350.00	298.51	82168	0.00	298.51

Table 4.2: Synthesis Results for the C-FEP-V1 and the A-FEP-V1

Based on these results we can state that the hardwired C-FEP-V1 performs better than the A-FEP-V1 in terms of area and frequency. The silicon area increases by 19% and the decrease in the maximum achievable frequency is almost 70%. In addition, the obtained frequency for the FPGA target is not high enough to process the A-FEP-V1 on the ExpressMIMO platform where a frequency of at least 100 MHz is required. The critical path can be found in EX1 (LUT access).

4.5 Functional Specification

To overcome the flexibility limitations of the first C-FEP design, there was a need for the design of a new version. The underlying functional specification is the same for both designs, C-FEP and A-FEP, except for the DFT / IDFT unit which is therefore neglected in this context.

4.5.1 Vector Processing

Like for the previous version, the throughput of the architectures is two samples per cycle. That is why the number of input vectors read in and output vectors written back to the MSS depends on the operation to be performed. In contrast, now four different data types are supported: vectors of 8 or 16 bit signed integers (`int8`, `int16`) and vectors of complex numbers where real and imaginary parts are 8 or 16 bit signed integers (`cpx16`, `cpx32`) as well as type conversions between them.

The C-FEP includes a set of basic vector operations listed in Table 4.3. The maximum vector length of all operations is 2^{14}.

Component-Wise Addition (CWA)	$Z[i] = X[i] + Y[i]$
Component-Wise Product (CWP)	$Z[i] = X[i] \times Y[i]$
Component-Wise Square of Modulus (CWM)	$Z[i] = \lvert X[i] \rvert^2$
Move (MOV)	$Z[i] = X[i]$
Component-Wise filter by a Lookup table (CWL)	$Z[i] = Y[X[i]]$

Table 4.3: C-FEP Vector Operations

Inputs can be modified on-the-fly before the actual computation (zeroing, absolute values, negate, conjugate, re-scaling, etc.) and outputs can also be modified after computation and before storage. Optionally, sum, max, min, argmax and argmin (the *SMA* values) can be computed on-the-fly for each of these operations and independently on real and imaginary parts. In the C-FEP, the CWL

vector operation consists in filtering an integer vector $X[i]$ through another one, $Y[i]$, which is used as an LUT. A programmable number of most significant bits of the components of $X[i]$ addresses the LUT. Optionally, the remaining least significant bits can be used to interpolate or extrapolate between two consecutive entries of the table. This operation approximates non-linear operations like the invert, log, square root, sine, cosine, etc. A component-wise division can be computed by first running CWL to store the inverse of the division factor in the MSS and a component-wise multiplication afterwards.

For the A-FEP design, the functionality of the C-FEP is split over different instructions (Table 4.4).

Component-Wise Addition (CWA)	$Z[i] = X[i] + Y[i]$
Component-Wise Product (CWP)	$Z[i] = X[i] \times Y[i]$
Component-Wise Square of Modulus (CWSM)	$Z[i] = \|X[i]\|^2$
Move (MOV)	$Z[i] = X[i]$
Component-Wise filter by a Lookup table (CWL)	$Z[i] = Y[X[i]]$, only division implemented
Component-Wise Square (CWS)	$Z[i] = X[i]^2$
Vector Sum (VECS)	$Z = \sum X[i]$
Vector Shift (VECSI)	$Z[i] = X[i] >> l$, $Z[i] = X[i] << l$
Vector max / min (VMM)	$Z = max(X[i])$, $Z = min(X[i])$

Table 4.4: A-FEP Vector Operations

It can be seen that the SMA value computation is represented by their own instructions (`vec_sum`, `vec_max_min`) and that additionally, a shift operation has been implemented. The complete CWL functionality is not included in the current A-FEP prototype yet. For the moment only a division can be performed to fulfill the requirements presented in Chapter 4.3. How the complete CWL functionality could be included in the A-FEP is illustrated in Chapter 4.8.

4.5.2 Memory Sub-System (MSS)

The MSS is build of a set of different memory blocks:

1. **UCM**, the 2 kBytes microcontroller program and data memory.
2. **MIO**, the 64 kBytes input-output data space
3. **TMP**, the 50 kBytes temporary DFT/IDFT memory
4. **TWD**, the DFT/IDFT 2 kBytes twiddle factors read-only memory
5. **PM**, the 4 kBytes A-FEP Program Memory

For the C-FEP, the MSS is made up of UCM, MIO, TMP and TWD. TMP and TWD are local private memories and are hidden to the host system. As the DFT / IDFT unit is currently not part of the A-FEP, TMP and TWD have been removed from its MSS. Additionally, a Program Memory, has been included to store the instructions of the program code. MIO and UCM are the same for both designs. From the external view, the MSS is seen as one coherent memory block that can be accessed by the processing unit, the DMA, the target AVCI interface and the UC. The highest priority is given to the processing unit, followed by UC, DMA and AVCI-Interface.
The number of cycles required till the instruction is available in the A-FEP pipeline or till the C-FEP has read a value is fixed to three cycles.

4.5.3 Address Generation Unit (AGU)

The addressing schemes are programmable and allow to build input vectors from non-contiguous data sets in the MSS. Symmetrically, the results can be stored at non-contiguous locations. In both directions, address skipping and (periodic) address repetition are supported. In addition, the MSS sections can be turned into circular buffers. Possible boundaries for these buffers are one half, one quarter or one eighths of an MSS bank. Fig. 4.3 illustrates the wrapping section distribution over one MSS bank for an input or output vector type of `int8`.

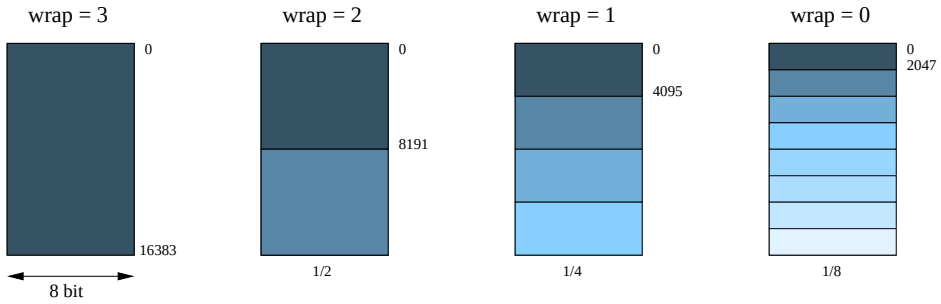


Figure 4.3: Wrapping Sections FEP MSS (`int8`)

The number of components, n_c per wrapping section can be expressed as

$$n_c = 2^{11+wrap}/k \tag{4.1}$$

where k depends on the vector type and represents the number of bytes per vector element (`int8`: $k = 1$, `int16`: $k = 2$, `cpx16`: $k = 2$, `cpx32`: $k = 3$). $wrap$ represents the chosen wrapping section as shown in Fig. 4.3.
The address of the first component inside this wrapping section, denoted as b_0, can be calculated if the addressed MSS bank is known.

$$b_0 = \text{bank\_index} - (\text{bank\_index} \bmod n_c) \tag{4.2}$$

Both, n_c and b_0 remain unchanged for each vector. Consequently, these values can be calculated as soon as the required parameters are available.

Each address depends on the memory bank and on an address offset. The latter is type dependent and has therefore to be multiplied with the number of bytes of the input / output vector elements (k).

$$\text{addr\_offset} = u_i * k \tag{4.3}$$

u_i is the bank indice inside the memory bank. For its calculation first an increment factor is needed which is based on two different parameters, n and m. n is the integer part of the increment, m is the fractional part. By choosing these two values appropriately, address skipping and address repetition can easily be implemented. The increment may vary between each of the two input vectors and the result vector.

$$inc = \begin{cases} n + 1/m & \text{if } m \neq 0 \\ n & \text{if } m = 0 \end{cases} \tag{4.4}$$

Finally, the resulting bank indice for each of the addresses can be expressed as

$$u_i = bx_0 + \left(\text{bank\_index} + (-1)^{\text{incr\_sign}} . \lfloor i \times inc \rfloor\right) \bmod n_c \tag{4.5}$$

The sign of the increment can either be positive or negative. In case of the latter, the addresses are decremented.
In read mode, addresses can also be periodic. For that a parameter is provided that represents the number of addresses to be generated within one period. Once this value is reached, the increment factor is reset to zero.
Table 4.5 presents possible addressing schemes for different parameter settings.

vector type	int8	int16	cpx32	cpx32	cxp32
bank_index	15	3	2	0	1011
incr_sign	1	0	0	0	0
n	1	1	0	1	6
m	0	2	2	5	0
address period	5	0	0	0	0
wrap	3	3	3	3	0
u_0/**addr_offset**$_0$	15/15	3/6	2/8	0/0	1011/4044
u_1/**addr_offset**$_1$	14/14	4/8	2/8	1/4	1017/4068
u_2/**addr_offset**$_2$	13/13	6/12	3/12	2/8	1023/4092
u_3/**addr_offset**$_3$	12/12	7/14	3/12	3/12	517/2068
u_4/**addr_offset**$_4$	11/11	9/18	4/16	4/16	523/2092
u_5/**addr_offset**$_5$	15/15	10/20	4/16	6/24	529/2116
u_6/**addr_offset**$_6$	14/14	12/24	5/20	7/28	535/2140

Table 4.5: AGU Adress Generation Examples

4.6 Architecture of the C-FEP

The C-FEP is a programmable DSP engine that has been developed by the System on Chip Laboratory (LabSoC) of Télécom ParisTech. Like for all other DSPs on the ExpressMIMO platform, the required processing parameters are included in the CSS of the standardized DSP shell. These parameters define the input / output data types, the vector size, etc.
The pipeline architecture is illustrated in Fig. 4.4. It can be seen that the processing core is split over two identical processing units (PU). To fulfill the requirements of the DFT / IDFT as well, each of them embeds twenty-four 25×18 bit signed multipliers and twelve 43 bit accumulators. For DFT processing, they implement two radix-4 butterflies but the same resources are used for the vector processing as well.

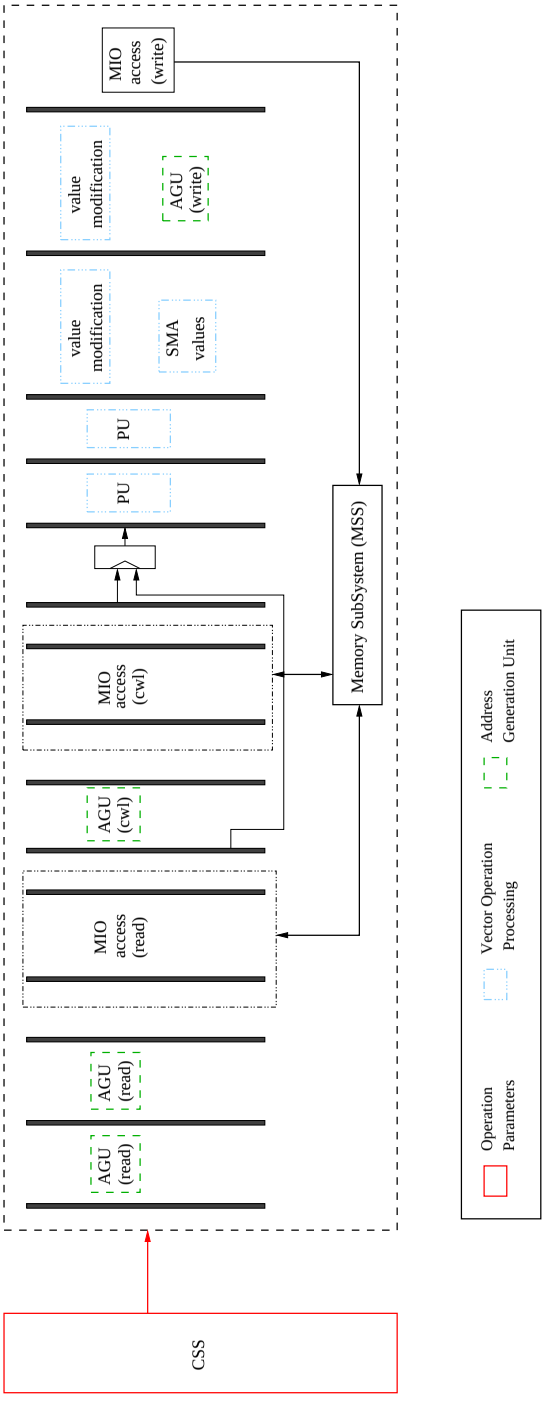


Figure 4.4: C-FEP Architecture

On the Xilinx Virtex5 FPGA target, the PUs are mapped on the hardwired DSP48E slices. Please note, that the pipeline has to be emptied before the next vector operation can be executed. When processing large vectors, the instruction fetches / data fetches ratio is close to zero. But on the other side this results in a significant performance drop especially when processing standards with short data sets which is worse when compared to a classical vector processor and even worse than with a DSP.

4.6.1 Synthesis Results

Synthesizing the C-FEP (PU and MSS) for the Xilinx Virtex 5 LX330 FPGA with a speed grade of -2, a maximum frequency of 96 MHz is achieved after place and route by requiring 20119 function generators, 5030 CLB slices, 10945 DFFs or latches, 33 block RAMs and 24 DSP48E slices. Although the maximum processing frequency of the ExpressMIMO platform is currently set to 100 MHz, this frequency value is acceptable as the mapping on the FPGA changes once all the DSPs are considered for synthesis. For the IEEE 802.11p receiver where we included Channel Decoder, Deinterleaver, FEP and VCI RAM, the obtained C-FEP frequency slightly increases so that the platform design can be processed at the mentioned target frequency.
For the ASIC target (65nm target library), only the processing engine of the C-FEP has been synthesized as the new MSS design is still not finalized. The maximum frequency achieved is about 450 MHz; the required area is 0.48mm^2.

4.7 A-FEP Design

The A-FEP design is based on the same functional specification as the C-FEP but has been optimized for an efficient processing of standards with short data sets. Like all other designs it is embedded in the standardized DSP shell so that the A-FEP can be added as a separate unit in the baseband engine if necessary. In this case, latency critical tasks can be scheduled to the A-FEP while DFT / IDFT and latency non-critical tasks are still in the responsibility of the C-FEP.

4.7.1 Instruction-Set and Opcode

The instruction-set of the ASIP comprises three different types of instructions: (1) configuration instructions needed by the AGU, (2) Arithmetic Vector Operation (AVO) instructions and (3) General Purpose (GP) instructions.

The number of the **configuration instructions** can vary and depends on the amount of AGU parameters to be updated. Per AVO instruction, between 3 and 6 AGU instructions are needed which results in an average overhead of maximum 6 cycles per instruction. For long vector operations this overhead is negligible.
Table 4.6 gives an overview of these instructions and the parameters associated with each of them. The parameters are further detailed in Table 4.7.
The second type of instructions are the nine different **AVO instructions** identified in context of the functional specification. Each instruction has the same set of parameters where the vector type or a downscaling factor can be set. Table 4.8 gives an overview of these instructions and the parameters associated with each of them. The parameters are further detailed in Table 4.9.

instruction	opcode	parameters
agu_cfg1	0010	vec_size || wrap_vec0 || wrap_vec1 || wrap_res || mss_bank_vec0 || mss_bank_vec1 || mss_bank_res
agu_cfg2	0011	type_vec0 || type_vec1 || type_res || incr_n_vec0 || incr_n_vec1 || incr_n_res || cwl_enable
agu_cfg3	0100	incr_m_vec0 || incr_m_vec1 || incr_m_res || l_cwl_vec1
agu_cfg_writeindex	0101	index_res || incr_sign_vec0 || incr_sign_vec1 || incr_sign_res
agu_readindex	0110	index_vec0 || index_vec1
agu_period	0111	period_vec0 || period_vec1

Table 4.6: Instruction Set and Opcode (AGU Configuration Instructions)

Parameter	Type	Description
vec_size	`uint15`	Vector length ($1 \leq l \leq 2^{14}$)
type_vec0, type_vec1, type_res	`uint2`	Type of components
mss_bank_vec0, mss_bank_vec1, mss_bank_res	`uint2`	MSS bank
index_vec0, index_vec1, index_res	`uint14`	Base bank index
incr_n_vec0, incr_n_vec1, incr_n_res	`uint7`	Integer part of index increment
incr_m_vec0, incr_m_vec1, incr_m_res	`uint8`	Fractional part of index increment ($m \neq 1$)
incr_sign_vec0, incr_sign_vec1, incr_sign_res	`uint1`	Sign of index increment
period_vec0, period_vec1	`uint14`	Period
wrap_vec0, wrap_vec1, wrap_res	`uint2`	Wrapping section

Table 4.7: Overview of the AGU Configuration Instruction Parameters

instruction	opcode	opcode	parameters
	1000	xxxx	mod_v0_real || mod_v1_real || mod_v0_img || mod_v1_img || maxmin || res_shift || shift_info || interpolate || scale_vec0 || scale_vec1 || scale_res || info_res
vec_mult		0000	
vec_add		0001	
vec_square_modulus		0010	
vec_square		0011	
vec_move		0100	
vec_sum		0101	
vec_shift		0110	
vec_max_min		0111	
vec_cwl		1000	

Table 4.8: Instruction Set and Opcode (AVO Instructions)

Parameter	Type	Description
mod_v0_real	`uint2`	modification real part vector 0
mod_v1_real	`uint2`	modification real part vector 1
mod_v0_img	`uint2`	modification imaginary part vector 0
mod_v1_img	`uint2`	modification imaginary part vector 0
maxmin	`uint1`	=1 if max processing
res_shift	`uint3`	number of right shift before the output value modification
shift_info	`uint5`	MSB = 1 if left shift, the remaining bits tell how many positions to shift
interpolate	`uint1`	=1 if interpolation (currently not used)
scale_vec0	`uint1`	=1 if downscaling by 256
scale_vec1	`uint1`	=1 if downscaling by 256
scale_res	`uint1`	=1 if downscaling by 256 / saturation
info_res	`uint1`	=1 use the real part (cpx->int)

Table 4.9: Overview of the AVO Instruction Parameters

For the A-FEP, the component-wise division ($Z[i] = Y[i]/X[i]$) is realized as a component-wise multiplication of $Y[i]$ and $1/X[i]$ by using the CWL instruction. This introduces an additional delay of three cycles as $1/X[i]$ has first to be read from an LUT that has been stored in the MSS before. To access the LUT, the 11 LSB of $X[i]$ are used. In contrast to the C-FEP, the component-wise division can be performed with only one instruction while it is split over two vector operations in case of the C-FEP.

The last type of instructions are the **GP instructions**, including NOP, IRQ and load / store instructions as well as conditional branch, compare and common ALU instructions. For these instructions, a Registerfile with a size of 16 registers a 32 bit has been added. *Immediate* values denote values included in the instruction word, while `src` and `dst` refer to registers in the Registerfile. Per cycle, one instruction is read from the Program Memory. Only exception is the `load` instruction, where a value has to be read from the MSS which introduces an additional delay of 3 cycles.

- **IRQ & NOP instructions:**
  For IRQ, an output pin of the A-FEP is set to one for one cycle. This pin can be interpreted by the CSS and is connected to the CSS IRQ pin to signal the main CPU that the processing of a scheduled task is finished. Table 4.10 gives an overview of these instructions.

instruction	opcode
nop	00000..0
irq	00010..0

Table 4.10: Instruction Set and Opcode (IRQ, NOP)

- **ALU instructions:**
  Each of the two main instructions, `alu_rr` and `alu_ri` (Table 4.11), can further be specified by a set of sub-instructions like xor, or, etc. The result is always written back to the Registerfile. ALU operations can either be performed between two register values or between a register value and an immediate value. In case the subsequent instruction requires the result of an ALU instruction, a NOP has to be included between the two as the processing is split over two different pipeline stages.

alu_rr (ALU operation between two register values)	1001	**opcode**	src1 || src2 || dst
alu_ri (ALU operation between a register value and an immediate)	1010	**opcode**	src1 || dst || imm16
opcode			
xor		0000	
or		0001	
and		0010	
sub		0011	
add		0100	
addc (add with carry)		0101	
asr (arithmetic shift right)		0110	
asl (arithmetic shift left)		0111	
addu (add unsigned)		1000	
adduc (add with carry unsigned)		1001	
subu (sub unsigned)		1010	

Table 4.11: Instruction Set and Opcode (GP - ALU)

- **load / store instructions:**
  Load / store instructions load a value from the MSS or store it in the MSS. For load instructions, `src` always denotes the MSS bank while the `imm16` value contains the 16 bit address offset inside this bank. `dst` is the destination register in the Registerfile. For store instructions, the `src` denotes the register in the Registerfile, while `dst` stands for the MSS bank in which the value has to be stored. Like for the load instructions, the `imm16` value contains the 16 bit address offset inside the MSS bank. Table 4.12 gives an overview of the load / store instructions and the parameters associated with each of them.

	1011	**opcode**	src || dst || imm16
ldc_ri (load sign extended immediate)		0000	
lui_ri (load zero extended immediate)		0001	
lhu (load half word unsigned)		0010	
lbu (load byte unsigned)		0011	
lb (load byte)		0100	
lh (load half word)		0101	
lw (load word)		0110	
sb (store byte)		0111	
sh (store half word)		1000	
sw (store word)		1001	

Table 4.12: Instruction Set and Opcode (GP - LOAD / STORE)

- **branch instructions:**
  Branch instructions influence the address of the Program Counter and apply on values stored in the Registerfile. A special instruction is `move` where `src2` denotes the new address of the Program Counter. Otherwise a branch to the address provided by the `imm16` value is performed depending on the outcome of the comparison of the values stored in the Registerfile at addresses `src1` and `src2`. Table 4.13 gives an overview of the branch instructions and the parameters associated with each of them.

	1100	**opcode**	src1 || src2 || imm16
mov (move)		000	
bge (branch if greater or equal than)		001	
ble (branch if less or equal than)		010	
bgt (branch if greater than)		011	
blt (branch if less than)		100	
bne (branch if not equal)		101	
beq (branch if equal)		110	
bal (branch and link)		111	

Table 4.13: Instruction Set and Opcode (GP - BRANCH)

- **compare instructions:**
  Comparisons can be performed either between two register values or between a register value and an immediate value. In case the subsequent instruction is based on the outcome of the comparison, a NOP has to be included as the processing is split over two different pipeline stages. Table 4.14 gives an overview of the compare instructions and the parameters associated with each of them.

cmp_ri (compare register value to immediate)	1101	**opcode**	src1 || src2 || dst
cmp_rr (compare two register values)	1110	**opcode**	src1 || dst || imm16
opcode			
geu (greater or equal - unsigned)		0000	
leu (less or equal - unsigned)		0001	
gtu (greater than - unsigned)		0010	
ltu (less than - unsigned)		0011	
ge (greater or equal)		0100	
le (less or equal)		0101	
gt (greater than)		0110	
lt (less than)		0111	
ne (not equal)		1000	
eq (equal)		1001	

Table 4.14: Instruction Set and Opcode (GP - COMPARE)

4.7.2 Pipeline

To achieve a high performance, the pipeline of the A-FEP consists of 11 stages (Fig. 4.5). Due to three cycles delay when accessing the MSS, the pipeline is stalled when performing an AVO instruction, a branch (in case the required program memory address is stored in the MSS) or a load instruction. For AVO instructions, the next instruction from the Program Memory is read as soon as the last vector elements are read from the MSS. In contrast to the C-FEP, the pipeline does not have to be emptied before the next instruction can be processed which reduces the internal latencies significantly.
The throughput of the A-FEP is two components per clock cycle for AVO instructions. Some extra clock cycles (14 or 17) are spent in initialization and termination. Independent of the instruction type, the writeback (either in the Registerfile or in the MSS) always happens in the last pipeline stage. In case a following GP instruction requires the previously computed GP result, bypasses are used to gain a higher performance of the processor.

- **Pre-Fetch (PFE):** The Program Counter indicating the address offset of the next instruction to be fetched from the Program Memory is incremented in PFE. As AVO instructions are multi-cycle ones, the Program Counter keeps its value during this time and is decremented by three (it takes three cycles to inform PFE that an AVO instruction is processed) before the next instruction is loaded.
- **Fetch (FE):** Due to the MSS delay till the fetched instruction is available, the FE stage is kept empty.
- **Decode (DC):** In DC, the top level processes are activated depending on the instruction that has been decoded.
- **Execute 0 (EX0):** This pipeline stage contains only AGU processes (read mode).
- **Execute 1 (EX1):** In this stage the samples read from the MSS are available and forwarded to the first ALU that scales the vectors down by 256 in case the related parameters are set.
- **Execute 2 (EX2):** EX2 contains the ALU responsible for the input value modification (force to zero, negate, absolute value calculation). This stage is only activated if an AVO instruction is processed.
- **Execute 3 (EX3):** EX3 contains 8 multipliers (17 x 17 bit) and a sign extension in case the multipliers are not required. This stage is only activated if an AVO instruction is processed.
- **Execute 4 (EX4):** EX4 contains inversion, shift operations and max/min operations. This stage is only activated if an AVO instruction is processed.
- **Execute 5 (EX5):** EX5 contains 2 adders that compute the sum of two 34 bit values, 2 adders that compute the sum of three 34 bit values and a sign extension in case the adders are not required. This stage is only activated if an AVO instruction is processed.
- **Execute 6 (EX6):** EX6 contains a truncation unit. How the results are truncated depends on the instruction that is processed.
- **Writeback (WB):** In WB, the write addresses are finally generated and the output values are modified depending on the parameters being part of the instruction word. The actual number of vector elements to be written back depends on the instruction to be processed.

4.7.3 Synthesis Results

Synthesizing the A-FEP together with its MSS for the Xilinx Virtex 5 LX330 FPGA (speed grade -2) a maximum frequency of 105 MHz has been obtained after place and route. Compared to the former version of the A-FEP, the frequency could thus be increased by almost 30 MHz but with the cost of additional resources. This is also due to the fact, that the A-FEP-V1 has never been synthesized together with its MSS. Resources required by the A-FEP for the FPGA target are 13122 function generators, 3281 CLB slices, 6433 DFFs or latches, 17 block RAMs and 8 DSP48E slices.
For the ASIC target where only the processing engine of the A-FEP has been synthesized, a maximum frequency of about 550 MHz with an area of 0.18 mm^2 is achieved.

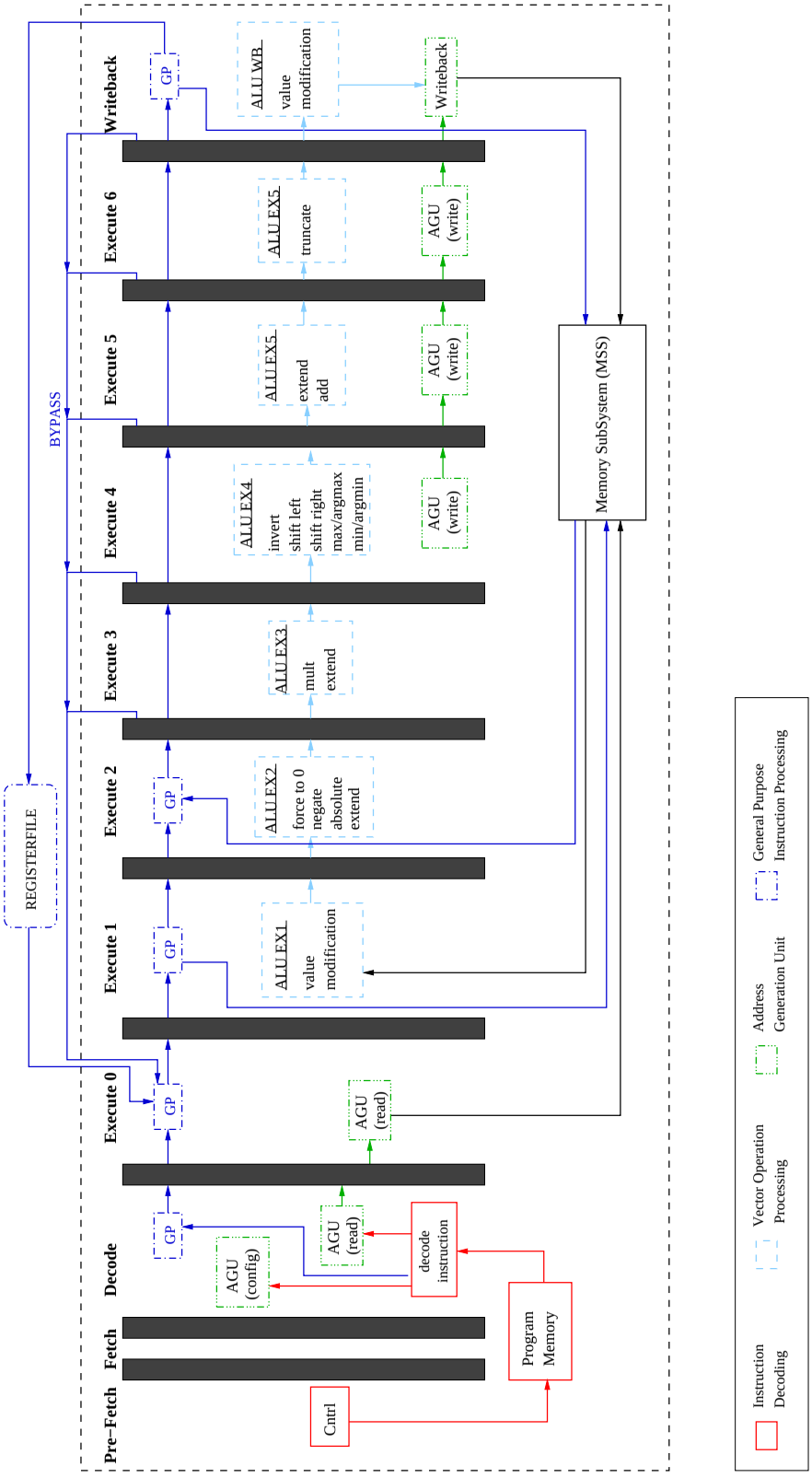


Figure 4.5: Pipeline Structur of the A-FEP

4.7.4 Cycle Counts

The cycle counts of the A-FEP design depend on the instruction that is processed. In case only one AVO instruction is processed and the A-FEP is stopped afterwards, the number of cycles are $\frac{N}{2} + 15$ with N as the vector length. Usually, AVO and AGU instructions are part of a program code. As the AGU instructions are executed first to program the AGU, 3 cycles have to be added for the very first AGU instruction and 1 cycle for all subsequent ones. This is due to the fact that the A-FEP can execute different instructions simultaneously. For subsequent AVO instructions, $\frac{N}{2} + 4$ cycles have to be added.

4.8 Component-Wise Lookup Table - Example of a Possible Future A-FEP Instruction

The complete CWL instruction allows to read out a vector from an LUT and to store the resulting vector in the MSS. Optionally it is possible to interpolate between two consecutive LUT entries. Input and output vector types have to be signed integer values with a size of 16 bit. The vector length is specified by the parameter l. For the LUT access, the ll ($1 \leq ll \leq 14$) most significant bits of the input vector X are used. The resulting length of the LUT is therefore 2^{ll}.

Each vector element i of $X[i]$ is split into two parts: x_{int} and x_{frac}. x_{int} is an unsigned integer stored in the ll most significant bits of $X[i]$. The remaining bits are denoted as x_{frac}. The AGU processing is similar to the other AVO instructions where only one input vector is read (e.g. vec_sum). For LUT access, x_{int} directly serves as address offset in case the parameter `ls` being part of the instruction word is set to zero, meaning that x_{int} represents an unsigned address value. Otherwise, the address is signed and 2^{15} has to be added to $X[i]$ first. This is necessary as otherwise, $2^{15} - 1$ and -2^{15} would be interpreted as $2^{15} - 1$ and 2^{15}

$$\forall\, 0 \leq i < 2^{ll},\ Y[i] = \begin{cases} f(i) & \text{if } ls = 0 \\ f(i - 2^{ll-1}) & \text{if } ls = 1 \end{cases} \tag{4.6}$$

The function f represents an arbitrary function stored in the LUT (e.g. sine, cosine,...). How it is stored depends on `ls` as illustrated in Table 4.15:

Y	$ls = 0$	$ls = 1$
$Y[0]$	$f(0)$	$f(-2^{ll-1})$
$Y[2^{ll-1} - 1]$	$f(2^{ll-1} - 1)$	$f(-1)$
$Y[2^{ll-1}]$	$f(2^{ll-1})$	$f(0)$
$Y[2^{ll} - 1]$	$f(2^{ll} - 1)$	$f(2^{ll-1} - 1)$

Table 4.15: $Y[i]$ LUT Organization

Once the vector element is read from the LUT, x_{frac} can be used to interpolate or extrapolate between its entries. Extrapolation is performed when $x_{int} = 2^{ll} - 1$ which is the last LUT entry. The following equations illustrate how the result vector of the CWL instruction is computed:

$$x = \begin{cases} uint16(X[i]) & \text{if } ls = 0 \\ X[i] + 2^{15} & \text{if } ls = 1 \end{cases} \tag{4.7}$$

$$x_{int} = \text{rsh}(x, 16 - ll) \tag{4.8}$$

$$x_{frac} = x \bmod 2^{16-ll} \tag{4.9}$$

$$y0 = Y[x_{int}] \tag{4.10}$$

$$y1 = \begin{cases} Y[x_{int} + 1] & \text{if } x_{int} < 2^{ll} - 1 \text{ (interpolation)} \\ 2 \times Y[x_{int}] - Y[x_{int} - 1] & \text{if } x_{int} = 2^{ll} - 1 \text{ (extrapolation)} \end{cases} \tag{4.11}$$

$$c1 = 2^{ll-1} \times x_{frac} \tag{4.12}$$

$$c0 = 2^{15} - c1 \tag{4.13}$$

$$Z[i] = \begin{cases} 2^{15} \times y0 & \text{if } li = 0 \text{ (no interpolation/extrapolation)} \\ c0 \times y0 + c1 \times y1 & \text{if } li = 1 \text{ (interpolation/extrapolation)} \end{cases} \tag{4.14}$$

Like for the other AVO instructions, the throughput of the pipeline has to be two vector elements per cycle. A possible pipeline processing for the CWL instruction could be:

1. The AGU is enabled only for the address computation of the input vector $X[i]$.

2. After the MSS delay of three cycles, the 16 bit value $X[i]$ is retrieved.
   (Same AGU processing like for the other AVO instructions!)

3. Once, the value $X[i]$ is available, it is modified in case the parameter `ls` is set to one.

$$x = \begin{cases} uint16(X[i]) & \text{if } ls = 0 \\ X[i] + 2^{15} & \text{if } ls = 1 \end{cases}$$

4. Based on the obtained value $X[i]$, x_{int} and x_{frac} can be determined:

$$x_{int} = \text{rsh}(x, 16 - ll)$$
$$x_{frac} = x \bmod 2^{16-ll}$$

   Then the address x_{int} is set to get the required vector element from the LUT ($Y[i]$).

5. In case of $ls = 0$, the output of the CWL operation is $Z[i] = Y[x_{int}]$ and no further processing is required. If not, the interpolation / extrapolation mode is enabled. For that, the necessary vector elements $y_0 = Y[x_{int}]$ and $y_1 = Y[x_{int+1}]$ have to be provided. Due to the data bus width of 32 bit, the latter is read automatically when reading $Y[x_{int}]$. Once the two vector elements are available, Equation (4.11) can be computed:

$$y1 = \begin{cases} Y[x_{int} + 1] & \text{if } x_{int} < 2^{ll} - 1 \text{ (interpolation)} \\ 2 \times Y[x_{int}] - Y[x_{int} - 1] & \text{if } x_{int} = 2^{ll} - 1 \text{ (extrapolation)} \end{cases}$$

6. Then the required interpolation factors can be estimated as

$$c1 = 2^{ll-1} \times x_{frac}$$
$$c0 = 2^{15} - c1$$

7. The resulting output vector $Z[i]$ is then obtained as

$$Z[i] = c0 \times y0 + c1 \times y1$$

The simple case is when no interpolation / extrapolation is required. For processing, the existing pipeline could be reused, although it is recommended to shift the LUT access in EX5, assuming the LUT read signals are set in EX2 ($X[i]$ is available in EX1). In contrast, additional instructions or additional adder or multiplier resources are required for step 5 to 7 to speed up processing. Alternatively these equations could also be computed by a short program consisting of a vector subtraction ($y1$), a vector multiplication (c_0), a vector subtraction (c_1) and a component-wise multiplication ($Z[i]$) based on `cpx32` vector elements where the imaginary part values are set to zero.

4.9 Design Comparison and Runtime Performance

4.9.1 Design Comparison A-FEP vs C-FEP

The main differences between the two designs are to be found in their processing engines (Table 4.16) as their MSS is almost identical. Instead of the Program Memory included in the A-FEP MSS, the MSS of the C-FEP contains twiddle factor and temporary data memories for DFT / IDFT computation with an overall size of 52 kB.
How these differences influence the performance especially when processing standards with short data sets is elaborated more detailed the next section.

Objective	C-FEP	- processes all required air-interface operations
	A-FEP	- processes all required air-interface operations - processes UC instructions (decreased communication overhead)
Architecture	C-FEP	- vector operations, DFT / IDFT - next command can be prepared when the PU is busy - pipeline has to be emptied after each operation - implementation of the complete CWL operation - for division, two operations are necessary - 11 to 16 cycles for initialization / termination
	A-FEP	- vector operation / AGU / GP instructions - multiple instruction processing - component-wise division using an LUT - fast paths for GP instructions - 14 to 17 cycles for initialization / termination - for division only one instruction necessary

Table 4.16: Design Comparison A-FEP vs C-FEP

4.9.2 Runtime Performance

The runtime performance of a DSP engine on the ExpressMIMO platform depends on two different factors:

- the DSP processing time and
- the time required for the communication flow between the main CPU and the DSP engines.

For a standard like IEEE 802.11p that is operating on short data sets, the second factor is of main importance as it determines the overall performance when executing the standard on the platform. Table 4.17 lists the number of cycles and the execution times of the IEEE 802.11p receiver algorithms presented in Chapter 3.3 for a frequency of 100 MHz. The results consider only the pure processing times, the communication overhead is neglected.

algorithm	**cycles A-FEP**	**cycles C-FEP**	**execution time A-FEP**	**execution time C-FEP**
energy detection	302	151	3.06 μs	1.51 μs
channel estimation (CWP)	45	43	0.45 μs	0.43 μs
data detection (16-QAM, init)	173	267	1.73 μs	2.67 μs
data detection (16-QAM)	114	129	1.14 μs	1.29 μs
data detection (64-QAM, init)	219	318	2.19 μs	3.18 μs
data detection (64-QAM)	341	387	3.41 μs	3.87 μs

Table 4.17: A-FEP Cycle Counts and Execution Times for the IEEE 802.11p Receiver

At a first glance, the C-FEP performs better for the energy detection. But it has to be considered, that the A-FEP performs the comparison to the threshold value itself while it is the main CPU in case of the C-FEP.
For channel estimation, only the cycle counts of the performed CWP operation are given, as the DFT has to be computed by the C-FEP. In case only one vector operation is executed, the processing times of the two designs are almost identical.
For the data detection, the performance of the A-FEP is better due to the reduced internal latencies of the pipeline architecture.

The presented results focus only on the pure processing times of the DSPs while the communication overhead is neglected. In the following, the A-FEP and C-FEP solutions will be compared to recent ASIPs from academia taking the examples of two different packet algorithms.

4.9.2.1 Auto-Correlation Based Packet Detection Algorithm

The packet detection algorithm presented in [76] is performed taking the STS of the IEEE 802.11a/n receiver. Similar to the presented packet detection algorithm for our IEEE 802.11p receiver, packet detection is performed by a sliding window over the incoming sample stream $r[d]$. The main difference is that the auto-correlation is not calculated taking the received and the reference STS but taking the first and the second half of the received STS. The resulting auto-correlation function for a single receiver can be expressed as

$$P_L[d] = \sum_{m=0}^{L-1} (r[d+m]^* \times r[d+m+L]) \tag{4.15}$$

where L is set to 80 samples which corresponds to half the size of the STS. Afterwards, the energy inside the current window is computed as

$$R_L[d] = \sum_{m=0}^{L-1} |r[d+m+L]|^2 \tag{4.16}$$

and the beginning of the packet is found if

$$|P_L[d]|^2 > \frac{|R_L[d]|^2}{2} \tag{4.17}$$

Otherwise the window is shifted and the whole procedure starts from the beginning.

Extending this packet detection algorithm to the 2x2 MIMO case, auto-correlation and energy computation are performed individually over the two receive streams while for comparison, the average results are used.

$$P_{L,avg}[d] = \frac{1}{2} \sum_{j=1}^{2} P_{jL}[d] \tag{4.18}$$

$$R_{L,avg}[d] = \frac{1}{2} \sum_{j=1}^{2} R_{jL}[d] \tag{4.19}$$

The resulting set of instructions and the cycle counts of the A-FEP for the SISO case are provided in Table 4.18.

	instructions	**cycles**
$P_L[d]$	agu_cfg (6x)	9
	vec_move	L/2 + 4
	agu_cfg (4x)	4
	vec_mult	L/2 + 4
	agu_cfg (2x)	2
	vec_sum	L/2 + 4
$R_L[d]$	agu_cfg (2x)	2
	vec_square_modulus	L/2 + 4
	agu_cfg (2x)	2
	vec_sum	L/2 + 4
	agu_cfg (2x)	2
	vec_square_modulus	L/2 + 4
$\|P_L[d]\|^2 > \frac{\|R_L[d]\|^2}{2}$	nop (7x)	7
	lw	5
	nop	1
	lw	5
	nop	1
	bgt	8

Table 4.18: A-FEP Instructions for the Auto-Correlation Based Packet Detection

In general, this results in a total amount of $6 \times \frac{L}{2} + 72$ cycles including the GP instructions and $6 \times \frac{L}{2} + 24$ cycles if the latter are not taken into account. If the C-FEP is used instead, the algorithm can be simplified (see Table 4.19).

	operations	**cycles**
$P_L[d]$	vector move	L/2 + 11
	vector multiplication + sum	L/2 + 12
$R_L[d]$	vector square_modulus + sum	L/2 + 12
$\|P_L[d]\|^2 > \frac{\|R_L[d]\|^2}{2}$	vector square_modulus + sum	L/2 + 11

Table 4.19: C-FEP Operations for the Auto-Correlation Based Packet Detection

The resulting amount of cycles is $4 \times \frac{L}{2} + 46$ when only the pure processing time is considered.

But how do these results change when the communication overhead is considered? Table 4.20 provides the resulting cycles counts and processing times for the A-FEP, the C-FEP and ASPE A for the SISO and the MIMO case.
It can be observed that the performance of A-FEP and ASPE A are almost identical for this packet detection algorithm, although the latter is optimized for the IEEE 802.11a/n standard. Comparing A-FEP to C-FEP, the communication overhead of the first is lower due to reduced internal latencies of the design and due to the GP instructions that reduce the communication overhead between the A-FEP and the main CPU. Therefore it can be stated that the implementation of algorithms is simplified when using the A-FEP as the required processing can be done by only one resource. No synchronization between the different processing engines is required.

Solution	**cycles (SISO)**	**cycles (MIMO)**	**execution time (SISO)**	**comm. overhead (SISO)**	**execution time (MIMO)**	**comm. overhead (MIMO)**
ASPE A	296	650	2.96 μs	-	6.5 μs	-
A-FEP	264	572	2.64 μs	0.48 μs	5.72 μs	0.64 μs
C-FEP	312	465	3.12 μs	1.2 μs	4.65 μs	1.2 μs

Table 4.20: Design Comparison for the Auto-Correlation Based Packet Detection

4.9.3 Energy Based Coarse Packet Detection Algorithm

The second example illustrates the performance differences when the A-FEP is compared to a specialized ASIP for synchronization and acquisition. The Sync-ASIP has recently been presented in [77] and executes a coarse packet detection algorithm. Here, two energy values denoted as a and b are computed over $L = 64$ vector elements and are divided through each other:

$$a = \sum_{n=0}^{L-1} |r_{n-L}|^2 \tag{4.20}$$

$$b = \sum_{n=0}^{L-1} |r_{n+L}|^2 \tag{4.21}$$

$$m = \frac{a}{b} \tag{4.22}$$

In case the result is beyond a certain threshold the probability that the beginning of the packet can be found in the current window is high and an auto-correlation based packet detection algorithm is applied to find the exact beginning of the packet.
The resulting set of instructions and the cycle counts of the A-FEP are provided in Table 4.21.

	instructions	**cycles**
a,b	agu_cfg (6x)	9
	vec_abs_square	L/2 + 4
	agu_cfg (2x)	2
	vec_abs_square	L/2 + 4
	agu_cfg (2x)	2
	vec_sum	L/2 + 4
m	agu_cfg (4x)	2
	vec_cwl	7
	agu_cfg (3x)	3
	vec_mult	4
	nop (7x)	7
	lw	5
	nop	1
	lw	5
	nop	1
	bgt	8

Table 4.21: A-FEP Instructions for the Energy Based Coarse Packet Detection

Including GP instructions, this results in a total amount of $3 \cdot \frac{L}{2} + 68$ cycles, where $3 \cdot \frac{L}{2} + 12$ cycles are required for the pure data processing. If the C-FEP is used instead, the algorithm can be simplified again (see Table 4.22).

	instructions	**cycles**
a,b	vec_abs_square + sum	L/2 + 12
	vec_abs_square + sum	L/2 + 12
m	vec_cwl	15
	vec_mult	11

Table 4.22: C-FEP Operations for the Energy Based Packet Detection

Table 4.23 lists the resulting cycles counts and processing times for the A-FEP, the C-FEP and Sync-ASIP for a frequency of 100 MHz. As expected, the weakly programmable but specialized Sync-ASIP performs better than the flexible FEP solutions. Besides, it is worth to note that for smaller vector lengths, the communication overhead of the A-FEP is only half of the pure data processing time while it is twice in case of the C-FEP.

Solution	**cycles**	**execution time**	**communication overhead**
Sync-ASIP	31	0.31 μs	-
A-FEP	108	1.08 μs	0.56 μs
C-FEP	114	1.14 μs	2.29 μs

Table 4.23: Design Comparison for the Energy Based Coarse Packet Detection

4.10 Conclusions

As an alternative to the C-FEP, we have presented an ASIP solution for flexible front-end processing that fulfills the latency requirements of the latest wireless communication standards. The A-FEP can be included as an additional block in the baseband engine for the execution of latency critical tasks while DFT / IDFT and latency non-critical tasks can be executed by the C-FEP. Observed timing differences are due to the reduced communication overhead of the A-FEP which results in a significant performance gain when operating on standards with short data sets, and which results in a simplified algorithm design.
Besides the comparison between these two solutions, the A-FEP has further been compared to a previous ASIP version and to recent ASIPs from academia. In contrast to the first, the A-FEP exhibits a higher frequency and a greater functionality. For a packet detection algorithm its performance is similar to the ASPE A - a design tailored to the processing of the IEEE 802.11a/n standard. As expected, the performance is worse than the one of a specialized ASIP for synchronization and acquisition.

Chapter 5

Flexible Sample Rate Converter Design

After presenting the IEEE 802.11p receiver chain including FEP, Deinterleaver and Channel Decoder and after focusing on an alternative FEP solution which is more appropriate for standards with short data sets, we conclude with the only missing DSP engine of the IEEE 802.11p chain - the Preprocessor. The Preprocessor connects the A/D, D/A converter interface with the remaining baseband engine and is responsible among others for I/Q imbalance correction, sample synchronous interrupt generation, framing and sample rate conversion. Most critical is the Sample Rate Converter (SRC). Its behavior can change dynamically as it can be tuned to any frequency band in the wireless communication domain. In the past, one SRC was dedicated to each standard of interest. For the ExpressMIMO platform, this approach is too space consuming why one fractional SRC architecture capable to process up- and downsampling is preferred. To ensure a low phase noise at the A/D and D/A converters, they are triggered with a fixed master clock. Dealing with the relation between the different sampling rates is therefore in the responsibility of the SRC. In this chapter we propose an efficient design for fractional sample rate conversion and present it in the context of the whole Preprocessor DSP engine. The different models that have been designed comprise C-models for simulation and a VHDL prototype that has been synthesized for the ExpressMIMO platform.

5.1 Motivation

The Preprocessor DSP engine establishes the connection between the A/D and D/A (ADA) converters through the ADA interface and the remaining baseband engine. This task is quite challenging as the clock frequency at the converter side is 32.768 MHz while it depends on the executed wireless communication standard on the baseband side. For DAB, for instance, the baseband sampling frequency is 2.048 MHz, while for IEEE 802.11p it is set to 10 MHz (Fig. 5.1). This results in a resampling factor of 15 for DAB and 3.2768 for IEEE 802.11p.
The relation between these different sampling rates is commonly handled by SRCs which are well-known architectures applied not only in wireless communication systems but also in image processes for instance. For SDR systems, they are one of the most critical and most demanding elements [104].
Challenges when designing an appropriate SRC solution for the ExpressMIMO platform are:

- A detailed analysis of nowadays wireless communication standards has shown that the SRC has to support a frequency range of 3 MHz $\leq f_{samp} \leq 61,44$ MHz with a resolution of 1 Hz.

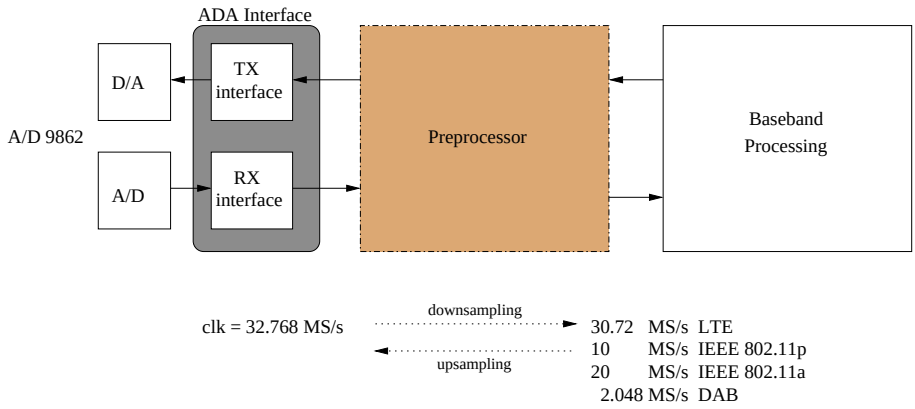


Figure 5.1: The Preprocessor connects the ADA Converters with the remaining Baseband Engine

- In the past usually one dedicated SRC was used per standard. For this wide frequency range, this approach is not applicable as the required resources are far beyond what is available on the FPGA target. The SRC has therefore to support all possible sampling rate ratios (integer and fractional ones) by only one architecture.
- Apart from that the ExpressMIMO platform can process up to four different channels in RX and up to four different channels in TX. Each channel is defined through its own set of parameters.

  Thus, when switching between two channels, the system may change its behavior dynamically at runtime.

These design challenges lead to different processing requirements which can be grouped in functional and non-functional ones. From the platform perspective it is of atmost importance that the required amount of DSP48E slices is reduced as much as possible. This task is not that simple as due to the high bandwidth of the signal coming from the A/D converters, the data rate is very high. This leads to a higher hardware complexity and a higher power consumption and results in a higher number of DSP48E slices and thus in a cost intensive application. Besides, the design of the Preprocessor has to follow the same design approach like all other DSPs on the platform and should be embedded in the standardized DSP shell.
Additionally the functional requirements comprise

- the preference of a generic design being able to perform fractional up- and downsampling by using the same architecture. Upsampling / Downsampling stands for increasing / decreasing the sampling rate. For transceivers where the data rate at the ADA converter side is higher than the one at the baseband engine, upsampling is performed in TX, while downsampling is performed in RX.
- the support of three different modes: (1) only reception, (2) only transmission and (3) reception and transmission simultaneously. From the platform perspective, the channels of RX and TX are executed in parallel while the SRC processes them consecutively in a Round Robin fashion. Therefore the channel switch has to happen within one cycle.
- the avoidance of aliasing when resampling. In this context it is important to tradeoff the length of the lowpass filter and thus the number of multipliers in the design with the complexity of calculating missing filter coefficients.

- the calculation of intermediate values of a discrete-time signal such that a certain frequency band of the signal is not distorted [105].
- that a high performance has to be guaranteed to meet the throughput and latency requirements of the different wireless communication standards.
- that the SRC takes care of the difference between the sampling rates. This approach allows to fix the master clock of the ADA converters to decrease the phase noise. [105]

5.1.1 Related Work

For the SRC design two different approaches are possible: (1) analog solutions to generate a variable clock for the ADA converters or (2) digital solutions where the ADA converter clock is fixed. The first comes with the drawback that the observed phase noise differs between the generated clocks and may lead to a significant performance drop. As the clock is fixed for the second approach, a high performance one with low phase noise can be selected at design time. Apart from that, digital filters come with the advantages of high precision, of a possible multiplexing between different channels and of thermal stability but they are also characterized by a limited bandwidth and quantization noise.
In the following different solutions from the analog and the digital domain are presented. The solution finally chosen for the ExpressMIMO platform is a digital one to keep the phase noise as low as possible.

5.1.1.1 Analog Solutions

Important terms when talking about how to choose the right analog solution are *phase noise* and *jitter* that both describe the variation of the ideal signal period which is equal to the instability of the sampling clock. While the expression phase noise is used when talking about the frequency domain, jitter is in the time domain [106]. For sinusoidal signals, the phase noise can easily be estimated while for Gaussian input signals the derivative of each sample point has to be calculated [107].
Another important term is *aperture jitter* or *aperture uncertainty* that describes the variation between the samples in the encoding process [108]. Aperture jitter influences the system performance in three different ways: (1) it increases the system noise, (2) it increases the ISI between the samples and (3) it increases the uncertainty of the sampled signal phase. Worst case values occur, when sampling a sinusoidal signal with the highest possible frequency in the Nyquist band (= sampling with half of the input sampling rate).

The basic functionality of all analog solutions is the following: The input signal is converted into a digital signal by an A/D converter that is triggered with a variable clock. Different chips that have been available on the market for years, support two different circuit technologies allowing this flexible clock adjustment. These technologies are Phase-Locked Loops (PLL) and Direct Digital Synthesizers (DDS).

A PLL is a closed loop frequency control system supporting fractional ratios between the sampling rates. Its functionality is based on the measurement of the phase difference between the incoming and outgoing samples of the control oscillator. To convert the resulting voltage into a frequency, a VCO (Voltage Controlled Oscillator) becomes necessary that is sometimes already included in the PLL. Advantages comprise low costs, widely availability, well-known architecture and that the

PLL output can be locked to the reference clock of the input phase. An efficient PLL solution of high performance is the ADF4157 [109]. This design is a 6 GHz fractional-N frequency synthesizer with a 25 bit fixed modulus and a subherz frequency resolution that consists of a low noise digital phase frequency detector, a precision change pump and a programmable reference divider.

For DDS architectures, the reference clock is scaled down by a factor that is provided to the DDS via a programmable tuning word. The tuning word typically has a length of 24 to 48 bit. For new generation technologies, the DDS power consumption is similar to the one of PLLs. Other advantages include the micro-hertz tuning resolution, the high output frequency span, the tunable reference clock oscillator that allows a higher operation range than a standard VCO, the possibility of fast frequency changes, the manual system tuning, the digital control interface and the phase-continuous frequency hops with no over/undershoot or analog-related loop settling time anomalies. On the other side, DDS designs cannot achieve exact frequencies for a division factor unequal to a power of two. Taking the example of 20 MHz, the obtained result is 19,9999999954 MHz. To overcome this drawback of an in-accurate frequency generation, [110] has presented a programmable modulus that leads to an exact frequency generation.
Another efficient DDS design is the AD9913 [111]. It comes with a low power consumption of max. 98,4 mW, supports a frequency range of up to 250 MHz (worst case frequency resolution: 0,058 Hz), has an analog output up to 100 MHz and features a 10 bit D/A converter.

Comparing DDS to PLLs the following observations can be made [112]:

- DDS support fast frequency changes which make these architectures more agile than PLLs.
- The DDS supports a higher frequency resolution of up to one millionth of a Hertz.
- The DDS performance is higher than the one of PLLs.
- Multiple DDS can be synchronized to support among others quadrature phase offset relations.
- PLLs allow to lock their output to the input phase of a reference clock.
- DDS have a lower output phase noise.

5.1.1.2 Digital Solutions

The major aim when designing SRCs for SDR systems is the fractional resampling support in the all-digital domain by using high performance DSPs. General overviews of simple SRC solutions are provided among others in [113], [114], [115], [116] or [117]. These designs comprise the sample and hold method as well as different interpolation approaches listed below. The basic idea for all arbitrary ratio interpolation schemes is to use an analog reconstruction filter for which the output is resampled. Additionally, a lowpass filter is required for both, upsampling and downsampling. For upsampling, the lowpass filter is needed as this process may add undesired spectrum images to the original signal, despite of the fact that the Nyquist-Shannon sampling theorem is fulfilled. This theorem states that aliasing occurs in case the sampling frequency is greater than two times the analog frequency of the signal to be resampled. For downsampling where the sampling rate is decreased, there is a high probability that this theorem is not satisfied. Thus the aliased frequency signal components that are not distinguishable from the original ones have to be filtered out by a lowpass filter which has to guarantee a high stopband attenuation.

For lowpass filter design, two different filter types can be employed: Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters. Before choosing the appropriate solution, one has to consider that an arbitrary fractional rate change factor can result in relatively large values of the resampling rate. The passband of the filters has to be very narrow for large values of the upsampling factor, while the first stopband is centered about the inverse of the downsampling factor for anti-aliasing. Besides, the transition-band has to be narrow for large values of the downsampling factor. Designing an FIR filter with these constraints leads to a very long impulse response. Although providing linear phase characteristics, main drawbacks of FIR filters are thus the high quantity of filter coefficients. On the other side, FIR filters are stable and linear phase which means that their phase changes proportional to the frequency. This makes these filters the appropriate solution for SRC designs. Due to the linear phase property the filter has a constant group delay so that no phase distortion can be observed.
The most significant error contribution arises from the coefficient quantization resulting in changes of the magnitude response but not of the filter phase. To decrease the quantity of required hardware resources like multipliers, the FIR filter can be transformed in a polyphase representation as enhanced by [118]. In addition, [119] states that the main advantages of such a design include lower computational requirements, a lower sensitivity to the filter coefficient length, less finite arithmetic effects, lower order filter design and implementations, and less storage of filter coefficients.
In contrast, IIR filters do not provide linear phase characteristics but have a lower complexity in terms of the filter order by fulfilling the same magnitude response requirements. In case of polyphase implementations the number of coefficients is heavily increased since the number of coefficients in the feed-forward branches of a direct implementation of IIR filters is multiplied by the number of polyphase branches. Although a polyphase implementation of IIR filters has no real advantage over a polyphase implementation of FIR filters in terms of performance.

In the following, an overview of different digital SRC solutions is provided. As we need a fractional design for the ExpressMIMO platform we mainly focus on fractional solutions although there are a lot of different publications about flexible SRCs operating on integer ratios available.

- The simplest method for SRC filter design is the **sample and hold method** where the analog signal is first sampled and the obtained value is then hold afterwards. The solution is easy implementable and requires only few resources but the achieved performance is too low for SDR applications. For multimodal processing one could imagine several FIFOs that are connected with the sample and hold block via a demultiplexer. A context switch becomes necessary in case a value still has to be hold when the switch is performed.

- The classical SRC method is to **increase the sampling rate by zero insertion and to decrease the sampling rate afterwards** (Fig. 5.2).

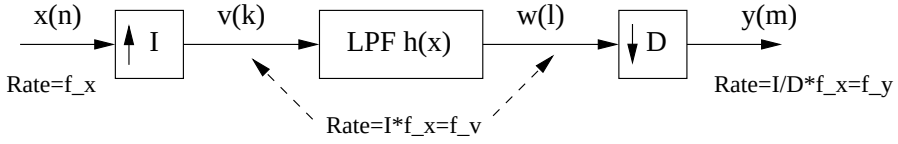


Figure 5.2: Classical SRC Approach

The interpolation stage increases the sampling rate of the received signal $x(n)$ by a factor I to obtain the interpolated signal $v(k)$.

$$v(k) = \begin{cases} x(\frac{k}{I}), & k = 0, \pm I, \pm 2I, ... \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

As interpolation and decimation stage are not merged one lowpass filter (LPF) is sufficient. $h(l-k)$ is the impulse response of this filter.

$$w(l) = \sum_{k=-\infty}^{\infty} h(l-k)v(k) = \sum_{k=-\infty}^{\infty} h(l-kI)x(k) \tag{5.2}$$

Afterwards, the filtered signal is downsampled by a factor D and the whole resampling chain can be expressed as

$$y(m) = w(mD) = \sum_{k=-\infty}^{\infty} h(mD-kI)x(k) \tag{5.3}$$

with

$$k = \lfloor \frac{mD}{I} \rfloor - n \tag{5.4}$$

By combining Equation 5.3 with Equation 5.4 we obtain

$$y(m) = \sum_{n=-\infty}^{\infty} h(mD - \lfloor \frac{mD}{I} \rfloor I + nI) * x(\lfloor \frac{mD}{I} \rfloor - n) \tag{5.5}$$

$$= \sum_{n=-\infty}^{\infty} h((mD) \mod I + nI) * x(\lfloor \frac{mD}{I} \rfloor - n) \tag{5.6}$$

The required resource consumption of this design is very high and context switches in case of multimodal processing become more complex than for the sample and hold method. [120] has shown that this standard approach can be transformed in a more efficient design by taking advantage of a polyphase filter structure for the required lowpass filter. Employing this architecture speeds up the time needed for the filter process, but does not decrease the number of filter coefficients.

Furthermore [121] describes an architecture where the presented standard approach is combined an adjustable number of times which is especially of interest in case of high ratios between the sampling rates. Filter reconfigurability is not required as all filter taps can be precomputed and hardwired.

- [122] presents a solution where a **lowpass filter is combined with A/D and D/A converters**. This approach is not appropriate for the ExpressMIMO platform due to the limited resources and due to the choice of one single master clock.
- Another standard approach are **interpolation filters** that compute the missing output samples / filter coefficients at runtime [121]. To reduce the space consumption, this filter is usually combined with the required lowpass filter. The performance therefore depends on two different factors: (1) on the number of stored filter coefficients and (2) on the selected

interpolation method. For the latter it can be stated that the higher the interpolation polynomial is, the higher is the computational effort and the higher is the space consumption. One option to deal with this challenging task is the filter coefficient calculation in a recursive way as presented in [123]. But this increases the latency and results in additional resources needed for the filter coefficient calculation.

A good overview of the fundamentals and the implementation of interpolation in digital modems can be found in [124] and [125].

For interpolation, different solutions are possible:

1. **nearest-neighbor interpolation**:
   This interpolation technique which is also known as proximal interpolation or point sampling is the easiest to implement and comes with a simple hardware architecture. The missing sample values are chosen by allocating the value of the nearest neighbor. To guarantee a high performance, the incoming signal has to be oversampled to lower the resulting phase noise as much as possible. But still, this technique has only a very low efficiency when compared to other interpolation methods.

2. **linear interpolation**:
   The linear interpolation comes with a better performance than the nearest-neighbor interpolation. As the name suggests, missing values are computed with the help of a linear function. Despite a low space consumption, good filter characteristics are possible but the filter coefficients need to be precomputed and stored in memory. A possible design improvement is the replacement of the lowpass filter by a polyphase filter structure.

3. **polynomial interpolation**:
   The polynomial interpolation is a generalization of the linear interpolation. The main difference is that the linear function is now replaced by a polynomial one of higher degree which results in a higher performance of the design [126]. Compared to linear interpolation, the calculation of the interpolating polynomial is computationally expensive and the improvement is not as much as between nearest-neighbor and linear interpolation. So the question is if the required space consumption justifies this obtained performance gain.
   For higher polynomials it is said that only one polynomial exists that interpolates the known samples of the incoming signal. This polynomial is also called *Lagrange polynomial*. The Lagrange interpolator is a polynomial constructed in such a way that each sample is exactly represented by a function which has zero values at all other sampling points. Its formula is only simple for low-order polynomials. The filter coefficients have to be calculated depending on the input samples so that the Lagrange method finally serves as coefficient design procedure ([104]).
   In general one can say that the Lagrange interpolation formula $y(t)$ fits an $(M-1)$th order polynomial $p_k(t)$ to a set of M data points of the incoming signal $x(t_k)$:

$$y(t) = \sum_{k=0}^{M-1} x(t_k) p_k(t) \tag{5.7}$$

with

$$p_k(t) = \prod_{t=0, t \neq k} \frac{t - t_l}{t_k - t_l} \tag{5.8}$$

One possibility of realizing polynomial interpolation is the Farrow structure where the value of the input signal between existing samples is estimated ([127], [128]). A different solution are Laguerre and Kautz filters ([129]) which are higher order forms of the unit delay elements of an FIR filter. By replacing the unit delays with these architectures, one (in case of Laguerre filters) or two (in case of Kautz filters) degrees of freedom are added to the final design, leading to a better performance.

Another drawback of the polynomial interpolation is Runge's phenomenon where the interpolation polynomial may oscillate wildly between the data points for higher order polynomials. A way of overcoming this drawback is to use a multirate structure where the interpolation is carried out at a higher sampling frequency. To meet the requirements in the pass- and the stopband, a digital pre-filter should be designed [130].

Most efficient in terms of performance is a solution based on polynomial interpolation in combination with CIC filters as illustrated by [131]. CIC (Cascaded Integrator Comb) filters are a class of FIR filters with only lowpass characteristics, a linear phase response and a constant group delay. Compared to FIR filters, they perform better for resampling factors higher than 10, due to their higher computation efficiency ([132], [133]). The two basic building blocks are called *integrator* which is a single pole IIR filter with unity feedback coefficients and *comb* which is an odd-symmetric FIR filter. Although CIC filters are equivalent to N FIR filters with rectangular impulse responses, an additional FIR filter at a low sampling rate may be added because of the passband droop (unstable narrow passband). This filter equalizes the passband droop and performs a low rate change, usually by a factor between two and eight. Advantages of CIC filters include the lack of multipliers and thus the lack of filter coefficients, the simple regular structure based on two basic building blocks, little control and low costs. Main disadvantages are the integer resampling, the different architectures for upsampling and downsampling, that the bandwidth and frequency response outside the passband are severely limited and that CIC filters are only useful for large ratios while for small ratios, FIR filters are preferred.
In [134] a solution based on time-variant CIC filters is proposed. Upsampling is performed by zero-insertion while downsampling corresponds to picking each m-th sample of the incoming sample stream. This architecture overcomes the traditional drawback of only integer ratio support and enables a fractional ratio between the sampling rates. Unfortunately the structure is quite space consuming as it is a combination of decimators and interpolators. Further an extra output clock has to be generated which leads to a higher complexity of the overall design.

4. **spline interpolation**:
   The spline interpolation is the most hardware consuming interpolation method [135]. A spline is a piecewise polynomial of small degree, so that problems due to Runge's phenomenon cannot be observed. The easiest approach is the linear spline interpolation where existing data points are connected by straight lines. Better are cubic splines that represent a cubic polynomial between two existing samples, or even quadratic

splines. The spline depends on the previous and the following sample why it has to be computed at runtime. An arbitrary sample rate conversion using B-spline interpolation for SDR has been introduced by [136]. B-splines are unequal to zero only for a few samples and allow an accurate approximation of the ideal filter response.

5. **Whittaker-Shannon interpolation**:
   The aim of this method is to reconstruct a continuous-time bandlimited signal $x(t)$ from a set of equally spaced samples $x[n]$ with the help of the ideal impulse response $sinc(t)$.
   $$x(t) = \sum_{n=-\infty}^{\infty} x[n] sinc(\frac{t - nT}{T}) \tag{5.9}$$
   The Whittaker-Shannon interpolation is not applicable for SRCs as it only works for infinite signals. So bandlimiting of $x(t)$ becomes necessary.
6. **bandlimited interpolation**:
   Bandlimited interpolation has been introduced in [137] and [138] and is a mixture between linear interpolation and Whittaker-Shannon interpolation. SRCs based on this approach are easy implementable and provide the same architecture for upsampling and downsampling. The filter coefficients have to precomputed and stored in a local memory.

Based on the presented SRC filter solutions, [139] and [105] concentrate on how flexible SRC filters could efficiently be implemented. They conclude, that a single-stage filter scheme is not well-suited for preprocessing prior to fine interpolation. As the filtering should be done at a high input sample rate, this scheme is computationally intensive and a large number of filter taps is required. Besides, the spectral characteristics of the filter itself must be changed for different output sample rates. Interpolation is considered as the optimum solution when dealing with fractional ratios between sampling rates. However the underlying hardware structure should not be underestimated as power consumption is a major issue in mobile communication systems. That is why there is a need for hardware architectures which enable efficient implementations of the necessary filtering tasks.

5.1.2 Contributions

The main contribution presented in this chapter is the design of a fractional SRC for the Express-MIMO platform which is based on the bandlimited interpolation algorithm. Its architecture can process up to four different channels in RX (downsampling) and up to four different channels in TX (upsampling). All channels are executed on the same parameterizable hardware architecture. To guarantee a continuous filter processing, context switches between them happen instantaneously within one cycle.
The provided SRC models comprise fixed- and floating-point C-models for quantization measurements and analysis of the filter characteristics, as well as a VHDL prototype.
The SRC is embedded in the Preprocessor DSP engine that establishes the connection between the ADA interface and the remaining baseband engine. To finalize the IEEE 802.11p receiver chain a first prototype of the Preprocessor has been described in VHDL and evaluated using Modelsim.

5.2 Functional Specification

The functional specification is split over two different parts. First the Preprocessor is specified and functional details to be considered for the design of the SRC are provided. Then we focus on the specification of the SRC itself.

5.2.1 Preprocessor Specification

The Preprocessor connects the external RF module with the digital baseband processing engine. To establish this connection, the standardized DSP shell has slightly been modified by a dedicated interface for a direct access between the processing unit and the ADA interface (Fig. 5.3). The latter handles the (de)multiplexing of the complex samples coming from and going to the A/D and D/A converters. In RX / TX, the signal provided by the A/D, D/A converters has a resolution of 12 bit / 14 bit. Sign extension and bit removal become necessary as the Preprocessor operates on samples in a Q1.15 format. These tasks are handled by the ADA interface as well.

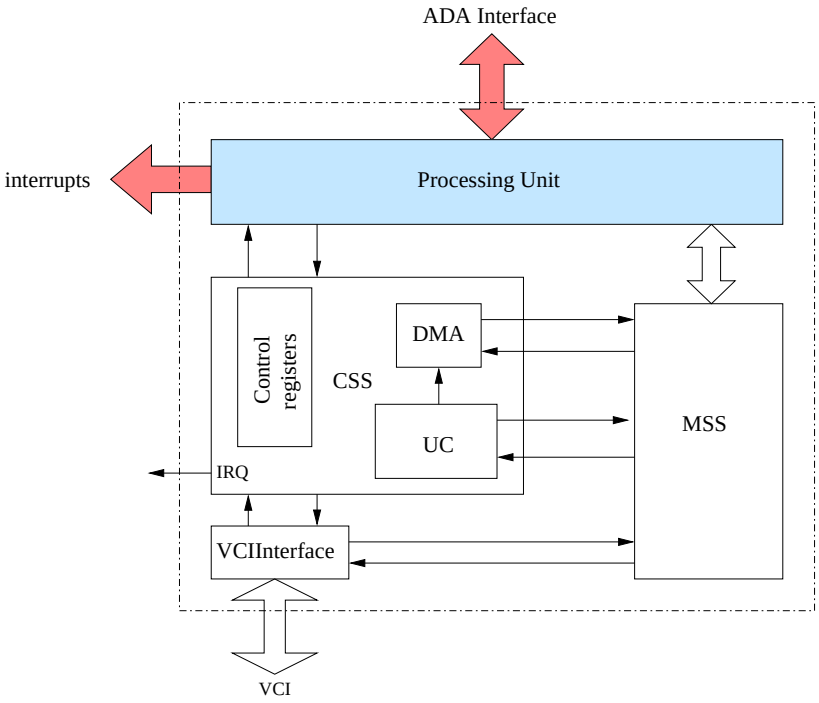


Figure 5.3: Modified Standardized DSP Shell

The main Preprocessor tasks are:

- Interface to the ADA converters
- I/Q imbalance correction. The quadrature offset compensation eliminates the errors caused by I/Q imbalance. More specifically, these errors result from amplitude and phase impairments between the local oscillator paths as well as from mismatches between I and Q branch after the analog down conversion.
- NCO for carrier frequency adjustment

- Basic signal processing functions like sample rate conversion
- Sample synchronous interrupt generation. An interrupt is generated for each RX channel after a given number of samples has been stored in the MSS. This is called the *acquisition cycle* of the Preprocessor. These interrupts may inform the main CPU to trigger a DMA transfer or they could inform the local UC so that the data is automatically streamed into a known baseband memory location.

To guarantee a high performance of these tasks, they are split over different internal modules which are I/Q imbalance (I/Q), a pre-distortion unit in TX (PD), NCO and SRC. Each of the two different modes supports four different channels that may possess a different set of parameters. In RX, the incoming samples provided by the ADA interface pass I/Q, NCO and SRC before the samples are stored in FIFOs in the MSS. In TX, the outgoing samples are loaded from FIFOs in the MSS and pass SRC, NCO, I/Q and PD before they are passed to the ADA interface. All modules are supervised by a global Preprocessor Control Unit as illustrated in Fig. 5.4. Main tasks of this state machine are to schedule the configured / active channels in a Round Robin fashion, to trigger the data write / read requires to the ADA interface and to the MSS, to update the parameters required to program the different modules if necessary, to generate the interrupts at the end of an acquisition cycle and to supervise the channel switch between two channels.

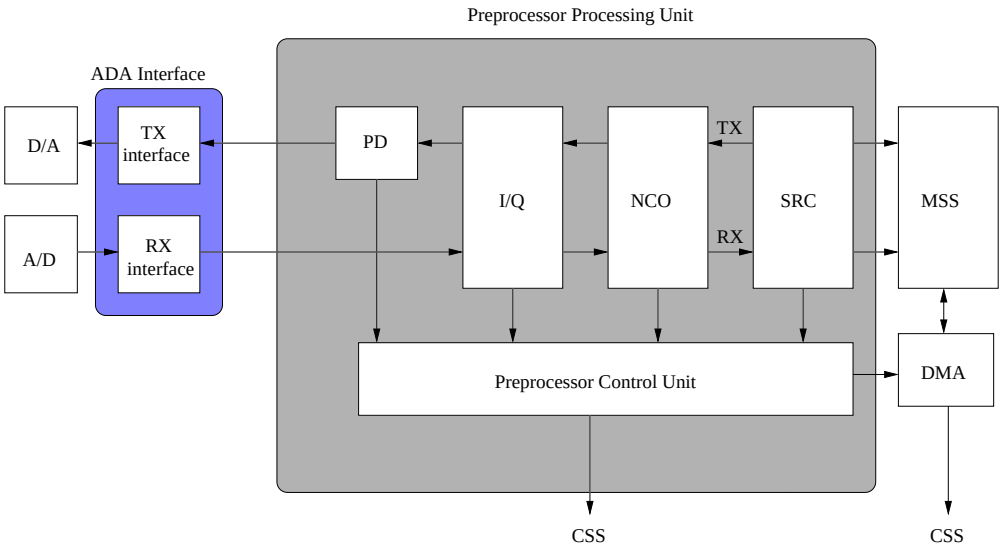


Figure 5.4: Preprocessor Architecture

Possible channel states are active (channel can be scheduled), inactive (channel currently not needed) and suspended (channel shall be processed, but without sample passing). To avoid an unpredictable behavior, channel parameter updates are only possible when a channel is not executed. This requires a minimum configuration of two different channels. A channel switch may occur after a fixed number of generated output samples. It can be handled flexibly at runtime depending on a CSS parameter providing the maximum number of samples to generate before switching. The minimum number of samples to be processed per channel depends on the time necessary to perform a channel switch in the whole DSP engine. It is important that the continuous processing of the Preprocessor is guaranteed, meaning that the channel switch has to happen instantaneously.

The different internal modules communicate via a handshaking protocol that is illustrated in Fig. 5.5. This protocol guarantees valid data transfers and stops the processing chain in case of absence of data. When a module provides new data, the data request signal REQ is set at the same time. So the module requests another one to take its data. The data transfer was successful once the acknowledgement signal ACK is received.

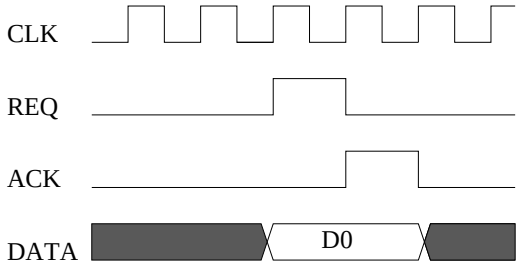


Figure 5.5: Handshake Protocol

The MSS and the memory space included in the ADA interface both contain different FIFOs for input and output sample storage. These FIFOs are autonomous components that manage their own memory space. To avoid sample loss, they inform the Preprocessor in case they are almost full or almost empty.

The MSS of the Preprocessor is build of

- a context memory for each of the modules. These memories are required in case of a channel switch to store the context of the previously processed channel and to provide the context of the next channel to process. Their size is flexible and depends on the amount of data to be stored.
- eight 32 bit input FIFOs with a size of 4 kB. For each channel, two FIFOs are provided. This is due to the fact as the SRC sample output port has a width of 32 bit while the DMA data bus width is set to 64 bit.
- eight 32 bit output FIFOs with a size of 4 kB.
- the filter coefficient memory of the SRC with a size of 2 kB.
- one parameter memory per module to enable parameter updates without conflicts. For the SRC, each channel has two parameter memories. While one is needed during processing, the other one can handle parameter updates triggered by the main CPU. Currently, the size of the SRC parameter memory has been fixed to 512 Byte.

Based on this functional specification, the following statements for the SRC design can be made:
(1) The parameter updates are handled by the Preprocessor Control Unit and are therefore not in the responsibility of the SRC.
(2) The moment in time when a channel switch has to happen is determined by the Preprocessor Control Unit. The SRC has to guarantee that the channel switch happens instantaneously once it is informed about this event.
(3) The communication between SRC and NCO has to follow the presented handshaking algorithm.
(4) The suspended mode is handled by the Preprocessor Control Unit. The SRC works as usual. Only difference is that no samples are passed.

5.2.2 SRC Specification

The SRC filter design is based on the bandlimited interpolation algorithm presented in [137] and [138]. Main advantages of this algorithm are (1) that no different architectures for up- and down-sampling are required, (2) that the architecture can be optimized for an efficient filter design and (3) that the combination of Whittaker-Shannon interpolation and linear interpolation results in a high performance with a reasonable space consumption.

5.2.2.1 Derivation of the Filter Structure

In the remainder of this chapter, we will denote the sampling rate at the filter input as $F_1 = \frac{1}{T_1}$ and the sampling rate at the filter output as $F_2 = \frac{1}{T_2}$. The relation between T_1 and T_2 is not predetermined and can even express fractional ratios between the sampling rates. The third sampling rate in the system is the one of the filter itself, denoted as $F_3 = \frac{1}{T_3}$.
The filter derived in this section is a combination of an FIR lowpass filter and a linear interpolation filter. To obtain a high performance, the ideal filter response is multiplied with a window function (please refer to Chapter 5.2.2.2) to obtain the basis waveform of the lowpass filter, $g(t)$.

The analog representation of a digital signal $x(nT_1)$ is computed as

$$x(t) = \sum_n x(nT_1)g(t - nT_1) \tag{5.10}$$

To get $x(t)$ at a different sampling rate with timing T_2, the equation above can be expressed as

$$x(kT_2) = \sum_n x(nT_1)g(kT_2 - nT_1) \tag{5.11}$$

The digital filter response $g(n)$ is sampled at a sampling rate with timing T_3 where

$$T_1 = M * T_3 \tag{5.12}$$

with M as the oversampling factor. So Equation 5.11 can be rewritten as

$$x(kT_2) = \sum_n x(nT_1)g(kT_2 - nMT_3) \tag{5.13}$$

k has to be chosen so that $k'T_3 \leq kT_2 < (k'+1)T_3$ or equivalently $k' = \lfloor k\frac{T_2}{T_3} \rfloor$. k' represents a known filter coefficient that has to be pre-stored in the Preprocessor MSS while k represents a filter coefficient that has to be computed with the help of the linear interpolation function. Considering this expression of k, Equation 5.13 can be approximated as

$$x(kT_2) \approx \sum_n x(nT_1)\left[(1-\alpha_k)g((k'-nM)T_3) + \alpha_k g((k'-nM+1)T_3)\right] \tag{5.14}$$

with

$$\alpha_k = k\frac{T_2}{T_3} - k' \tag{5.15}$$

For an efficient filter design, the lowpass filter can be described in a polyphase representation. To do so, k' is further expressed as

$$k' = k''M + l_k \tag{5.16}$$

where $l_k = k' \bmod M$ and

$$g_{l_k}(n) = g((nM + l_k)T_3), l_k = 0, 1, \cdots, M-1 \tag{5.17}$$

$g_{l_k}(n)$ is the polyphase filter representation including M different filters. The two filters that are necessary for each output sample calculation run in parallel and are selected by the value l_k. Based on these considerations, Equation 5.14 can be expressed as

$$x(kT_2) \approx \sum_n x(nT_1) \left[(1-\alpha_k) g_{l_k}(k'' - n) + \alpha_k g_{(l_k+1) modm}(k'' - n + I(l_k = M-1))\right] \tag{5.18}$$

with $I(\cdot)$ as the unit-valued indicator function.

The resulting filter structure is presented in Fig. 5.6. It includes a polyphase filter structure with M filters a 19 filter coefficients where $g_M(n)$ is the time-shifted version of the first one, $g_0(n-1)$. For each output sample, two filters are selected depending on the value of l_k which is computed by the module `Interpolation Control`. Per filter, the incoming samples are multiplied with the filter coefficients before the result is summed up. To compute the result of the linear interpolation, one filter output is multiplied with the interpolation factor α while the other one is multiplied with $1-\alpha$ before the sum of the two results is build. α and $1-\alpha$ are calculated by the `Interpolation Control` as well.

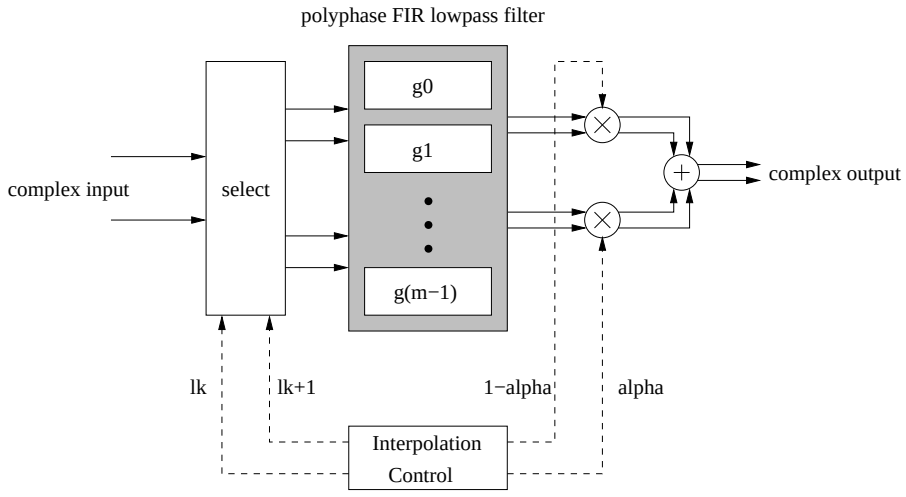


Figure 5.6: Basic SRC Architecture

5.2.2.2 Lowpass Filter Design

Major challenge in filter design is the right choice of the filter coefficients. As the best performance is achieved for filters based on the sinc function, best would be to retrieve the filter coefficients directly from the ideal impulse response $h(t) = sinc(F_s t) = \frac{sin(F_s t)}{F_s t}$. Unfortunately, this impulse response ranges from $-\infty$ to ∞ which is not implementable in reality. To get the impulse response causal, a common approach is the window method whose advantages are robustness and simplicity. Disadvantages on the other side comprise the lack of a precise control of the cutoff frequency which depends on the window size and the filter length.

The basic idea of the window method is to multiply the digitized impulse response $h(n)$ with a window $w(n)$. Cutting of the impulse response may lead to undesired effects like Gibbs phenomenon or leakage effect which can be observed in a high ripple content. Therefore the right choice of the window is very important. In [126] a comparison of different windows like rectangular window, Hanning window, Blackman window oder Kaiser window is provided.

For our design, it turned out through simulation that the best achievable performance was obtained using the Kaiser window which is defined as

$$w_{kaiser}(n) = \begin{cases} \frac{I_0(\beta\sqrt{1-[\frac{n-\alpha}{\alpha})^2]}}{I_0(\beta)} & for \quad 0 \leq n \leq N \\ 0 & otherwise \end{cases}$$

where $\alpha = \frac{N}{2}$. $I_0(\bullet)$ is the zero-order Bessel function of the first kind:

$$I_0(x) = \sum_{k=0}^{\infty} \left[\frac{(\frac{x}{2})^k}{k!} \right]^2 \tag{5.19}$$

β depends on the maximum tolerable approximation error and can be expressed as $\frac{1}{4}$ of $\Delta t * \Delta\omega$. The latter denotes the so-called "time-bandwidth product" of the chosen window in radians. For the computation of β (values are typically between 3 and 9) first

$$A = -20log_{10}\delta \tag{5.20}$$

has to be computed, with δ as the maximum tolerable approximation error of the filter. Then β can be obtained.

$$\beta = \begin{cases} 0.1102(A - 8.7) & \text{for } A > 50 \\ 0.5842(A - 21)^{0.4} + 0.07886(A - 21) & \text{for } 21 \leq A \leq 50 \\ 0.0 & \text{for } A < 21 \end{cases} \tag{5.21}$$

Apart from that, the Kaiser window is sometimes further parameterized by a value α which represents half of the window's time-bandwidth product $\Delta t * \Delta f$ in cycles. It is expressed as $\alpha = \frac{\beta}{\pi}$.

To quantify the trade-off between the main-lobe width and the side-lobe area of the window, the values $N + 1$ and β are important. While the trade-off between side-lobe level and main-lobe width is determined by the latter, decreasing the window length, $N + 1$, results in decreasing the main lobe while the side lob is not affected. N has to be chosen so that

$$N = \frac{A - 8}{2.285\Delta\omega} \tag{5.22}$$

$\Delta\omega$ is the width of the transition region defined as the cutoff frequency of the stopband ω_s minus the cutoff frequency of the passband ω_p:

$$\Delta\omega = \omega_s - \omega_p \tag{5.23}$$

Fig. 5.7 illustrates the resulting Kaiser window for $\beta = 5$.

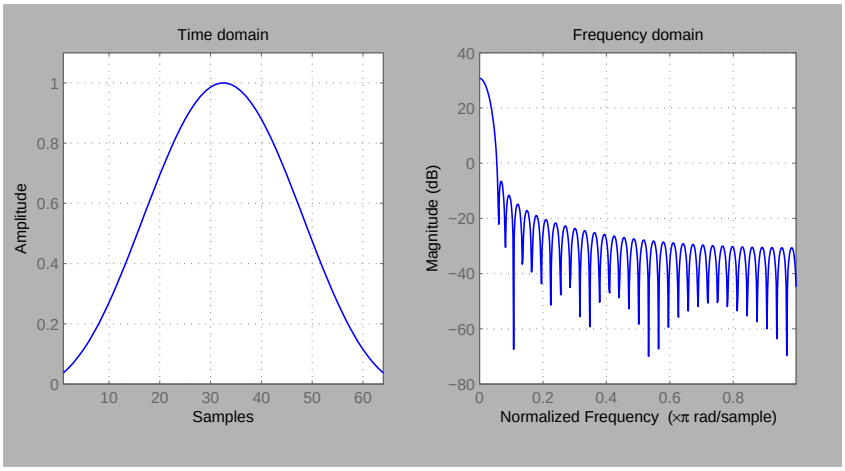


Figure 5.7: Kaiser Window for $\beta = 5$

The filter coefficients retrieved after multiplying the ideal sinc function with the Kaiser window are distributed over the polyphase filter bank as derived in Equation 5.17. Taking the example of a prototype filter with 15 filter coefficients and 3 filters a 5 filter coefficients in the polyphase filter bank, the coefficients are distributed as follows:
- Filter 1: 1, 4, 7, 10, 13
- Filter 2: 2, 5, 8, 11, 14
- Filter 3: 3, 6, 9, 12, 15

All of these filter coefficients have to be stored in the MSS and to be loaded before the filter is started. For our design, the maximum number of filters can dynamically be set to $M = 2$, $M = 4$ or $M = 8$ filters. This quantity may differ between the different channels. The maximum energy per filter has to be equal to one. Thus the value M further denotes the maximum energy of the lowpass filter.
Without any optimization, the resulting filter coefficient memory would have a size of $8 * 19 = 152$ entries a 16 bit. This amount can further be decreased when taking symmetries between the filter coefficients into account. For illustration, an example for $M = 4$ and seven filter coefficients per filter is provided in Table 5.1.

	filter 1	**filter 2**	**filter 3**	**filter 4**
filter coefficient 1	0.0000	0.0689	0.1105	0.0892
filter coefficient 2	0.0000	-0.1201	-0.2018	-0.1739
filter coefficient 3	0.0000	0.2964	0.6331	0.8991
filter coefficient 4	1.0000	0.8991	0.6331	0.2964
filter coefficient 5	0.0000	-0.1739	-0.2018	-0.1201
filter coefficient 6	0.0000	0.0892	0.1105	0.0689
filter coefficient 7	0.0000	-0.0544	-0.0686	-0.0434

Table 5.1: Filter Coefficient Example ($M = 4$, Seven Filter Coefficients per Filter)

There, the following observations can be made:

- Filter 1 comprises only one 1 and 0s otherwise. These two values can be hardcoded in the design and do not need to be stored in the MSS.
- The first three filter coefficients of Filter 3 are equal to the filter coefficients 6 to 4 of Filter 3. Only the last filter coefficient is unique.
- The first six filter coefficients of Filter 2 correspond to the mirrored first six filter coefficients of Filter 4. Only the last filter coefficients are unique.

Extending these observations for the case of $M = 8$ filters, it results that

- The filter coefficients of Filter 1 do not have to be stored in the MSS.
- Filter 2 is the mirrored version of Filter 8 (only the last filter coefficients are unique).
- Filter 3 is the mirrored version of Filter 7 (only the last filter coefficients are unique).
- Filter 4 is the mirrored version of Filter 6 (only the last filter coefficients are unique).
- Filter 5 corresponds to Filter 3 in the example above. Thus only half of its filter coefficients have to be stored.

Taking advantage of these observations, the number of filter coefficients to be stored can be decreased to 107 entries a 16 bit.

Remains the question, why choosing an odd number of filter coefficients is preferred to a filter with an even number of filter coefficients. It was mentioned earlier that in case l_k points to the last filter in the bank, the filter coefficients of the first filter have to be right shifted by one entry. For an even number of filter coefficients as illustrated in Fig. 5.8(a), this shift has to be implemented in hardware which introduces a complexity that could easily be avoided by choosing an odd number of coefficients. There (Fig. 5.8(b)), only the value of two filter coefficients have to be exchanged as the second filter is a modified version of Filter 1 consisting of one 1 and 0s otherwise.

5.2.2.3 Computation of l_k

The choice of the two filters required for the output sample calculation depends on the value l_k. For its computation, first the next known filter coefficient k' has to be determined based on the number of filter coefficients and on the $ratio$ that can be expressed as $\frac{T2}{T1}$. This relation can be obtained when replacing T_3 by $\frac{T1}{M}$ (please refer to Equation 5.12). So k' can be computed as

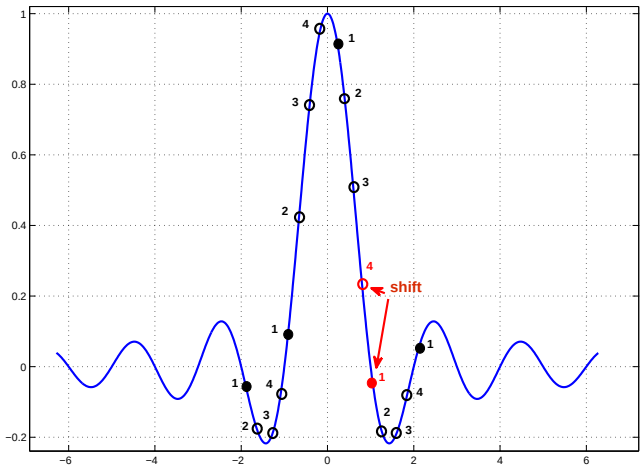
$$k' = \lfloor k * M * ratio \rfloor \tag{5.24}$$

Then the interpolation factor α is expressed as

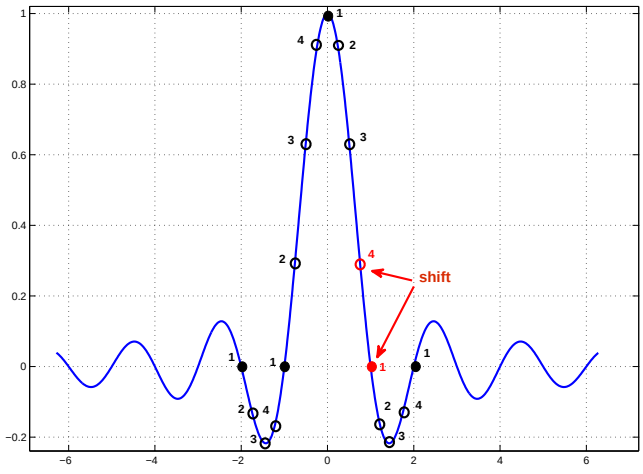
$$\alpha = k * M * ratio - k' \tag{5.25}$$

and l_k can be calculated

$$l_k = k' \mod M \tag{5.26}$$



(a) Even Number of Filter Coefficients



(b) Odd Number of Filter Coefficients

Figure 5.8: Filter Coefficient Distribution

Considering floating-point values, these calculations are quite simple, but for a fixed-point representation, the quantization error leads to wrong values of l_k. For illustration, we imagine the case when the filter has to upsample from 10 MHz to 50 MHz. The ratio between these two sampling rates is 0.2 . In a fixed-point representation with a resolution of 15 bit, 0.2 corresponds to $\lfloor 0.2 * 2^{15} \rfloor = 6553$ while the real result is 6553.5. For the 10th output sample, the value of k' is $k' = 524240$ which corresponds to a floating-point value of 15.9985. Thus the value obtained for l_k is 7 and the last filter in the polyphase filter bank is selected. If the same is calculated with floating-point values, k' obtains a value of 16 and l_k points to the first filter.

This quantization error can be expressed as

$$k * M * (ratio - \lfloor ratio * 2^{15} \rfloor \frac{1}{2^{15}}) \tag{5.27}$$

which is equal to zero in case no error is measured. To guarantee a correct computation of l_k despite quantization, a correction factor has to be introduced. As M is a constant factor, it has to depend on k only. By solving

$$k * M * ratio - k * M * \lfloor ratio * 2^{15} \rfloor * \frac{1}{2^{15}} - corr * k * \frac{1}{2^{15}} = 0 \tag{5.28}$$

where $corr$ is the correction factor we are looking for, $corr$ is obtained as

$$corr = \lceil M(ratio * 2^{15} - \lfloor ratio * 2^{15} \rfloor) \rceil \tag{5.29}$$

The result is rounded up to ensure that the correct filter is selected. This is tolerable, as $k' \mod M$ finally is rounded down to obtain the value l_k. A drawback of this approach is that the introduced error will again lead to wrong results, but not before thousands of output samples have been computed. To avoid this error, we take advantage of an observed periodicity of k: For each ratio a maximum value of k denoted as k_{wrap} can be identified after which the values of l_k and α repeat.

$$k_{wrap} * ratio = \text{integerValue} \tag{5.30}$$

We observed that resetting k once the maximum value is reached avoids wrong filter selections based on the error introduced with the correction factor $corr$.

Please note: The values of M, k_{wrap} and $corr$ have to be precomputed and provided to the SRC via the parameter memory that is embedded in the MSS.

5.2.2.4 Implementation of a Notion of Time

When deriving the filter structure of the SRC, we have distinguished between three different time domains:

- the time difference between the two input samples: T_1
- the time difference between the two output samples: T_2
- the time difference between the two filter coefficients: T_3

To handle these relations, T_1 and T_2 do not need to correspond to the real timing values. Instead it is sufficient if they just express the relation between the sampling frequencies. E.g. when upsampling from 10 MHz to 50 MHz, it is sufficient if T_1 is set to five and T_2 is set to one to represent the ratio of 0.2 between them. This decision is based on the fact that the required resolution for the frequency range of 3 MHz $\leq f_{samp} \leq 61,44$ MHz is 1 Hz which corresponds to

$$|\frac{1}{61.44\text{MHz}} - \frac{1}{61.44\text{MHz} - 1\text{Hz}}| = 2.649 * 10^{-16}s \tag{5.31}$$

To be sure, the usage of atto steps (10^{-18}) is recommended, resulting in a required resolution of 60 bit. Computations based on this resolution are very time and space consuming and may result in a performance drop of the SRC.

In the next subsections, the underlying algorithms required for the decision when to compute a new output sample based on the values of T_1 and T_2 are provided.

5.2.2.5 Upsampling

Upsampling implies that the sampling rate of an incoming signal is increased. Thus the number of generated output samples is higher than the one of the input samples as illustrated in Figure 5.9.

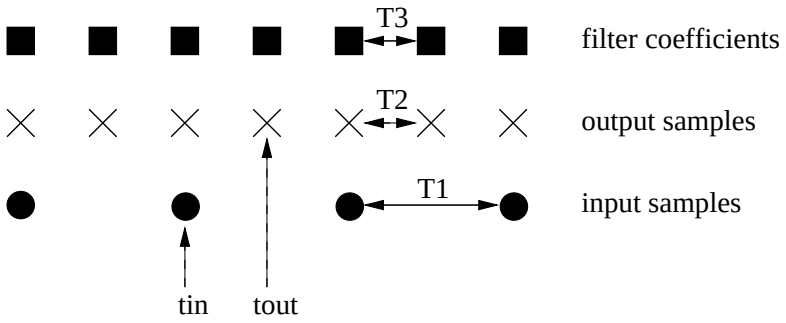


Figure 5.9: Upsampling: Relation between Input and Output Samples

As stated in the algorithm in Fig. 5.10, an output sample is generated in each iteration. In case the value of the input counter `tin` is greater or equal than the value of the output counter `tout` a new input sample has to be shifted in (provided to) the lowpass filter. The counter values are only reset in case they are equal.

For a ratio of 0.4 we set $T_1 = 10$, $T_2 = 4$ and initialize `tin` and `tout` with the values of T_1 and T_2 respectively. Then we start the execution of the algorithm:
Iteration 1) `tout` = 4, increment `tout` by T_2
Iteration 2) `tout` = 8, increment `tout` by T_2
Iteration 3) `tout` = 12, `tout` > `tin`, increment both counter values
Iteration 4) `tout` = 16, increment `tout` by T_2
Iteration 5) `tout` = 20, `tout` = `tin`, reset both counters

5.2.2.6 Downsampling

Downsampling implies that the sampling rate of an incoming signal is decreased. Thus the number of generated output samples is lower than the one of the input samples as illustrated in Figure 5.11.

```
tin = T1
tout = T2

loop{
    calculate

    if tout < tin then
        tout += T2

    else
        shift

        if tout = tin then
            tin = T1
            tout = T2
        else
            tin += T1
            tout += T2
        end if

    end if
}
```

Figure 5.10: SRC Upsampling Algorithm

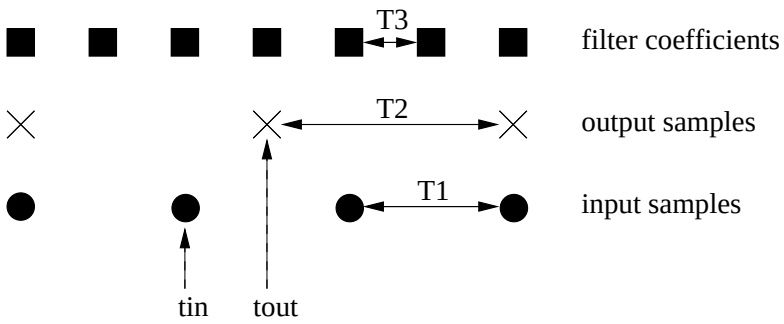


Figure 5.11: Downsampling: Relation between Input and Output Samples

As stated in the algorithm in Fig. 5.12, an input sample is provided to the lowpass filter in each iteration. In case the value of the input counter `tin` is greater or equal than the value of the output counter `tout` a new output sample is generated. The counter values are only reset in case they are equal.

```
tin = T1
tout = T2

loop{
    shift

    if tin < tout then
        tin += T1

    else
        calculate

        if tout = tin then
            tin = T1
            tout = T2

        else
            tin += T1
            tout += T2
        end if

    end if
}
```

Figure 5.12: SRC Downsampling Algorithm

5.3 System Integration

5.3.1 Preprocessor Prototype

To complete the IEEE 802.11p receiver chain, a prototype of the Preprocessor has been implemented. This version is not fully optimized yet, but already includes some of the major functionalities of this DSP engine which are

- the connection to the ADA interface
- the design of the MSS including the FIFOs
- a first design of the processing unit embedding the SRC and a first version of the main control
- a small wrapper to connect the SRC to the ADA interface
- scheduling of the active channels in a Round Robin fashion

- parameter update at runtime
- RX interrupt generation
- channel switch state machine that signals to the SRC when a channel switch has to be performed
- provision of the input samples to the SRC
- storage of the SRC output samples in the MSS (RX) and their provision to the ADA interface (TX)

Still missing are the suspended mode and the implementation of I/Q, NCO and PD. The finalization of the Preprocessor is thus part of our future work.

5.3.2 C-Models of the SRC

Before the SRC was described in VHDL, two different C-Models were implemented for proof of concept and to define the dynamics of the SRC parameters. These parameters were the sampling rates and their ratio, the number of filters embedded in the polyphase filter bank and the number of required bits for the fixed-point representation of the parameters. The only difference between the two models is the value representation. While one is implemented using a fixed-point representation, the other one is based on a floating-point one. This allows to get clear numbers about the quantization error and about the Signal to Noise Ratio (SNR) necessary to evaluate the SRC performance. The SNR can be expressed as

$$SNR = 10 \log_{10} \frac{\frac{1}{n} \sum_1^n x_{ideal}(n)^2}{\frac{1}{n} \sum_1^n (x(n) - x_{ideal}(n))^2} \tag{5.32}$$

with $x(n)$ as the obtained and $x_{ideal}(n)$ as the ideal result. In addition the fixed-point model enables a fast validation and verification of the final VHDL design.

5.3.3 VHDL Model

The top level view of the SRC is illustrated in Fig. 5.13.
Although the input samples are complex ones where real and imaginary part have a size of 16 bit, the processing is based on real values where real and imaginary part processes are executed in parallel. The filter coefficients and the values computed by the Interpolation Control (IPC) are the same for both execution chains. In `Load Coefficients`, the filter coefficients are loaded from the MSS and stored in local registers. The connection between these registers and the FIR filters is established via multiplexers that are triggered by the value l_k. Despite the fact that only a subset of the filter coefficients is stored in the MSS, all 8 x 19 filter coefficients have to be provided to the two FIR filters when required. Table 5.2 illustrates how the real filter coefficients are generated from the filter coefficients stored in the MSS. 0/1* denotes the shifted version of the second filter value in case l_k points to the last filter in the polyphase filter bank and CCx* denotes the mirrored version of CCx where only the last filter coefficients are unique.
l_k, α and $1 - \alpha$ are provided by the module `IPC`. To save one multiplier, $M * ratio$ is a parameter given to the SRC. The other parameters are k_{wrap} and the correction factor $corr$. The architecture of this module is shown in Fig. 5.14.

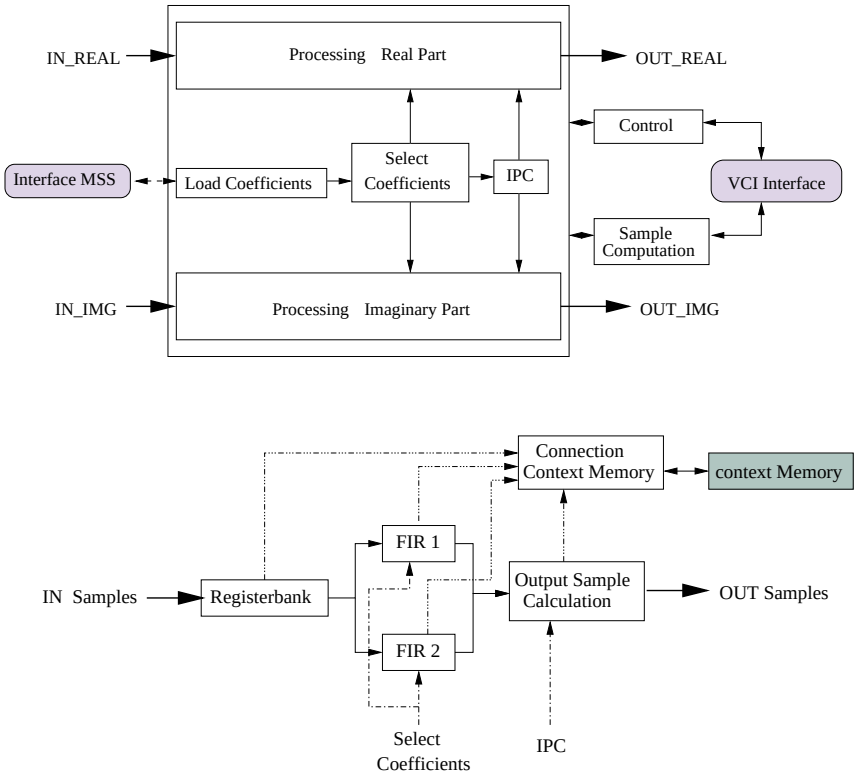


Figure 5.13: SRC Top Level View

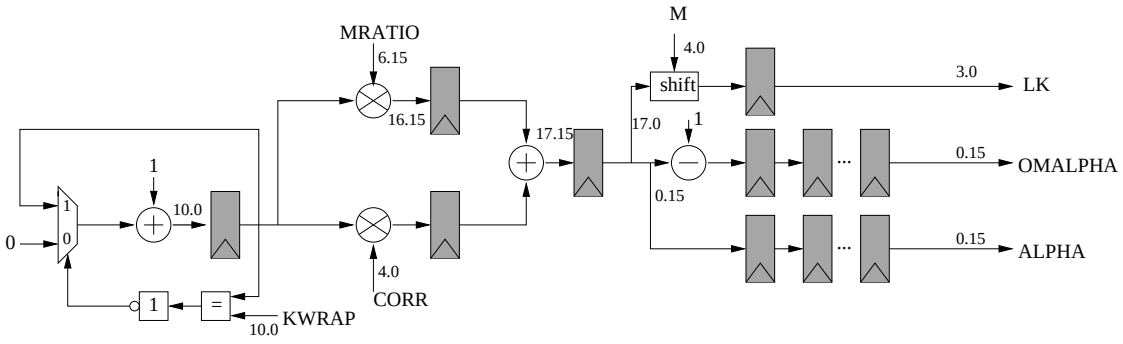


Figure 5.14: Module InterpolationControl

	$M = 2$	$M = 4$	$M = 8$
$l_k = 0$	CI = 0/1	CI = 0/1	CI = 0/1
	CII = CM	CII = CC1	CII = CC1
$l_k = 1$	CI = CM	CI = CC1	CI = CC1
	CII = 0/1*	CII = CM	CII = CC2
$l_k = 2$		CI = CM	CI = CC2
		CII = CC1*	CII = CC3
$l_k = 3$		CI = CC1*	CI = CC3
		CII = 0/1*	CII = CM
$l_k = 4$			CI = CM
			CII = CC3*
$l_k = 5$			CI = CC3*
			CII = CC2*
$l_k = 6$			CI = CC2*
			CII = CC1*
$l_k = 7$			CI = CC1*
			CII = 0/1*

Table 5.2: Generation of the Missing Filter Coefficients

α and $1 - \alpha$ are delayed by six cycles as l_k is needed before the FIR filter processing and the two interpolation factors afterwards for the calculation of the output samples.

The modulo operation in the design is realized by forwarding

- the LSB of l_k in case $M = 2$
- the two LSB of l_k in case $M = 4$
- the three LSB of l_k in case $M = 8$

The FIR filter is split over the modules `Registerbank` and `FIR`. For illustration, Fig. 5.15 provides a simple architecture for an FIR filter with four filter coefficients.
The input samples are first shifted via registers and then multiplied with the filter coefficients c0 to c3. For the nth filter coefficient, the samples are delayed by $n - 1$ cycles. Finally, the multiplication results are summed up.
In case of our prototype, the registerbank consists of 8 x 19 registers a 16 bit. The output of the registerbank is the same for both filters in the polyphase filter bank. At a first glance, such an architecture does not seem to be very critical. But one has to be aware of the fact that each filter in the polyphase filterbank consists of 19 multipliers and 18 adders that are distributed over a 5 stage adder pipeline. An alternative could be a sequential FIR filter solution, but this would result in 18 cycles necessary to build the sum of the multiplier outputs and would thus result in a significant performance drop. Another issue are the invoked DSP48E slices when mapping the filter on the FPGA target. Per filter, 19 of these elements are required, resulting in a total number of 76 DSP48E slices out of the available 172 ones.

For the generation of each output sample, the two invoked FIR filters are executed in parallel. In the module `Output Sample Calculation`, the two filter results are finally multiplied with α and $1 - \alpha$ before their sum is build to generate the output samples that are either provided to the

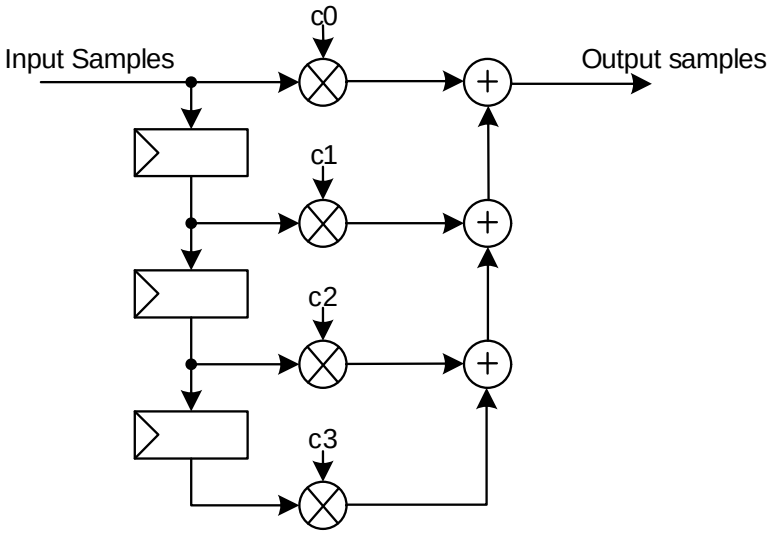


Figure 5.15: Example FIR Filter with 4 Filter Coefficients

ADA interface (TX) or that are stored in the MSS (RX).

The decision about when to shift a new sample in the `Registerbank` and when to compute a new output sample are made by the module `Sample Computation` where the difference between the two different sampling rates is handled. In Figure 5.16 the signal settings for the upsampling mode are illustrated. In case a new sample has to be loaded from the MSS, the filter execution continues once it is available.

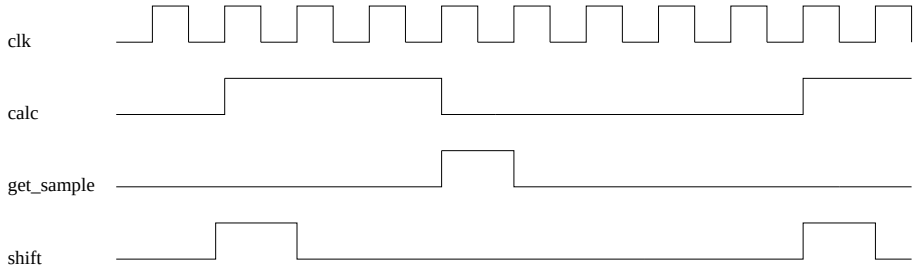


Figure 5.16: Example: Upsampling by a factor of 3

Figure 5.17 illustrates the downsampling process by a factor of two where T_1 is set to 2 and T_2 is set to 1. The whole process is triggered to `nco_req`, allowing to read one sample per cycle as long as samples are available. Each new sample is then shifted into the `Registerbank`. To do so `shift` is set to one. Depending on the internal counter values, each second value is computed by the SRC (`calc` is set to one each second cycle).
To realize an instantaneous channel switch, all registers in the design are duplicated. In Fig. 5.18 an example for the processing of four different channels is provided. Whether these channels are RX or TX ones is not important as the scheduling does not depend on the execution mode. The four channels are invoked in a Round Robin fashion. While, for instance, CH2 is executed, the values of CH1 are stored in the context memory, and the values of CH3 are loaded.

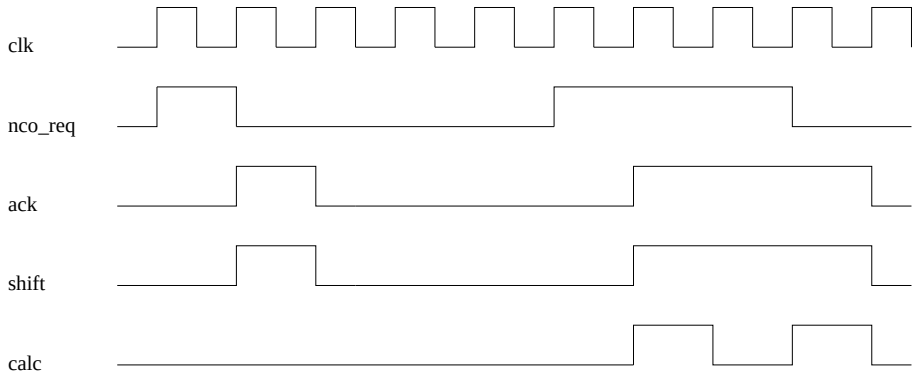


Figure 5.17: Example: Downsampling by a factor of 2.5

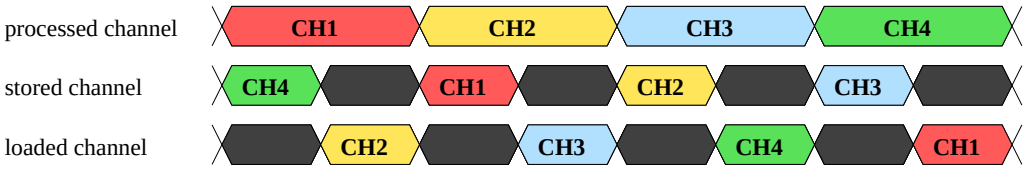


Figure 5.18: Channel Scheduling

The first time a channel is executed, the modules of the SRC have to be synchronized, including the load procedure of the first filter coefficients. For this purpose, a parameter in the CSS control registers can be set.

The control of the SRC is split over two different state machines that are executed in parallel. One is responsible for the handling of the context switch, the other one takes care of the normal SRC processing, including the synchronization of the different modules once a channel is executed for the first time.

5.4 Performance Analysis

5.4.1 C-Model Performance Results

The maximum achievable SNR value is determined by the ADA converter performance. After [140], it can be estimated as

$$\text{SNR} = (6.02 * r + 1.76)\text{dB} \tag{5.33}$$

with r as the signal resolution in bits. In RX / TX r is equal to 12 / 14 bit which results in a maximum SNR of 74 dB / 86.04 dB.

Taking the C-models, the SNR values of different known input signals like sinusoidals or sweeps are computed automatically when executing the code. To do so, the incoming floating-point signal is A/D converted, quantized to 16 bit, resampled and D/A converted to obtain a floating-point representation. Then the ideal result without ADA conversion and finally the SNR are computed.

In case no interpolation is required, the SRC easily obtains the maximum SNR values for sinusoidal and sweep test signal. When interpolation becomes necessary, the SNR depends on

the ratio between the sampling frequencies, on the oversampling factor and on the input signal type. Table 5.3 lists some results obtained for an exemplary sinusoidal signal defined as $y(x) = \frac{1}{4}sin(x') + sin(\frac{x'}{3}) + sin(\frac{x'}{2}) + cos(x')$ with $x' = 2\pi fx$

mode	**ratio**	**SNR**
upsampling	1.45	80.12 dB
upsampling	2	86 dB
downsampling	4.3	73.8 dB
downsampling	5	74 dB

Table 5.3: SRC Results for a Sinusoidal Test Signal

In case white Gaussian noise test signals are used, the SNR can be obtained by evaluating the Power Spectral Density (PDS) which describes how the signal power in time domain is distributed over the frequencies. Before starting the SRC, the test signal is lowpass filtered and oversampled. Upsampling by a factor of 2.5 results in an SNR of around 82 dB while the SNR is very close to the maximum possible one in case no interpolation is needed.

Different results are obtained when modifying the adjustable filter parameters or when changing the resolution of internal signals in the architecture. For a simple sinusoidal signal defined as $y(x) = sin(2 * \pi * 1000000 * x)$ we observed that

- The resolution of the filter coefficients can be at maximum 25 bit. Otherwise, the number of DSP48E slices increases which is not acceptable as the filter already uses almost 43 % of the available ones.
- In case of realizing a filter structure with $M = 16$ or $M = 32$, the higher resource consumption does not justify the gained performance. Not only that more memory space is required for the filter coefficient storage, also the time to load them in the SRC increases significantly (Fig 5.19).
- Changing the β parameter of the Kaiser window also leads to different SNR results. This parameter controls the width of the main lobe of the filter and provides information about its 3 dB cutoff frequency. Therefore the obtained results (Fig 5.20) do not only depend on the test signal but also on the filter characteristics.

Finally we compare the two different C-models to get an idea about the quantization noise. For the sinusoidal test signal, results are provided in Table 5.4.

ratio	**floating-point SNR**	**fixed-point SNR**
0.2	68.120699	67.061089
0.5	82.138105	68.991791
0.69	68.130715	67.037849
1.7	68.147434	65.075775
1.875	94.336565	67.939613
2.0	inf	76.64704

Table 5.4: SRC Results for a Sinusoidal Test Signal for Quantization Noise Measurements

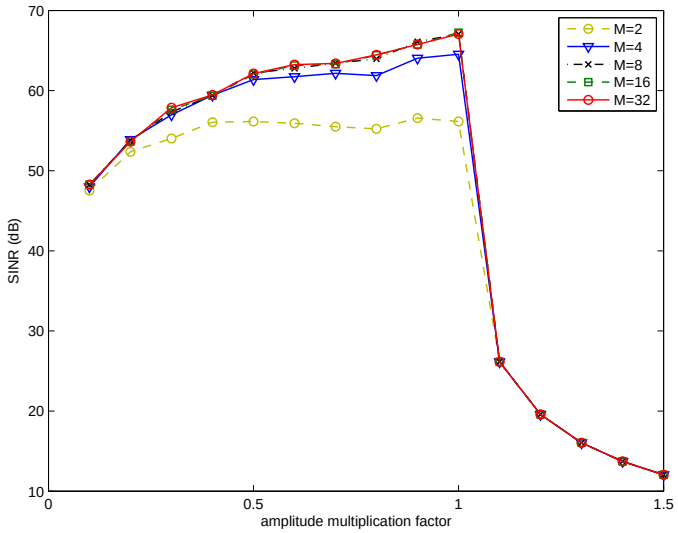


Figure 5.19: SNR Performance for Changing M

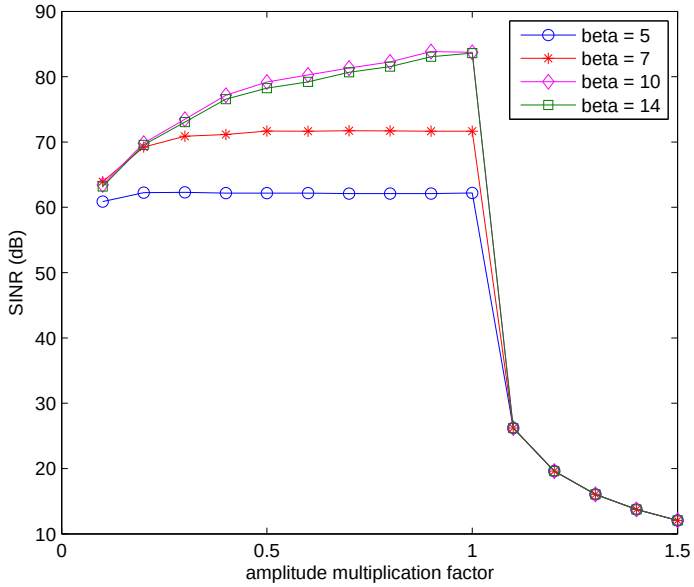


Figure 5.20: SNR Performance for Changing Beta

5.4.2 Synthesis Results

When synthesizing the SRC for the baseband FPGA target, a maximum frequency of 130.005 MHz is obtained after place and route. Required resources are
- 32899 function generators (15.87 %)
- 8225 CLB slices (15.87 %)
- 20013 DFFs or latches (9.54 %)
- 30 Block RAMs (10.42 %)
- 82 DSP48E slices (42.71 %)

For the first Preprocessor prototype, a maximum frequency of 98.261 MHz has been obtained after place and route. Required resources in this case are
- 41007 function generators (19.78 %)
- 10252 CLB slices (19.78 %)
- 26206 DFFs or latches (12.49 %)
- 55 Block RAMs (19.10 %)
- 82 DSP48E slices (42.71 %)

5.5 Conclusions

In this chapter, we focused on the Preprocessor DSP engine to complete the design of the IEEE 802.11p receiver. Most critical part in terms of performance and space consumption is the SRC embedded in the processing unit of the Preprocessor. The presented design is a flexible high performance filter based on bandlimited interpolation. It supports fractional ratios between the fixed sampling rate of the ADA converters and the sampling rates of today's wireless communication standards with a resolution of 1 Hz. Up to four channels in both directions, RX and TX, are supported that may possess a different set of parameters - resulting in dynamic system changes at runtime. To guarantee an continuous processing of the SRC, channel switches happen instantaneously.
To complete the IEEE 802.11p receiver chain we further presented a first prototype of the Preprocessor that includes already the main features of this DSP engine.

The performance evaluation has shown that the SRC performance mainly depends on two different factors: (1) the sampling rate of the wireless communication standards in process and (2) the ratio between the sampling rates. It is obvious that the less interpolation is required and the lower the ratio, the higher the measured performance is. Possible design extensions to obtain an excellent performance even for high upsampling ratios could be

- splitting the ratio in two parts: an integer and a fractional one. Integer resampling could be performed by CIC filters while our SRC solution could be used for the fractional part.
- processing the SRC several times. This approach is not suitable as a continuous processing of the SRC would no longer be guaranteed in this case.

Besides, the performance of the current design could be increased by the implementation of an addressable registerbank or by the realization of higher order filters. The latter comes with the drawback of a longer filter initialization time and more memory required for the filter coefficient storage. Apart from that it is also imaginable to increase the Preprocessor frequency by adding

more registers between the MSS control part and the actual RAMs, as it is already done for the other DSP engines on the ExpressMIMO platform.

Chapter 6

Conclusions and Future Work

The work presented in this thesis is strongly connected to latest trends in the automotive industry that demand for a combination of C2X and TPEG information. Standards of interest are IEEE 802.11p and ETSI DAB. To combine these two we have chosen the approach of a flexible SDR platform that is not limited to the automotive context but to wireless communication standards in general. The thesis focused on an efficient physical layer design of the IEEE 802.11p receiver for the OpenAirInterface ExpressMIMO platform by following the basic development methodology described in Chapter 2.3. This included (1) the development of purely functional models using the emulation library of the platform, (2) the cycle accurate HW / SW co-simulation via Modelsim and (3) the receiver validation on the hardware platform. One missing component was the Preprocessor where a first solution including a fractional SRC has been provided. Based on the obtained results we identified design bottlenecks and presented possible solutions to overcome these drawbacks. Apart from that we had a look at a multimodal processing of the two standards of interest, IEEE 802.11p and ETSI DAB.

In the introduction, all these objectives have been expressed in a list of different tasks that had to be accomplished throughout this thesis:

1. Emulation of the IEEE 802.11p receiver with the help of the Library for ExpressMIMO baseband called *libembb*

2. Implementation of the IEEE 802.11p receiver and its performance evaluation on the ExpressMIMO platform

3. Focusing on the question how DAB and IEEE 802.11p can be executed simultaneously on the ExpressMIMO platform

4. Identification of design bottlenecks and the provision of possible solutions

5. Implementation of a Preprocessor DSP engine prototype to complete the IEEE 802.11p receiver chain

In the following we recall these tasks to summarize the achieved contributions as well as design limitations and possible future enhancements.

6.1 Receiver Emulation

Thanks to the emulation library of the ExpressMIMO platform, libembb, easy receiver validation and verification in a pure software environment became possible. The functions included in this bit-accurate C++ library represent all functions of the real hardware platform, including basic commands for the main CPU and the local microcontrollers. The design of the emulation prototype of the IEEE 802.11p receiver has been implemented with a sequential execution in mind where no concurrency of the DSPs was exploited. Advantages of this approach were (1) the fast improvement of algorithms and (2) the creation of first performance figures based on the pure processing time of the DSPs. The latter has been facilitated by an annotation with cycle counters and by an automatic generation of trace files that can be interpreted by software programs like Matlab or Octave. To enable a simplified integration of standards in case of multimodal processing and to simplify updates due to changes in the baseband, we further extended our design by an additional layer, called `expressmimo_emu`.
The presented receiver code supports all modulation schemes and code rates of the standard and could theoretically be executed on the hardware platform itself without any modifications. Unfortunately, this approach still results in a significant performance drop for standards operating on short data sets, so that a redesign of the receiver code became necessary.

Limitations:

- Currently, no parallelism of the DSP engines is exploited when executing the receiver code on the hardware platform. Therefore the code has been designed with a sequential execution in mind and has not been optimized for its execution on the ExpressMIMO platform.
- The emulation model can only serve as an input for a runtime analysis based on the pure processing times of the DSPs and the data transfer times of the DMAs. Not supported is an estimation of the resulting communication overhead.

Possible future enhancements:

- In the future, libembb will be extended to exploit hardware parallelism on the platform. Once this is finished, it would be an interesting task to redesign the current emulation receiver prototype and to carry out a detailed performance analysis on the ExpressMIMO platform. This would give an idea about the efficiency of the new libembb design, especially when executing standards with short data sets.
- The obtained performance figures could be compared to the optimized receiver prototype presented in Chapter 3 to identify possible design bottlenecks in either of the two design approaches.

6.2 Receiver Implementation for Prototyping

In the future design flow, the code written for emulation can directly be compiled for the hardware platform even for standards with short data sets. We have shown, that this approach is currently too time consuming for this kind of standards as the programming of the DSP engines at runtime is still too time consuming. Therefore, code optimization by hand was unavoidable. The identified design bottlenecks and the provided solutions to improve the receiver performance are further detailed in Section 6.4.
Besides, the performance has further been improved by taking advantage of the CSS command

preparation and by implementing a simple scheduler being able execute different DSP engines in parallel. Throughout different performance evaluations in Modelsim and on the real hardware platform we observed, that the IEEE 802.11p receiver can be processed in real-time for BPSK, QPSK and 16-QAM. For a slightly higher target frequency of the whole baseband engine (e.g. ASIC target), this is also the case the 64-QAM. Apart from that we recognized that a further reduction of the communication overhead can only be achieved by a distributed control flow based on the local CSS UCs or by a microprocessor or sequencer on the baseband side.

Limitations:

- Executing the hardware version of libembb on the ExpressMIMO platform results in a significant performance drop, due to the huge communication overhead that occurs when programming the DSP engines. Currently a manual code optimization is still unavoidable when executing standards operating on short data sets.

Possible future enhancements:

- A possible enhancement could be the implementation of an efficient API for standards with short data sets which already includes the design optimization identified in Section 6.4.
- Currently, scheduling the different DSP engines is based on a Round Robin policy. It could be interesting to experiment with different scheduling policies to analyze the resulting performance changes.
- Another improvement would be the realization of a distributed control flow on the platform and the comparison to the presented design. This could be either realized by the local UC inside the standardized DSP shells or by a sequencer or microprocessor on the baseband side. In case of the FEP, more complex algorithms like channel estimation or data detection could be delegated to this DSP while the main scheduling is still in the responsibility of LEON3.
- Furthermore, performance figures based on the power consumption have to be obtained for the whole receiver chain.

6.3 Multimodal Standard Execution

The execution of IEEE 802.11p and ETSI DAB was still an open research topic at the beginning of this thesis. Therefore we were interested in obtaining first performance figures to design an efficient scheduler for the ExpressMIMO platform. Due to major differences between these two standards this task became very challenging. IEEE 802.11p is a packet based standard whose packet interarrival time is not known in advance. Operating on short vectors, this standard further requires a very fast baseband engine. DAB, instead, is a frame based standard whose future execution is known once the beginning of a frame is detected. In contrast to IEEE 802.11p it operates on larger vectors so that the communication overhead caused when programming the DSPs can almost be neglected.
To get first key figures for the scheduler design, a detailed runtime performance analysis based on the emulation prototypes of both receivers has been carried out. Throughout this analysis it turned out, that most of the tasks were running on the FEP. Although the FEP is not the computationally most intensive DSP engine, it has to execute most of the tasks, including the latency critical ones. Based on these results, scheduling guidelines have been derived and a simple first scheduler has

been presented.

Limitations:

- Throughout the runtime performance analysis, the FEP turned out to be a bottleneck of the design. A possible solution to overcome this drawback is recalled in Section 6.4.

Possible future enhancements:

- The scheduling algorithm has to be finalized and tested on the ExpressMIMO platform together with the IEEE 802.11p and the DAB receiver.
- Improvement of the DAB code by applying the same hardware optimizations than for the IEEE 802.11p receiver code.

6.4 Identification of Design Bottlenecks

6.4.1 Receiver Optimizations

When testing the receiver code on the real hardware platform using FPGAs, different design bottlenecks have been identified and solved. First the C code running on the main CPU has been optimized to decrease the overhead due to function calls after compilation. Modifications comprised a higher number of inline functions and macros as well as a limited number of parameters to be set dynamically at runtime. Apart from that we observed, that the interrupt handler provided by MutekH still decreases the IEEE 802.11p receiver performance significantly so that we had to poll the status registers instead. Another identified optimization in the design flow was grouping the OFDM symbols included in the IEEE 802.11p DATA field. We have shown, that the communication overhead can be decreased as this approach enables the FEP to operate on larger vectors. Besides, programming the DSP engines at runtime turned out to be very time consuming and resulted in a huge performance drop. To overcome this drawback, we have proposed an efficient alternative by preparing the commands in advance and to store them in a local memory before starting the receiver.

Limitations:

- Although the interrupt handler provided by MutekH is very efficient when compared to others, the introduced communication overhead results in a huge performance drop when executing the IEEE 802.11p standard.
- Programming the DSP engines at runtime may be very time consuming and may result in a lower performance. Command preparation as an alternative is only suitable in case the commands are almost static and require only few modifications in case of dynamic parameter changes at runtime.

Possible future enhancements:

- One possible enhancement could be the implementation of a faster interrupt handler or the provision of libembb functions based on the polling of the status registers.
- The way how the DSP engines are currently programmed lowers the performance for standards operating on short data sets. It is therefore strongly recommended to reimplement the invoked libembb interface to make it suitable for all kinds of standards.

6.4.2 ASIP Design for Front-End Processing

When executing operations on short vectors on the FEP synthesized for the FPGA target, the programming communication overhead is huge when compared to the pure processing time of this DSP. To overcome this limitation, the FEP vector processing unit has been replaced by an ASIP solution, called A-FEP. For development, the LISA language that has gained commercial acceptance over the past years has been chosen. In contrast to the programmable FEP DSP engine (C-FEP), the A-FEP version embeds general purpose instructions and comes with reduced internal latencies. We have shown, that this makes the A-FEP the appropriate solution when processing standards with short data sets while the performance is more or less equal for standards like DAB. We therefore recommended to include the A-FEP solution as an additional block in the baseband engine. The main advantage would be that the A-FEP could execute latency critical tasks while DFT / IDFT and latency non-critical tasks can be processed on the C-FEP in parallel.
Apart from that, the execution time of the A-FEP has further been compared to two different solutions from academia. For a packet detection algorithm, the achieved performance was similar to an ASPE ASIP presented by ETH Zürich but was still worse than a specialized ASIP solution for synchronization and acquisition.

Limitations:

- For the packet detection algorithm the performance of the A-FEP was worse than the one of a specialized ASIP solution. Although this phenomenon is widely known, it would be interesting to analyze if the A-FEP performance can still be improved.

Possible future enhancements:

- Up to now, the A-FEP has only been validated in Modelsim and by the Synopsis development tools. Still missing is the integration and validation on the ExpressMIMO platform.
- Furthermore, performance figures based on the power consumption have to be obtained for the A-FEP.

6.5 Implementation of a Preprocessor Prototype

This tasks comprised the finalization of the IEEE 802.11p receiver chain by the implementation of the Preprocessor. The Preprocessor connects the external A/D and D/A converters with the remaining baseband processing engine and embeds among others an SRC, an NCO and an I/Q imbalance module. We mainly focused on the implementation of the SRC, which is most critical in terms of performance and space consumption. The presented design is a flexible high performance filter based on the bandlimited interpolation algorithm supporting a fractional ratio between the sampling rates. Up to four channels in both directions, RX and TX, are supported and channel switches happen instantaneously. For the proof of concept and to finalize the IEEE 802.11p receiver chain, a first Preprocessor prototype has been implemented.

Limitations:

- Due to the fixed number of filter coefficients, the performance goes down for high upsampling ratios where a lot of samples have to be obtained via linear interpolation.
- The presented Preprocessor design is a first version and does not yet fully exhibit the whole required functionality.

Possible future enhancements:

- To obtain a high performance even for high upsampling ratios, the SRC could be combined with CIC filters. While the latter could perform resampling by a high integer factor, the fractional upsampling can still be performed by the presented SRC solution.

- The presented Preprocessor architecture has to be optimized and the missing functionality has to be included before the final version of this DSP can be integrated and validated on the ExpressMIMO platform.

6.6 Guidelines for a Future Standard Deployment

To sum up, we can state, that the implementation of standards operating on large vectors can be already performed in a very efficient way on the ExpressMIMO platform when FPGAs are considered as target technology. When executing a vector operation over a size of 4096 samples, for instance, the required processing time would be about 20 μs while the programming time of the DSP stays at a maximum of 360 ns. The resulting communication overhead can thus be neglected. When processing standards with short data sets instead, the code has currently to be manually optimized by command preparation, symbol grouping, polling instead of interrupts, etc. The resulting receiver design is thus more complicated but following these recommendations, a high performance can be achieved even for this type of standard.

Remains the question about possible future projects. It is not a secret that LTE stands for the new generation of wireless communications standards. Compared to HSPA and HSPA+, LTE comes with an improved performance in terms of throughput and latency, as well as with lower costs. This makes this standard the preferable solution for future wireless broadband internet systems. Even in the automotive industry there are already projects that focus on a possible multimodal execution of IEEE 802.11p and LTE. So the question is if LTE can be executed on the ExpressMIMO platform and if yes, may the work presented in this thesis help for the deployment of this standard. To answer this question, let us have a look at the different DSP engines required for the LTE receiver design:

- **Preprocessor:**
  Currently the Preprocessor only supports TDD, but will support FDD in the future version. The required resampling can already be performed by the presented SRC solution.

- **FEP:**
  Air-interfaces of interest are OFDMA and SC-FDMA that are both supported by the FEP. Related air-interface operations are among others primary synchronization to detect the beginning of the frame, secondary synchronization to determine the frame type or channel estimation where the component-wise product is only performed at the pilot positions while all missing values are computed via linear interpolation. The resulting set of FEP operations include radix 2/4 DFT / IDFT (between 128 and 2048) and different vector operations that are all provided by the current version of the FEP.

- **Deinterleaver:**
  The required deinterleaving operations can be performed by the Deinterleaver integrated on the platform.

- **Channel Decoder:**
  LTE requires a 3GPP LTE Turbo decoder with rate 1/3 and a Viterbi decoder with rate 1/3 and with a tail-biting option. Both will be included in the next version of this DSP engine.

So the answer to the question whether the LTE receiver can be implemented on the ExpressMIMO platform is: currently it cannot but soon it can, once the whole ExpressMIMO design is finalized. Remains the question if the work presented in this thesis may help for the deployment of this standard. The answer in this case is yes again. As the IEEE 802.11p receiver was the first prototype developed for the ExpressMIMO platform it paved the way for all future standard deployment by
(1) providing a simple scheduler to execute the DSP engines in parallel.
(2) providing a complex libembb example.
(3) identifying possible design optimizations.
(4) providing performance evaluation frameworks.
(5) proving that the ExpressMIMO platform is functional and capable to meet the real-time constraints of latency critical designs.

Appendix A

Résumé Français

A.1 Introduction

Aujourd'hui, les applications de communication sans fil sont devenus une partie importante de notre vie. Presque tous les jours nous vérifions nos courriels soit sur les smartphones ou les ordinateurs personnels via le réseau local sans fil (Wireless Local Area Network - WLAN). En outre, nous communiquons via nos téléphones mobiles ou nous consultons des systèmes de navigation ou des cartes en ligne en cas nous nous sommes trompés de chemin. En particulier pour la jeune génération, il est impossible d'imaginer vivre dans un monde où ils ne peuvent pas être connectés à leurs amis en tout lieu et à tout moment. Des entreprises des plus en plus ont reconnues cette tendance et cherchent à introduire des nouveaux produits sur le marché. Ces produits intègrent plusieurs d'applications dans un seul appareil, qui est plus petit et plus léger, qui coûte moins cher et qui a un rendement plus élevé que les autres produits qu'on trouve dans les magasins.

Un autre marché intéressant pour les appareils de communication sans fil peut être trouvé dans l'industrie automobile. C'est un fait bien connu que l'évolution démographique conduit à un pourcentage croissant de personnes âgées, en particulier en Europe. Dans des pays comme l'Allemagne, où il n'existe pas de limite d'âge pour la conduite automobile, il y a un grand besoin de nouvelles applications de sécurité comme les mesures de vitesse, des avertissements quand il y a des obstacles sur la route ou la mesure de la distance entre des voitures. Deux termes clés qui sont utilisés souvent dans ce contexte sont Car-to-Car communication (C2C) et Car-to-Infrastructure communication (C2I) qui comportent également la mise à disposition des applications non-sécurité comme des péages, des informations touristiques ou internet mobile. Normes d'intérêt sont IEEE 802.11p et DAB (Digital Audio Broadcasting). La première est une amélioration de la norme IEEE 802.11a qui est utilisée pour les connexions wifi. Pour combiner ces deux normes, deux approches sont envisageables. Soit ils sont mis en œuvre individuellement et viennent avec leurs propres récepteurs et émetteurs qui doivent être intégrés dans la voiture, ou les deux sont combinées en un seul dispositif. Comme c'est le cas pour le marché de la téléphonie mobile, il est important que les appareils sont petites, pas chères et de haute performance. En plus elles doivent être facilement adaptable à des futures normes. Surtout l'intégration d'une nouvelle norme dans une voiture prend beaucoup de temps et coûte beaucoup d'argent (par exemple l'intégration du LTE - Long Term Evolution). Par conséquent, une architecture unique qui est capable de traiter n'importe quelle norme de communication sans fil est la solution préférable.
Faire face à ces exigences croissantes pour les architectures de radio reconfigurables est une tâche très difficile. Une solution peut être trouvée dans le cadre de Software Defined Radio (SDR). Un objectif majeur du SDR est de fournir des solutions des plate-formes flexibles qui support-

ent un large éventail de différentes normes de communication sans fil de manière multimodale. Cette approche ne vient pas seulement avec l'avantage d'un développement plus rapide et d'un déploiement plus rapide des nouvelles normes, mais aussi avec l'adoption automatique dans les environs.

Notre intérêt particulier est l'analyse des normes de latence critiques sur la plate-forme Express-MIMO au moyen de la norme IEEE 802.11p. En plus nous avons enquêté la combinaison d'un récepteur IEEE 802.11p avec un récepteur DAB. Comme la spécification du premier a été publié en Juillet 2010, des implémentations efficaces des récepteurs pour la couche physique sont encore un sujet de recherche ouvert. Et à notre connaissance, à ce jour peu d'efforts ont été consacrés à la description d'une plate-forme SDR qui peut traiter ces deux normes d'intérêt en parallèle.
Comme cible nous avons choisi la plate-forme OpenAirInterface ExpressMIMO qui est développée par Eurecom et Télécom ParisTech. Contrairement aux d'autres plates-formes SDR, les fonctions de traitement en bande de base sont réparties sur plusieurs Digital Signal Processors (DSPs) comme le décodeur canal, l'interleaver ou le Front-End Processor (FEP) qui peuvent être exécutées en parallèle. Cela permet non seulement une meilleure performance de tout le système, mais permet aussi de remplacer facilement un DSP en cas futures mises à jour deviennent nécessaires. La plate-forme est capable de traiter jusqu'à huit canaux différents en même temps (quatre canaux en transmission, quatre en réception) en réutilisant les ressources programmables sur la plate-forme. Défi de conception principale est la synchronisation de ces ressources en fournissant un maximum de précision et en répondant à toutes les exigences en temps réel. La plate-forme peut en outre être émulé avec la library for ExpressMIMO baseband appelé *libembb*, qui permet une validation et une vérification du récepteur dans un environnement purement logicielle.
Au tout début de cette thèse, le travail sur cette plate-forme était toujours en cours. Pour ça, le récepteur présenté est la toute première création complet qui a été élaborée et évaluée sur cette plate-forme cible et qui a été émulé à l'aide de libembb. Elle a donc été servie comme une première preuve de concept de la conception tout entière. Normes d'exécution qui opèrent sur des vecteur de petite taille, comme IEEE 802.11p, besoin d'un traitement très rapide. Donc, choisir cette norme comme un premier cas d'utilisation nous a permis d'évaluer la conception actuelle de la plate-forme pour trouver les goulots d'étranglement et pour trouver des solutions possibles pour les surmonter.
Finalement nous proposons un prototype d'un Préprocesseur. Celui-ci relie le convertisseurs A/D et D/A avec la plate-forme complète et intègre entre autres un convertisseur de fréquence d'échantillonnage (Sample Rate Converter - SRC).

Pour atteindre tous ces contributions, les objectifs essentiels ont été regroupés en cinq tâches différentes:

- Emulation du récepteur l'IEEE 802.11p avec l'aide de libembb
- L'implémentation du récepteur IEEE 802.11p et l'évaluation de la performance sur la plate-forme ExpressMIMO
- L'analyse d'une exécution multimodal des récepteurs DAB et IEEE 802.11p sur la plate-forme ExpressMIMO
- Identification des goulots d'étranglement et la conception des solutions possibles
- L'implémentation d'un prototype Préprocessor

A.2 Intégration du Système

A.2.1 La Plate-Forme OpenAirInterface ExpressMIMO

La plate-forme OpenAirInterface ExpressMIMO ([30], [31]) est développée par Eurecom et Télécom ParisTech. Elle potentiellement prend en charge un large éventail des normes différentes comme GSM, UMTS, WLAN, DAB ou LTE ainsi que leur traitement multimodal. La plate-forme est capable de traiter jusqu'à huit canaux différents en même temps (quatre en réception, quatre en transmission) en réutilisant les mêmes ressources matérielles. Comme chaque canal peut prendre en charge d'un standard de communication sans fil différent, le défi principal de conception est la synchronisation de ces ressources en fournissant un maximum de précision et en répondant à toutes les exigences en temps réel. ExpressMIMO est seulement utilisée pour des analyses expérimentales. Pour ça, la technologie cible qui a été choisie sont des FPGAs. Des avantages contiennent un temps de conception réduit, flexibilité pendant l'exécution, une utilisation simplifiée et des coûts réduits pour les petites quantités par rapport aux autres solutions. Néanmoins ASICs sont pris en compte dans une future version une fois la conception de bande de base a été validée.

Contrairement aux solutions présentées précédemment, la conception actuelle de la plate-forme ExpressMIMO est répartie sur deux FPGAs de Xilinx différentes: (1) un Virtex 5 LX330 pour le traitement de bande de base et (2) un Virtex 5 LX110T pour l'interfaçage et le contrôle (Fig. A.1).

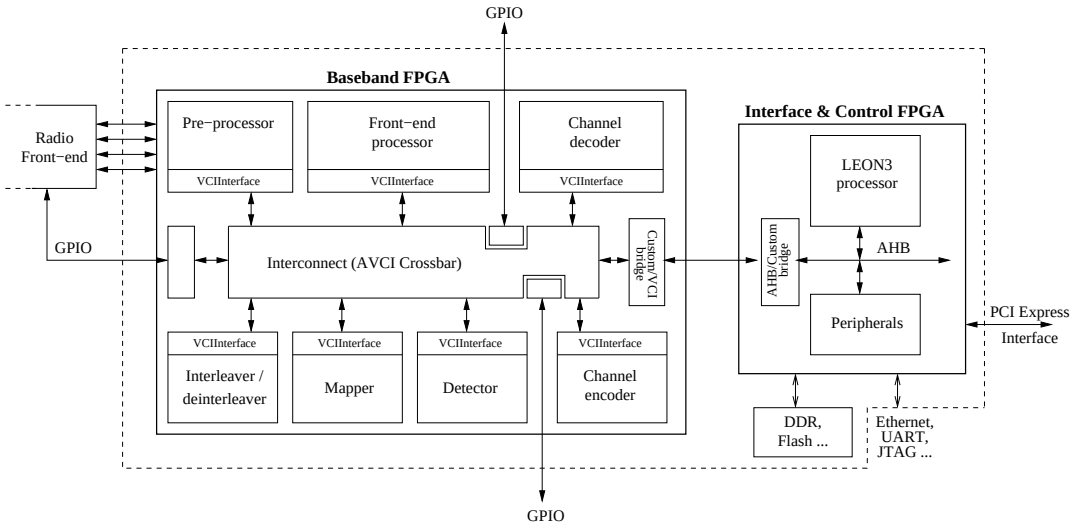


Figure A.1: L'Architecture de la Bande de Base de la Plate-Forme ExpressMIMO

La conception de bande de base est répartie sur plusieurs DSPs indépendants qui sont contrôlées par un processeur SPARC LEON3 de Gaisler Aeroflex. La connexion est établie via une Advanced Virtual Component Interface (AVCI) crossbare. L'architecture des DSPs est basé sur une design standardisée qui est montrée dans Fig. A.2. Cette architecture se compose d'un Control Sub-System (CSS), d'une unité de traitement (Processing Unit - PU) et d'un sous-système mémoire (Memory Sub-System - MSS). L'architecture des deux derniers dépendes de l'utilisation du DSP. Le CSS est commun à tous les blocks et est spécialisée par paramètres. Il contient entre autres un microcontrôleur 8 bits (UC), un DMA, un ensemble de registres de contrôle et d'état ainsi que plusieurs arbitres et FIFOs. En outre, il agit comme une passerelle avec le système hôte

en utilisant deux interfaces de 64 bits qui sont conformes avec la norme AVCI. En outre, des interruption sont utilisées pour la signalisation et la synchronisation avec le système hôte. Pour le moment, l'UC n'a pas encore été intégrée dans le CSS. La version actuelle du récepteur est ainsi orchestrée par un flux de contrôle centralisé où le programme du récepteur entier est exécuté sur le processeur principal. Pour obtenir un premier aperçu sur le comportement fonctionnel d'un émetteur-récepteur sur la plate-forme, une bibliothèque C++ est fournie qui permet d'émuler toutes les fonctions de traitement de base dans un environnement SW.

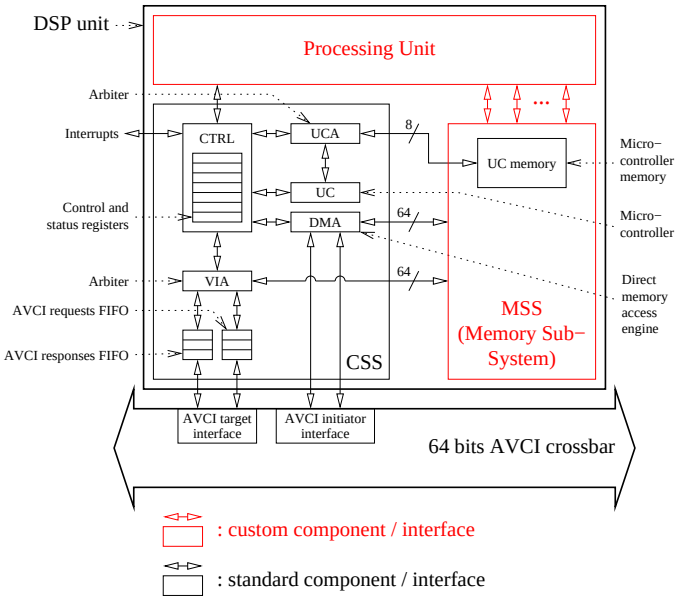


Figure A.2: OpenAirInterface Standardized DSP Shell

A.2.1.1 Méthodologie de Développement

La méthodologie de conception pour chaque design développé pour la plate-forme ExpressMIMO peut être divisée en plusieurs étapes. La première étape est **l'élaboration d'un modèle purement fonctionnelle** qui est le point de départ commun pour tous les modèles d'émetteurs-récepteurs. Les buts de cette étape sont d'analyser la partie algorithmique de l'émetteur, identifier les ressources nécessaires aussi que le flux et les dépendances de données. Ainsi, il est déjà possible d'identifier les goulots d'étranglement quand plusieurs émetteurs et récepteurs sont exécutées dans une manière multimodale sur la plate-forme. Les modèles considérés sont généralement séquentielle et n'exploite pas encore le parallélisme de la plate-forme. Pour la conception d'ExpressMIMO, libembb est utilisé pour la conception du modèle fonctionnel. La deuxième étape est la **HW / SW co-simulation** qui est cycle précis. Cette étape permet d'exploiter pleinement le parallélisme sur la plate-forme. Une approche commune est le HW / SW co-simulation avec l'aide des simulateurs comme par exemple Modelsim. Le parallélisme de la plate-forme comprend le traitement simultané des DSPs, les transferts de données à l'aide des DMA ainsi que la préparation de commandes pour activer les DSPs. Les résultats de cette étape sont les chiffres précis concernant la performances de la conception. La dernière étape est la **validation émetteur-récepteur sur la plate-forme matérielle** où la conception a été testée et validée sur ExpressMIMO soi-même.

A.3 IEEE 802.11p Récepteur pour la plate-forme ExpressMIMO

A.3.1 Motivation

Actuellement, les experts se concentrent sur la conception de C2C et de C2I également connu sous le nom de Vehicle-to-Vehicle et Vehicle-to-Infrastucture communication. Le concept de base de la communication C2C est la suivante: une fois une voiture envoie des messages à d'autres via un canal de communication sans fil, les voitures forment spontanément un réseau ad hoc qui est connu comme Vehicular Ad Hoc Network (VANET). VANETs étendre la vision du conducteur de la route qui peut être limitée en raison de l'obscurité ou des obstacles et prennent en compte le fait que le conducteur peut avoir besoin de temps pour réagir à un événement inattendu. Des cas d'utilisation possibles se concentrent sur la réduction des embouteillages et des accidents, et notamment sur la prévention des collisions, la surveillance des véhicules dangereux, les avertissements d'accidents, etc.
La communication C2X (X = Car, Infrastructure) s'inscrit dans le cadre des futurs systèmes de transport intelligents (Intelligent Transport Systems, ITS). Un excellent aperçu de ITS est donné dans [41]. Dans ce document, non seulement différents scénarios sont présentés, mais aussi les différences d'attribution des fréquences entre plusieurs pays sont renforcées. Les applications possibles dans ITS ne sont pas seulement les applications de sécurité, mais aussi d'éviter les embouteillages, la perception des péages, des informations touristiques, Internet mobile, etc. En général, ils peuvent être divisés dans deux domains: des applications non sécurisées et des applications sécurisée. Pour faire la distinction entre les différentes applications ITS, [42] a proposé un ensemble de critères importants pour la communication C2X qui sont la convivialité, robustesse, coût, l'efficacité, l'évolutivité et l'effort de développement.
En Août 2008, la Commission des Communautés Européennes a décidé que le 5,875 à 5,905 GHz bande est dédié pour les applications liées à la sécurité ITS [43]. La division de cette bande de fréquence est définie par l'European Telecommunications Standards Institute (ETSI) dans [44]. Il est divisé en plusieurs canaux d'une largeur de 10 MHz qui peuvent être combinés pour obtenir des débits de données plus élevés. Une norme d'intérêt principal dans ce contexte est WLAN IEEE 802.11p ([45], [46]) qui est une amélioration de la norme IEEE 802.11a [47]. Contrairement à cette dernière la bande passante de l'IEEE 802.11p a été ramenée de 20 MHz à 10 MHz. Il en résulte des symboles OFDM qui sont plus longs dans le domaine temporel, et donc dans des systémes avec un délai de grande propage pour éviter ISI (Inter Symbol Interference). ISI est d'une importance majeure pour les cas d'utilisation véhiculaires où les canaux sont fortement variant dans le temps. Ainsi, une réception fiable du signal transmis peut toujours être garanti. La norme IEEE 802.11p est également connue sous le nom Wireless Access in Vehicular Environments (WAVE) qui a son origine en 1999 lorsque la Federal Communication Commission a alloué 75 MHz du spectrum de la Dedicated Short Range Communication (DSRC) exclusivement pour la communication C2X. Un bon aperçu de DSRC est fourni dans [48]. Comme la norme a été sous forme de projet jusqu'à Juillet 2010, une efficace conception d'une émetteur-récepteur est encore un sujet de recherche ouvert. Cette tâche est assez difficile par rapport à d'autres normes. Des émetteurs-récepteurs pour la norme IEEE 802.11p venir avec les exigences de latence très fortes et nécessitent donc un moteur de bande de base très rapide.

Un projet très important dans le contexte de ITS est un projet allemand qui s'appelle SimTD [49], où C2X est mis en œuvre sur la couche physique et sur la couche MAC. Dans le cadre de SimTD, des expériences réelles sont effectuées dans la région de Francfort en Allemagne.

A.3.2 La Norme IEEE 802.11p

Avoir un regard sur les différentes normes de communication sans fil, on peut distinguer deux types différents: (1) les normes basées sur des trames (par exemple LTE, DAB) et (2) des normes basées sur des paquets (2) (par exemple WLAN). IEEE 802.11p fait partie de la seconde catégorie. Lors de l'élaboration d'un émetteur-récepteur à base de paquets pour un système multimodal, un inconvénient majeur est que l'heure d'arrivée du prochain paquet n'est pas connue à l'avance. Cela introduit un indéterminisme qui demande d'un scheduler très flexible dans le cas où des normes multiples sont traitées simultanément.

IEEE 802.11p est une norme OFDM ce qui signifie que son signal de données à débit élevé est réparti sur plusieurs signaux indépendants avec des débits de données plus faibles. Les symboles OFDM sont composés de 80 sous-porteuses. Dans le reste de ce document, une sous-porteuse peut également être notée comme un complexe échantillon avec une taille de 32 bits (la partie réelle et la partie imaginaire ont une taille de 16 bits chaqune). Par symbole OFDM, 16 sous-porteuses représentent un intervalle de garde qui sépare deux symboles OFDM voisins pour éviter leur interférence. Ces intervalles de garde sont construites en utilisant une technique de préfixe cyclique qui signifie que l'intervalle de garde est identique à la dernière partie du symbole OFDM. Les 64 autres sous-porteuses pilotes contiennent 4 comb pilotes nécessaires à l'estimation et compensation de canal, 12 transporteurs null et les informations transmises.
La structure de paquet représenté sur Fig. A.3 est similaire à l'une de la norme IEEE 802.11a. Chaque paquet est constitué d'une partie constante et d'une partie variable. Pour un channel spacing de 10 MHz, la partie constante a une durée de 40μs. Elle est composée du préambule et du SIGNAL Field:

- **Short Training Symbol (STS):** Le STS fait partie du préambule et est formé de 10 répétitions de la même séquence avec une taille de 16 échantillons. Il est utilisé pour la synchronisation des paquets.
- **Long Training Symbol (LTS):** Le LTS est constitué d'un intervalle de garde de 32 échantillons et deux symboles OFDM contenant des séquences pilotes qui sont nécessaires pour l'estimation de canal.
- **SIGNAL field:** Le SIGNAL field indique comment décoder le message transmis. Il est BPSK modulé avec un taux de codage de 1/2 et contient tous les paramètres nécessaires pour la détection des champs de données suivants (DATA field).

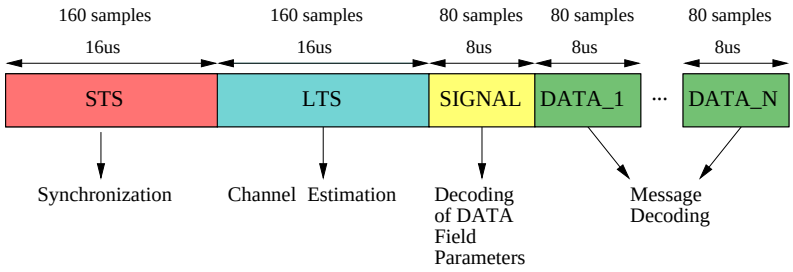


Figure A.3: IEEE 802.11p Paquet (channel spacing 10 MHz)

Contrairement à la partie constante du paquet, le **DATA field** se compose d'un nombre variable de symboles OFDM. Sa taille n'est pas connue avant la procédure de décodage du SIGNAL field est

terminée. Le nombre des symboles OFDM dans le DATA field peut varier de 1 à 1366. Tous les paramètres obtenues du SIGNAL field appliquent sur le champ de données entiére et ne peut pas changer avant le prochain paquet est reçu. Le temps entre la fin d'un paquet et la réception d'une suivante est au moins 10μs. Schémas de modulation possibles pour le DATA field sont BPSK, QPSK, 16-QAM et 64-QAM. Des taux de codage possibles sont 1/2, 2/3 et 3/4.

A.3.3 Développement du Récepteur

Pour l'implementation du récepteur, seulement VCI RAM, FEP, Deinterleaver et le décodeur canal sont utilisés pour décoder les paquets du récepteur IEEE 802.11p (Fig. A.4). Le Préprocesseur sera inclus dans une future version. Après chaque interruption, le Préprocesseur va copier 640 complexes échantillons dans la mémoire circulaire qui est inclue dans le MSS du FEP. Cela correspond à une memcopy de huit symboles OFDM.

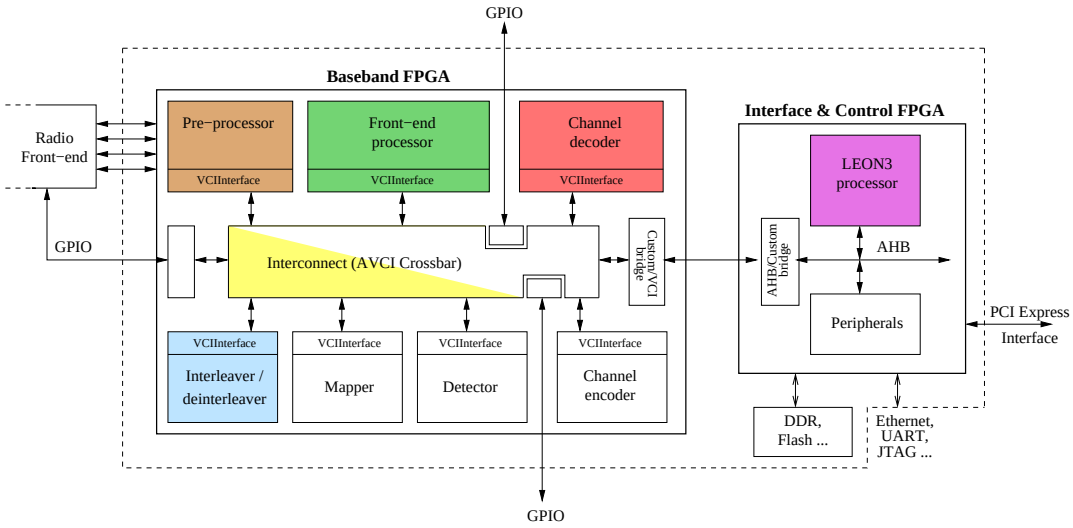


Figure A.4: Architecture de Bande de Base de la Plate-Forme ExpressMIMO

A.3.3.1 Prototype Matlab du Récepteur IEEE 802.11p

Une première version du récepteur IEEE 802.11p a été mis en œuvre dans Matlab pour la validation algorithmique rapide. Pour générer les signaux de test, un code Matlab d'un émetteur fournie par le Telecommunications Research Center Vienna [70] a été utilisé. En outre, différents snapshots réels fournis par BMW ont également été testés pour valider les algorithmes qui ont été choisies.

A.3.3.2 Emulation du Récepteur IEEE 802.11p

Le prototype d'émulation du récepteur IEEE 802.11p est basés sur une exécution séquentielle. Ainsi, il n'exploite pas complètement la simultanéité des DSPs qui est possible sur la plate-forme. A l'heure actuelle, l'émulation du récepteur est considérée comme untimed. C'est pourquoi la concurrence n'est pas encore significative. Au lieu de cela, l'émulation est important pour identifier les fonctions DSP qui sont nécessaire pour la réalisation du récepteur dans un environnement purement logicielle. Le prototype d'émulation du récepteur IEEE 802.11p prend en charge tous

les différents schémas de modulation et taux de codage. Il a été annoté par des compteurs de cycle et étendu par la génération de fichiers de trace pour une évaluation efficace du récepteur. En dehors de cela il génère automatiquement des fichiers qui peuvent être utilisés pour tracer les résultats dans Matlab ou Octave. Toutes ces améliorations permettent une validation simple et l'identification des nécessaires améliorations algorithmiques dans un environnement purement logicielle.

A.3.3.3 Prototype Matériel du Récepteur IEEE 802.11p

Dans le flot de la future conception, le code écrit pour l'émulation peut être directement compilé pour la plate-forme matérielle, même pour les normes avec des vecteurs de petite taille. Actuellement, chaque fonction qui peut modifier un DSP paramètre lis la valeur du paramètre, effectue les modifications dans le processeur principal et écrit la valeur du registre de retour. Comme cela se fait pour chacun des paramètres, cette procédure prend beaucoup de temps (d'environ 425 ns par paramètre). Si on imagine qu'une opération de FEP exige au moins 14 paramètres, il est évident que cette procédure n'est pas efficace dans le cas de fortes exigences de latence. Pour répondre à ces fortes contraintes temps-réel, le code de l'émulation a donc été révisé et optimisé à plusieurs reprises avant d'être porté sur la plate-forme ExpressMIMO. Les améliorations inclus le choix d'un système d'exploitation approprié pour le processeur principal (plus précisément LEON3), un programmateur flexible pour exécuter les différents DSP simultanément, le regroupement de symboles OFDM et la génération de mots de commande hors ligne avant que le récepteur est démarré.

A.3.4 Résultats

Les résultats présentés ont été obtenus avec le prototype d'émulation du récepteur IEEE 802.11p et par un cycle précis HW / SW co-simulation. Auparavant, la chaîne de réception a été validé sur la plate-forme matérielle elle-même pour une fréquence de référence de 100 MHz. Les résultats ont été récupérées en utilisant JTAG et la connexion PCIExpress. Pour atteindre une meilleure performance cette fréquence sera augmentée dans un proche avenir. La fréquence maximale possible est déterminée par le processeur principal qui peut être traité à 133 MHz.
Pour obtenir des chiffres exacts sur les performances du récepteur, différents signaux de test ont été générés pour validation. En premier, les signaux de test générés par le modèle de référence Matlab ont été utilisés. Ces signaux sont basés sur l'exemple fourni dans l'annexe de la spécification standard et peuvent être générés pour n'importe qu'elle configuration du paquet. Deuxièmement, le récepteur a été vérifié en testant des snapshots différents fournis par notre partenaire de projet, BMW. Celles-ci ont été générés avec le Densobox, NEC Linkbird et une moto SimTD.

A.3.4.1 Résultats obtenus avec libembb

Le prototype d'émulation du récepteur IEEE 802.11p donne un premier aperçu de la consommation des ressources des DSPs différents avec le but de répondre aux questions suivantes:

1. Quelle DSP est utilisé la plupart du temps?
2. Combien de temps est nécessaire pour le traitement de transferts DMA?
3. Considérant que le temps de traitement, le récepteur peut-il être exécuté en temps réel sur la plate-forme? Si non, où sont les goulots d'étranglement?

4. Serait-il possible d'exécuter le récepteur avec d'autres émetteurs-récepteurs en parallèle? Si non, où sont les goulots d'étranglement

Grace à l'libembb, ces résultats peuvent déjà être obtenus à un stade de la conception. Ça permet une amélioration de l'implémentation dans un environnement purement logicielle.
Fondamentalement, le temps de traitement du récepteur sans la surcharge de communication peut être divisée en deux parties: (1) le temps quand les DSPs sont occupés et (2) le temps quand aucune ressource est utilisée. Ce dernier est le moment où aucun nouveau tâche peut être planifiée comme la fin du cycle d'acquisition du Préprocesseur n'est pas encore atteint.

Nos résultats montrent que, considérant que le temps de traitement pur des DSPs, les exigences de latence du récepteur IEEE 802.11p sont remplies. En plus, nous avons identifié le FEP en tant que moteur DSP critique comme la plupart des tâches sont à programmer sur cet accélérateur matériel. Compte tenu d'une exécution multimodal de IEEE 802.11p et de DAB, nous montrons que les exigences de latence de ces deux normes sont satisfaites si un seul des deux est exéutée. En outre, nous étudions le choix d'un flux de contrôle approprié et montrons comment un algorithme d'ordonnancement sophistiqué peut être réalisé.

A.3.4.2 Analyse de la performance d'éxécution - Résultats matériel

Les résultats matériels ont été obtenus par un cycle précis HW / SW co-simulation à l'aide de Modelsim. Sauf le temps de traitement les résultats présentés incluent maintenant la surcharge de communication quand un flux de contrôle centralisé est appliqué. En plus ils exploitent un traitement parallèle des différentes moteurs DSP sur la plate-forme ExpressMIMO. La surcharge de communication peut être observé lorsque aucun des DSP est occupé. En évaluant la relation entre ce facteur et le temps de traitement des DSP, des déclarations claires sur les performances du récepteur peut être faite.

Sur la base des résultats obtenus, nous pouvons affirmer

- que le récepteur IEEE 802.11p peut être exécuté en temps réel pour BPSK, QPSK et 16-QAM. En supposant une fréquence plus élevée comme il est automatiquement le cas lorsque ASICs sont considérées, 64-QAM peut exécuter en temps réel aussi.
- qu'une réduction supplémentaire de la surcharge de communication ne peut pas être atteint par un flux de contrôle distribué en utilisant les microcontrôleurs locaux ou par un microprocesseur ou un séquenceur sur la côté bande de base.
- que polling est la solution préférable pour déterminer la fin du traitement du DSP
- que les commandes doivent être préparées en avance pour les normes de latence critiques. Pour les normes comme DAB ou LTE, le comportement en temps réel est toujours garantie, même si cette recommandation n'est pas pris en compte. Lors du traitement d'une opération de vecteur sur une taille de 4096 échantillons, par exemple, le temps de traitement nécessaire serait de l'ordre de 20μs tandis que le temps de programmation du DSP reste à un maximum de 360 ns.

A.4 Conception ASIP pour le FEP

Dans le contexte des travaux sur le récepteur IEEE 802.11p, le FEP a été conçu comme un front-end générique pour OFDM / A (Orthogonal Frequency Division Multiplexing / Multiple Access), SC-FDMA (Single Carrier FDMA), W-CDMA (Wideband Code Division Multiple Access) et SDMA (Space Division Multiple Access). Pour l'évaluation de la norme IEEE 802.11p, nous avons considéré une solution qui est basée sur une DSP programmable (*C-FEP*). Cette version du FEP est composée d'une unité de traitement vectoriel qui est combinée avec une unité DFT / IDFT. Dans le chapitre précédent, nous avons identifié le besoin d'un FEP secondaire ou d'un supplément de DFT / IDFT pour augmenter la performance en particulier lorsque les normes basées sur des vecteurs de petite taille sont considerées. Un inconvénient principal de cette conception était la surcharge de communication qui a résulté dans une baisse significative de la performance. Il est intéressant de noter que ces limites sont liées seulement aux FPGAs et n'est pas aux ASICs. Pour le nouveau design du FEP nous avons pris la chance de collaborer avec l'Université RWTH Aachen (Allemagne) pour évaluer la méthodologie de conception des ASIPs pour la conception de la plate-forme ExpressMIMO. Un autre but de notre collaboration était de surmonter les inconvénients du C-FEP en enlevant l'unité DFT / IDFT du DSP et en remplaçant l'unité de traitement vectoriel par une solution ASIP qui est appelée A- FEP. Selon cette approche, l'A-FEP peut facilement être intégré dans le moteur de traitement de bande de base de la plate-forme ExpressMIMO. Comme ça, les tâches du FEP peut être facilement divisée sur deux FEP simultanément, par exemple. Pour l'évaluation de la conception, l'A-FEP n'est pas seulement comparée au C-FEP mais aussi à des solutions ASIP du milieu universitaire en termes d'architectures et en terme du temps de traitement.

Mais où est l'avantage principal d'ASIP par rapport à d'autres technologies? Parmi les facteurs importants à prendre en compte pour la conception des plate-formes SDR sont la consommation d'espace et de puissance ainsi que les coûts de production. Objectif majeur est de réduire la surface et de minimiser la puissance autant que possible en maintenant la performance. Dans [75], un aperçu détaillé des implémentations différentes des System on Chip (SoC) est prévue. Technologies d'intérêt sont

- **General Purpose Processors** qui peuvent être divisés en deux catégories: GPP proper appropriés pour les applications générales et microcontrôleurs pour applications industrielles.
- **Digital Signal Processor** qui sont une sous-catégorie d'Application Specific Processor (ASP). Ils sont par exemple utilisés pour les microprocesseurs programmables qui sont spécialisées pour le domaine de traitement de signal numérique.
- **Application Specific Integrated Circuits** qui sont aussi une sous-catégorie des ASPs. Ils sont mis en œuvre dans le matériel, le plus souvent en utilisant un Hardware Description Language (HDL) comme VHDL ou Verilog.
- **Application Specific Instruction-set Processors** qui sont une sous-catégorie des ASPs, aussi. Ils peuvent être vus comme une classe de microprocesseurs avec une Instruction-Set Architecture (ISA) spécialisée.

Les auteurs concluent que ASIPs ont une tendance à être de bons candidats car ils sont destinés à combler une lacune entre GPPs et ASICs. Etant adapté à une application spécifique, ASIPs offrent une plus grande souplesse que les ASICs en présentant une consommation énergétique plus faible que GPPs ou DSPs en même temps. Comme ca, ASIPs permettent de compromis la performance

des ASICs contre la flexibilité du GPPs. Le prototypage est facilitée en utilisant des outils de haut niveau alors que la conception générée n'est pas optimisée ne temps du matériel et ne peut pas s'adapter à la ressource dédiée (par exemple FPGA). De l'autre côté, VHDL permet une utilisation efficace des ressources du FPGA, bien que la mise en œuvre nécessite beaucoup de temps et beaucoup de ressources. Cet inconvénient est surmonté par des outils comme System Generator de Synopsis qui accélèrent le processus de conception VHDL par une conception de haut niveau et par le soutien des modifications rapides.

Tous les résultats ont été obtenus en collaboration avec l'Université RWTH Aachen (Allemagne). Pour la conception LISA, nous avons utilisé le Processor Designer de Synopsis (ex Coware). Tout au long de la collaboration, deux versions différentes de l'ASIP ont été développées:

1. La première version de l'A-FEP, appelé A-FEP-V1, a été conçu en collaboration avec un collègue et était basé sur la spécification du FEP qu'il tirait au cours de sa thèse de doctorat. Nous noterons cette première conception C-FEP-V1. Peu de temps après les résultats de nos travaux ont été présentés dans son rapport de thèse [37], la spécification du FEP a été retravaillé pour améliorer la performance. En plus, certaines fonctionnalités du A-FEP-V1 ont été intégrées dans la conception actuelle du C-FEP. Pour cette raison et aussi pour pallier les inconvénients de la première conception (principalement la faible fréquence), nous avons opté pour une seconde version du ASIP - bien que la première version était déjà très flexible.

2. La deuxième contribution est une conception nouvelle du ASIP basée sur la nouvelle spécification du FEP. Contrairement à la première version, l'A-FEP comporte également des instructions d'usage général. L'UC est maintenu dans le DSP pour le traitement des transferts DMA mais pas pour le traitement algorithmique. Par rapport à la première version de l'A-FEP la deuxième version est livré avec un jeu d'instructions élargi et obtient une fréquence plus élevée.

A.4.1 Exigences du Moteur de Traitement

Les exigences de traitement front-end pour le soutien de l'OFDM / A, SC-FDMA, CDMA et W-SDMA a déjà été détaillés dans [23], entre autres. Ce document indique que les opérations à exécuter par le processeur comprennent entre autres l'estimation de canal et synchronisation. Cettes opérations peuvent être construite en utilisant des opérations vectorielles et une unité DFT / IDFT. Celui-ci est négligée pour l'A-FEP et conservée comme un moteur de traitement distinct dans la conception de bande de base de la plate-forme ExpressMIMO.
L'ensemble des opérations vectorielles être pris en charge par l'A-FEP est répertoriée dans le tableau Tab. A.1. Par ailleurs, les opérations shift, max / min et argmax / argmin sont inclus qui peuvent fonctionner de manière indépendante sur les parties réelles et imaginaires des éléments des vecteurs traitées. En outre, les valeurs peuvent être modifiée avant et après l'opération vectorielle, comprenant valeur absolue, la négation, mise à zéro, mise à l'échelle et de saturations. Les éléments du vecteur d'entrée et du vecteur du sortie peuvent être de quatre types de données différentes: 8 ou 16 bits entiers signés et les nombres complexes ayant une taille de 16 ou 32 bits. Les conversions de type entre eux sont spécifiées par les paramètres faisant partie du mot d'instruction.
Un défi majeur quand un large éventail de normes différentes est supportée est de s'assurer que chacun d'entre eux répond aux contraintes de temps réel. Par conséquent, l'A-FEP est livré avec une unité de génération d'adresses programmable (AGU) qui permet de construire des vecteurs d'entrée à partir d'ensembles de données non contigues dans le MSS connecté. Symétriquement,

l'AGU peut également être utilisée pour stocker des vecteurs de résultat en adresses non contigus, ce qui permet la répétition des valeurs par exemple. De plus, l'auto-programmables mécanismes permettent de transformer sections du MSS en zones de stockage circulaires (Fig. A.5).

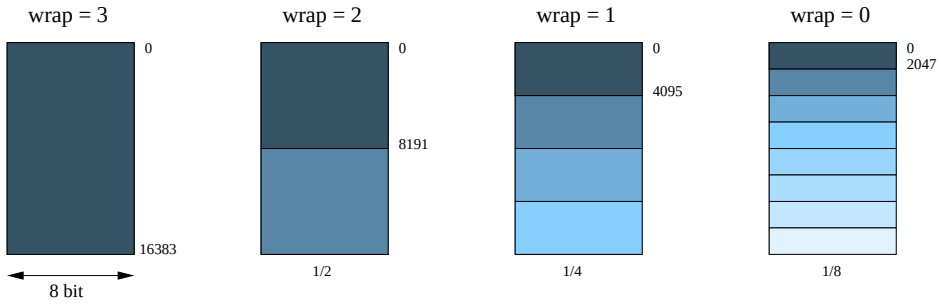


Figure A.5: Zones de Stockage Circulaires FEP MSS (`int8`)

Le MSS contient le Program Memory avec une taille de 4 kB et l'espace d'entrée-sortie de données qui a été conçu pour le support des normes qui opèrent sur des formats vectoriels volumineux tels que LTE ou DAB. Il est divisé en quatre blocs de mémoire différents, chacun avec une taille de 4096 entrées de 32 bit. La longueur du vecteur maximal qui peut être traitée dépends du type de données. Pour les éléments de vecteur avec une taille de 32 bits la longueur maximale est 4096 lorsqu'il est 16384 pour une taille de 8 bits.

Component-Wise Addition	$Z[i] = X[i] + Y[i]$
Component-Wise Product	$Z[i] = X[i] \times Y[i]$
Component-Wise Square Absolute	$Z[i] = \|X[i]\|^2$
Move	$Z[i] = X[i]$
Component-Wise Division	$Z[i] = X[i]/Y[i]$
Vector Sum	$Z = \sum X[i]$

Table A.1: A-FEP - Opérations Vectorielles

A.4.2 Architecture HW et Instruction-Set

Le jeu d'instructions de l'A-FEP comprend trois types d'instructions différentes:

1. **AGU configuration instructions:** Ces instructions portent les paramètres nécessaires à la programmation de l'AGU. Six instructions différentes ont été mises en œuvre dont la quantité dans le code du programme peut varier en fonction de la quantité de paramètres à modifier pour l'instruction de traitement arithmétique vectorielle qui suivre.

2. **Arithmetic Vector Processing (AVO) instructions:** Pour répondre aux exigences des moteurs de traitement, l'A-FEP prend en charge neuf instructions différentes: multiplication, addition, square, square modulus, sum, shift, move, division et max, min. Longueur maximale du vecteur supporté est de 16384 entrées pour un vecteur composé des éléments vectoriels de 8 bits.

3. **General Purpose (GP) instructions:** Ces instructions sont basées sur une architecture load-store et prennent en charge les instructions comme compare, branch ou IRQ qui est utilisé pour signaler le processeur principal la fin d'une tâche planifiée. Cettes tâches peuvent représenter une seule instruction ou des algorithmes plus complexes comme la synchronisation des paquets.

La structure de la pipeline est illustrée dans Fig. A.6. Elle se compose de onze étapes et traite deux éléments vectoriels par cycle. Habituellement, une instruction par cycle est extraité de la mémoire de programme. Une exception sont les instructions AVO qui peuvent travailler avec des vecteurs de longueur variable.

Quand l'A-FEP est synthétisé pour la cible FPGA, il obtient une fréquence de 105 MHz en exigeant 3122 function generators, 3281 CLB slices, 6433 DFFs, 17 block RAMs et 8 DSP48E slices. Pour la cible ASIC, seul le moteur de traitement a été synthétisé comme la nouvelle conception du MSS fait partie de notre travail en cours. La fréquence maximale pouvant être atteinte est d'environ 550 MHz, la surface est de 0,18 mm^2.

A.4.3 Comparaison des Performances d'Exécution

La performance d'exécution dépend de deux facteurs: le temps de traitement nécessaire pour la communication entre le processeur principal et les moteurs de bande de base et le temps de traitement des données pures des DSP. Pour une norme qui opère sur les vecteurs de petite taille comme IEEE 802.11p, le premier facteur est d'une importance majeure alors qu'il est plus ou moins négligeable pour les normes comme LTE qui opèrent sur de grands vecteurs. Le tableau Tab A.4.3 liste les temps d'exécution du A-FEP pour les différents algorithmes de traitement front-end d'un récepteur IEEE 802.11p pour une fréquence de 100 MHz. Structure de paquet et la procédure de la décodage ont été présenté dans [3].
Pour la démonstration et pour comparer les performances des différentes solutions présentées le A-FEP et comparée avec le C-FEP et deux autres solutions du milieu universitaire ([77], [76]).

algorithm	**cycles**	**execution time**
energy detection	302	3.06 μs
channel estimation	45	0.45 μs
data detection (16-QAM, init)	172	1.72 μs
data detection (16-QAM)	114	1.14 μs
data detection (64-QAM, init)	219	2.19 μs
data detection (64-QAM)	342	3.42 μs

Table A.2: A-FEP Performances d'Exécution

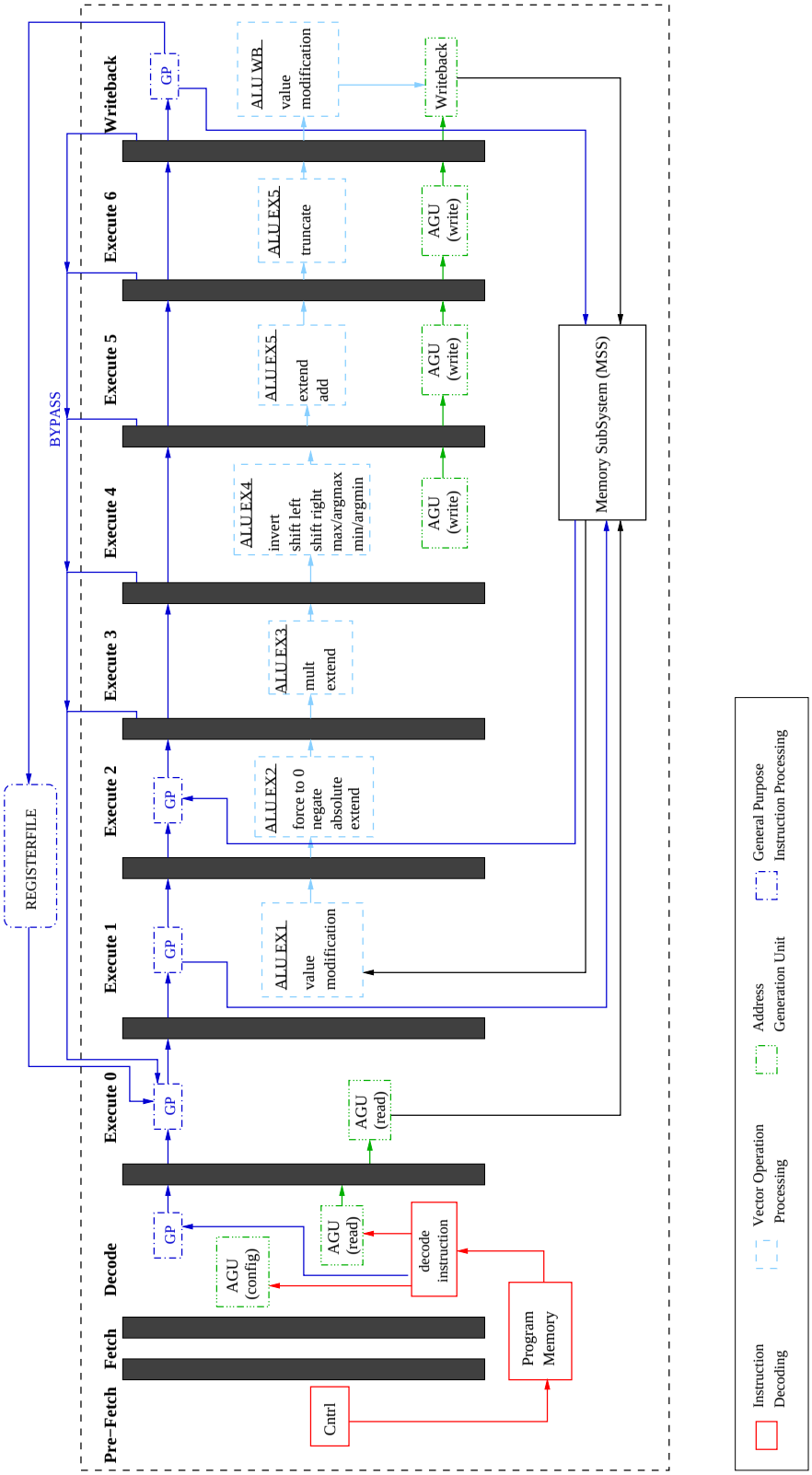


Figure A.6: Structure de la Pipeline du A-FEP

A.5 Conception flexible d'un Sample Rate Converter

A.5.1 Motivation

Le Préprocesseur établit la connexion entre les convertisseurs A/D et D/A (ADA) à travers l'interface de l'ADA et le moteur de bande de base restant. Cette tâche est assez difficile car la fréquence d'horloge sur le côté convertisseur est 32,768 MHz alors que cela dépend de la norme de communication sans fil exécuté sur le côté bande de base. Pour DAB, par exemple, la fréquence d'échantillonnage en bande de base est 2,048 MHz, tandis que pour IEEE 802.11p il est fixé à 10 MHz (Fig. A.7). Il en résulte un facteur de ré-échantillonnage de 15 pour DAB et de 3,2768 pour IEEE 802.11p.
La relation entre ces différents taux d'échantillonnage est généralement assurée par des SRC qui sont des architectures bien connus et qui sont non seulement appliquées dans les systèmes de communication sans fil, mais aussi dans les processus d'image par exemple. Pour les systèmes de SDR, ils sont l'un des éléments les plus critiques et les plus exigeants [104].
Les défis lors de la conception d'une solution SRC approprié pour la plate-forme ExpressMIMO sont les suivants:

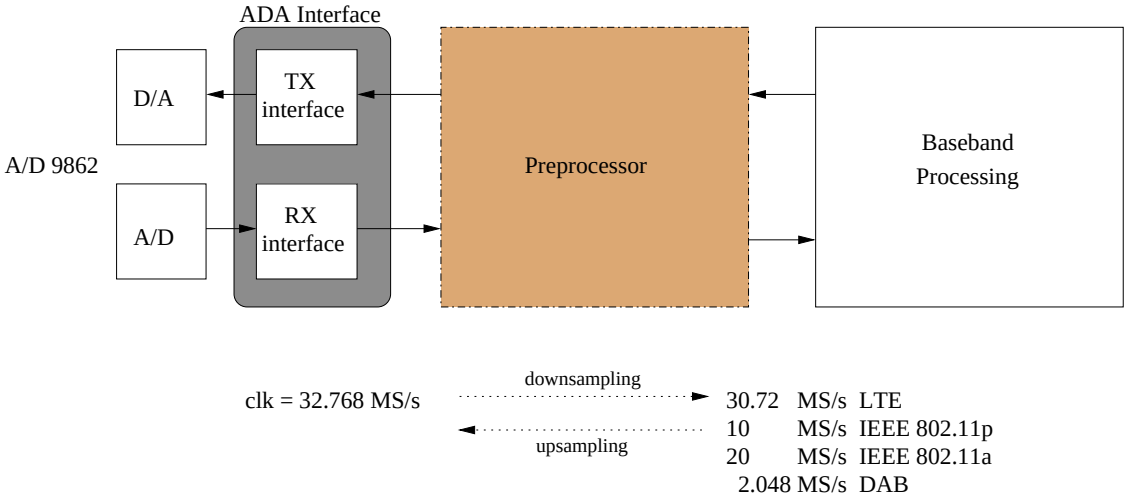


Figure A.7: Le Préprocesseur relie les Convertisseurs ADA avec le Moteur de Bande de Base restant

- Une analyse détaillée des normes de communication sans fil de nos jours a montré que le SRC doit prendre en charge une gamme de fréquences de 3 MHz $\leq f_{samp} \leq 61,44$ MHz avec une résolution de 1 Hz.

- Dans le passé, généralement un SRC a été utilisé par norme. Pour cette large gamme de fréquences, cette approche n'est pas applicable car les ressources nécessaires sont bien au-delà ce qui est disponible sur la cible FPGA. Le SRC doit donc supporter tous les rapports possibles de fréquence d'échantillonnage (les entiers et les fractionnaires) avec une seule architecture.

- En dehors de cela la plate-forme ExpressMIMO peut traiter jusqu'à quatre canaux différents en RX et jusqu'à quatre canaux différents en TX. Chaque canal est définie par son propre

ensemble de paramètres. Ainsi, lors de la commutation entre les deux canaux, le système peut changer son comportement dynamiquement à l'exécution.

Ces défis de conception conduisent à des exigences de traitement différentes qui peuvent être regroupées dans exigences fonctionnels et exigences non-fonctionnels. Du point de vue de la plate-forme, il est très important que la quantité des DSP48E slices est réduite autant que possible. Cette tâche n'est pas si simple que en raison de la grande largeur de bande du signal provenant du convertisseur A / D, le débit de données est très élevé. Cela conduit à une plus grande complexité du matériel et une plus grande consommation d'énergie et entraîne une augmentation du nombre de DSP48E slices et donc une application très coûteux. En outre, la conception du Préprocesseur doit suivre la démarche de conception comme tous les autres DSP sur la plate-forme.

En outre, les exigences fonctionnelles comprennent

- la préférence d'une conception générique qui peut effectuer le sur- et le sous-échantillonnage fractionnel en utilisant la même architecture. Sur-échantillonnage (upsampling) / Sous-échantillonnage (downsampling) correspond à augmenter / diminuer le taux d'échantillonnage. Pour émetteurs-récepteurs lorsque le débit de données au niveau du côté convertisseur ADA est plus élevée que celle du moteur de la bande de base, sur-échantillonnage est effectué en TX, tandis que le sous-échantillonnage est effectué en RX.
- l'appui de trois modes différents: (1) seulement réception, (2) seulement transmission et (3) réception et transmission simultanément. A partir de la perspective de la plate-forme, les canaux de RX et TX sont exécutés en parallèle. Ils sont traité par le SRC en façon Round Robin. Par conséquent, le commutateur de canal doit se faire dans un cycle (Fig. A.8).

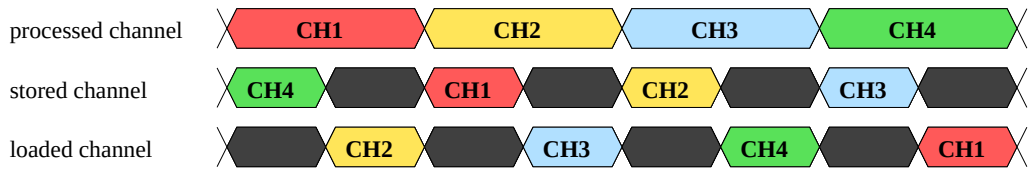


Figure A.8: Channel Scheduling

- d'éviter l'aliasing pendant la ré-échantillonnage. Dans ce contexte, il est important de trouver une bonne longueur du filtre et donc le nombre de multiplicateurs
- le calcul des valeurs intermédiaires d'un signal temps discret de telle sorte qu'une certaine bande de fréquence du signal ne soit pas faussée [105].
- ce qu'une performance élevée doit être garantie pour répondre aux exigences de débit et de temps de latence des différentes normes de la communication sans fil.
- que le SRC prend soin de la différence entre les fréquences d'échantillonnage. Cette approche permet de fixer l'horloge maître des convertisseurs ADA pour diminuer le phase noise [105]

La contribution principale présentée dans cette thèse est la conception d'un SRC fractionnaire pour la plate-forme ExpressMIMO qui est basée sur l'algorithme d'interpolation à bande limitée. Son

architecture peut traiter jusqu'à quatre canaux différents en RX (sous-échantillonnage) et jusqu'à quatre canaux différents en TX (sur-échantillonnage). Tous les canaux sont exécutés sur la même architecture matérielle qui est paramétrable. Afin de garantir un traitement continu du filtre, les changements de contexte entre des canaux différent se produisent instantanément pendant un seul cycle.
Les modèles qui sont fournis comprennent des modèles en C pour les mesures de quantification et l'analyse des caractéristiques du filtre, ainsi que un prototype VHDL.
Le SRC est intégré dans le Préprocesseur. Pour finaliser la chaîne du récepteur IEEE 802.11p un premier prototype du Préprocesseur a été décrit en VHDL et évalués en utilisant Modelsim.

A.5.2 Design du Préprocessor et du SRC

Comme déjà dit, le Préprocesseur relie le module RF externe avec le moteur de traitement en bande de base numérique. Pour établir cette connexion, le DSP a été légèrement modifié par une interface dédiée pour un accès direct entre l'unité de traitement et l'interface ADA (Fig. A.9). Le dernier gère le (de)multiplexage des échantillons complexes provenant de et allant vers les convertisseurs A / D et D / A. Dans RX / TX, le signal fourni par les convertisseur A / D, D / A a une résolution de 12 bits / 14 bits. Sign extension et bit removal devient nécessaire parce que le Préprocesseur opère sur des échantillons dans un format Q1.15. Ces tâches sont aussi assurées par l'interface ADA.

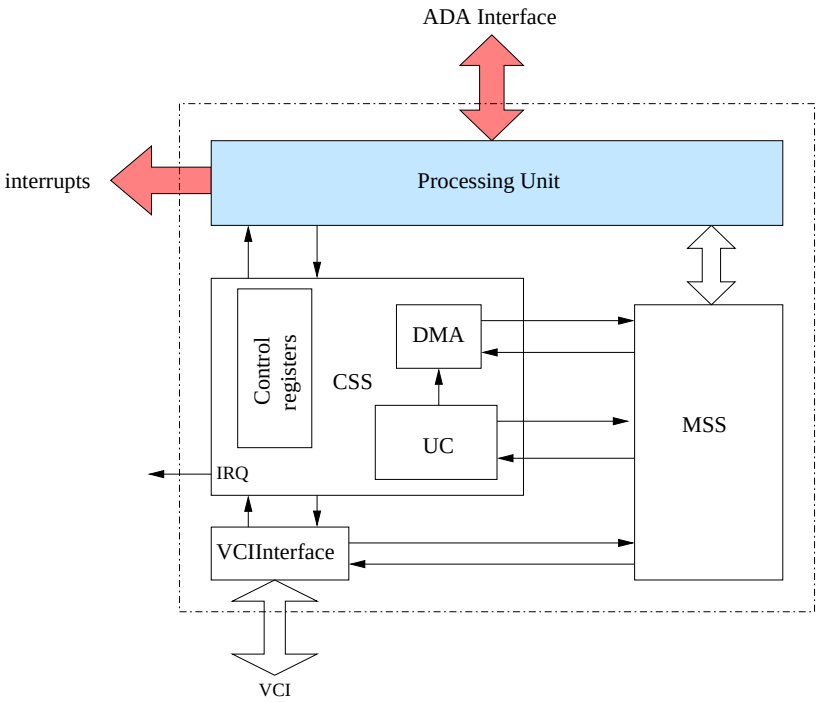


Figure A.9: DSP Modifications

Les tâches principales du Préprocessor sont: Interface avec les convertisseurs ADA, I/Q imbalance correction, NCO pour le carrier frequency adjustment et les fonctions de base de traitement du signal telles que la conversion de fréquence d'échantillonnage
Afin de garantir une performance élevée de ces tâches, ils sont répartis sur différents modules internes qui sont I/Q imbalance (I/Q), une unité de pré-distorsion dans TX (PD), NCO et SRC.

Chacun des deux modes différents supporte quatre canaux différents qui peuvent posséder un ensemble de différent paramètres. Tous les modules sont surveillés par une unité de commande globale du Préprocesseur comme illustré dans Fig. A.10. Les tâches principales de cette machine d'état sont (1) programmer les canaux configurés / actif dans un façon Round Robin, (2) déclencher les transferts de données en lecture / écriture a travers l'interface ADA et MSS, (3) mettre à jour les paramètres nécessaires pour programmer les différents modules, (4) générer des interruptions à la fin d'un cycle d'acquisition et (5) superviser la commutation entre les canaux.

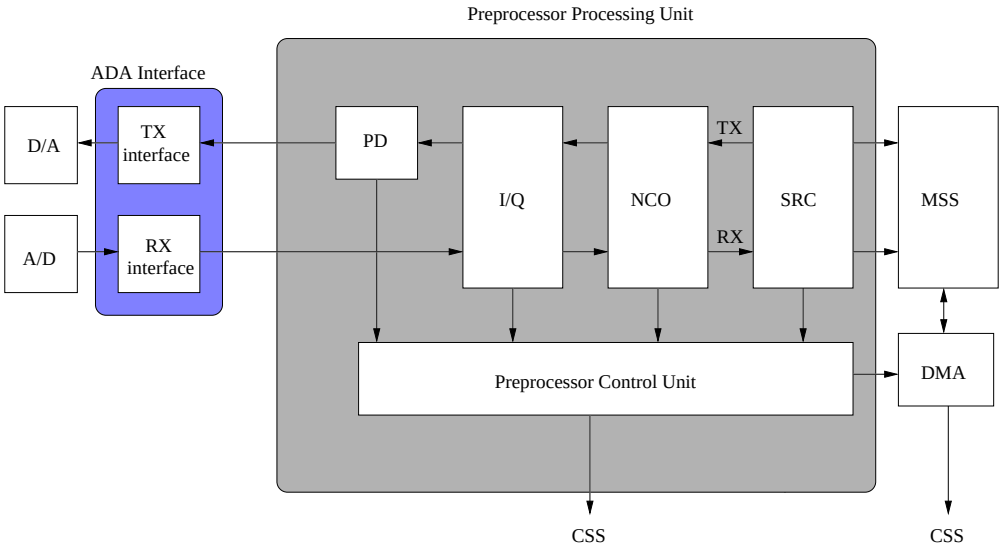


Figure A.10: Architecture du Préprocesseur

Les différents modules internes communiquent via un protocole de handshaking. Ce protocole garantit des transferts de données valides et arrête la chaîne de traitement en cas d'absence de données. Quand un module fournit de nouvelles données, le signal de demande de données REQ est réglé en même temps. Le transfert de données a réussi une fois le signal d'acquittement ACK est reçu.

Le MSS et l'espace de mémoire qui sont inclus dans l'interface contiennent des FIFOs différents pour le stockage des échantillons de sortie et d'entrée. Ces FIFOs sont des composants autonomes qui gèrent leur propre espace mémoire. Pour éviter la perte d'échantillons, ils informent le Préprocesseur dans le cas où ils sont presque pleine ou presque vide.

Sur la base de cette spécification fonctionnelle, les instructions suivantes pour la conception du SRC peuvent être faites:
(1) Les mises à jour de paramètres sont gérés par l'unité de commande du Préprocesseur et ne sont donc pas du ressort du SRC.
(2) Le moment dans le temps où un commutateur de canal doit se produire est déterminée par l'unité de commande du Préprocesseur. Le SRC doit garantir que la commutation se passe instantanément une fois qu'il est informé de cet événement.
(3) La communication entre le SRC et le NCO doit suivre l'algorithme du handshaking.
(4) Le mode de suspension est assurée par l'unité de commande du Préprocesseur. Le SRC fonctionne comme d'habitude. La seule différence est qu'aucun échantillon est passé.

La conception du filtre inclue dans le SRC est basé sur l'algorithme d'interpolation à bande limitée présentée dans [137] et [138]. Les avantages principaux de cet algorithme sont (1) que les architectures pour le sur- et le sous-échantillonnage sont les mêmes, (2) que l'architecture peut être optimisé pour une conception efficace du filtre et (3) que la combinaison de Shannon-Whittaker interpolation et d'interpolation linéaire résulte dans une performance élevé avec une consommation d'espace raisonnable.

A.5.3 Résultats

Lors de la synthèse du SRC pour la cible FPGA bande de base, une fréquence maximale de 130,005 MHz est obtenu après placement-routage. Ressources nécessaires sont
- 32899 function generators (15.87 %)
- 8225 CLB slices (15.87 %)
- 20013 DFFs or latches (9.54 %)
- 30 Block RAMs (10.42 %)
- 82 DSP48E slices (42.71 %)

Pour le premier prototype du Préprocessor, une fréquence maximale de 98,261 MHz a été obtenue après placement-routage. Les ressources nécessaires dans ce cas sont
- 41007 function generators (19.78 %)
- 10252 CLB slices (19.78 %)
- 26206 DFFs or latches (12.49 %)
- 55 Block RAMs (19.10 %)
- 82 DSP48E slices (42.71 %)

L'évaluation de la performance montre que la performance du SRC dépend principalement de deux facteurs: (1) le taux d'échantillonnage entre des normes de communication sans fil en cours et (2) le rapport entre les taux d'échantillonnage. Il est évident que la performance augmente quand moins d'interpolation est nécessaire et quand le rapport entre les fréquences d'échantillonnage diminue. Extensions de conception possible pour obtenir d'excellentes performances, même pour des ratios élevés de sur-échantillonnage pourrait être

- diviser le rapport en deux parties: un nombre entier et d'une nombre fractionnel. Ré-échantillonnage entier pourrait être effectuée par des filtres CIC, tandis que notre solution SRC pourrait être utilisé pour la partie fractionnel.
- le traitement du SRC en plusieurs fois. Cette approche ne convient pas comme un traitement en continu du SRC ne serait plus garantie dans ce cas.

Par ailleurs, la performance de la conception actuelle pourrait être augmentée par la mise en œuvre d'un registerbank adressable ou par la réalisation de filtres d'ordre supérieur. Celui-ci est livré avec l'inconvénient d'un temps d'initialisation plus long et avec plus de mémoire nécessaire pour le stockage de coefficients de filtrage. En dehors de cela, il est également possible d'envisager d'augmenter la fréquence du Préprocesseur en ajoutant plus de registres entre le MSS et le RAM réelle.

A.6 Conclusion

Le travail présenté dans cette thèse est fortement liée aux tendances dans l'industrie automobile qui demande une combinaison de C2X et de TPEG. Normes d'intérêt sont IEEE 802.11p et ETSI DAB. Pour combiner ces deux, nous avons choisi l'approche d'une plate-forme SDR flexible qui ne se limite pas au contexte de l'automobile mais aux normes de communication sans fil en général. La thèse est basée sur une conception de la couche physique efficace du récepteur IEEE 802.11p pour la plate-forme OpenAirInterface ExpressMIMO en suivant la méthodologie de développement de base comme décrit dans le Chapitre 2.3. Cela comprenait (1) le développement de modèles purement fonctionnels en utilisant la bibliothèque d'émulation de la plate-forme, libembb, (2) le cycle précis HW / SW co-simulation via Modelsim et (3) la validation du récepteur sur la plate-forme matérielle. Un composant manquant au dŐbut est le Préprocesseur où une première solution comprenant un Sample Rata Converter (SRC) a été fourni. Sur la base des résultats obtenus, nous avons identifié les goulots d'étranglement de conception et nous avons présenté des solutions possibles pour remédier à ces inconvénients. En dehors de cela nous avons eu un coup d'oeil à un traitement multimodal des deux normes d'intérêts, IEEE 802.11p et ETSI DAB.

Ces objectifs ont été exprimés dans une liste de tâches différentes qui ont dû être accompli au cours de cette thèse:

1. Emulation du récepteur IEEE 802.11p avec l'aide de la Library for ExpressMIMO baseband appelée *libembb*
2. La mise en œuvre du récepteur IEEE 802.11p et l'évaluation de sa performance sur la plate-forme ExpressMIMO
3. Savoir comment DAB et IEEE 802.11p peuvent être exécutées simultanément sur la plate-forme ExpressMIMO
4. Identification des goulots d'étranglement et la fourniture de solutions possibles
5. La mise en œuvre d'un prototype de Préprocesseur pour compléter la chaîne du récepteur IEEE 802.11p

Pour résumer, nous pouvons affirmer que la mise en œuvre de normes d'exploitation de grands vecteurs peut être déjà effectué d'une manière très efficace sur la plate-forme ExpressMIMO lorsque FPGAs sont considérées comme la technologie cible. Lors de l'exécution d'une opération sur un vecteur de taille de 4096 échantillons, par exemple, le temps de traitement nécessaire serait de l'ordre de 20 μs tandis que le temps de programmation du DSP reste à un maximum de 360 ns. La surcharge de communication qui en résulte peut donc être négligée.
Lorsque le traitement des normes basée sur des court données, le code a été actuellement manuellement optimisée par une préparation de commande, groupement de symboles, d'interrogation au lieu d'interruptions, etc . La conception du récepteur qui en résulte est donc plus compliqué mais en suivant ces recommandations, une performance élevée peut être obtenue, même pour cette type de norme.

Bibliography

[1] http://www.openairinterface.org/projects/proton_plata.en.htm.

[2] C. Schmidt-Knorreck, M. Ihmig, R. Knopp, and A. Herkersdorf. Multi-Standard Processing using DAB and 802.11p on Software Defined Radio Platforms. In *WSR2012, 7th Karlsruhe Workshop on Software Radios*, 2012.

[3] C. Schmidt-Knorreck, D. Knorreck, and R. Knopp. IEEE 802.11p Receiver Design for Software Defined Radio Platforms. In *Conference on Digital System Design Architectures, Methods and Tools, 2012. DSD '12. 15th EUROMICRO*, sept. 2012.

[4] http://www.newcom-project.eu/.

[5] http://www.ict-acropolis.eu.

[6] 216715 NEWCOM++ DRC.3, Performance Evaluation and Guidelines for Future Flexible Radio Architectures.

[7] C. Schmidt-Knorreck, R. Pacalet, A. Minwegen, U. Deidersen, T. Kempf, R. Knopp, and G. Ascheid. Flexible Front-End Processing for Software Defined Radio Applications using Application Specific Instruction-Set Processors. In *Conference on Design and Architectures for Signal and Image Processing, DASIP'12*, oct. 2012.

[8] C. Schmidt-Knorreck, R. Knopp, and R. Pacalet. Hardware Optimized Sample Rate Conversion for Software Defined Radio. In *WSR2010, 6th Karlsruhe Workshop on Software Radios, March 3-4, 2010, Karlsruhe, Germany*, Karlsruhe, GERMANY, 03 2010.

[9] C. Schmidt-Knorreck, R. Knopp, and R. Pacalet. Hardware Optimized Sample Rate Conversion for Software Defined Radio. *FREQUENZ, Journal of RF-Engineering and Telecommunications, November-December 2010*, 64:11– 12, 12 2010.

[10] J. Mitola. The Software Radio Architecture. *Communications Magazine, IEEE*, 33(5):26 –38, may 1995.

[11] http://www.wirelessinnovation.org.

[12] First European Colloquium on Re-configurable Radio Systems & Networks, 4 March 1999, Q.E. II Conference Center, London, UK.

[13] Jacques Palicot and Tim Hentschel. Software radio: Implementation aspects. *Annales des Télécommunications*, 57(7-8):567–569, 2002.

[14] R.I. Lackey and D.W. Upmal. Speakeasy: the Military Software Radio. *Communications Magazine, IEEE*, 33(5):56 –61, may 1995.

[15] J. Melby. JTRS and the Evolution toward Software-Defined Radio. In *MILCOM 2002. Proceedings*, volume 2, pages 1286 – 1290 vol.2, oct. 2002.

[16] FCC. *47 C.F.R. Section 2.1., Facilitating Opportunities for Flexible, Efficient, and Reliable Spectrum Use Employing Cognitive Radio Technologies, ET Docket No. 03-108, Report and Order, 20 FCC Rcd 5486*, 2005.

[17] http://www.vanu.com.

[18] http://www.gnuradio.org.

[19] http://www.ettus.com.

[20] S. Knowless. The SoC Future is Soft. In *IEEE Cambridge Processor Seminar*, December 2005.

[21] http://www.nvidia.com/object/nvidia-icera-technologies.html.

[22] U. Ramacher. Software-Defined Radio Prospects for Multistandard Mobile Phones. *Computer*, 40(10):62 –69, oct. 2007.

[23] K. van Berkel, F. Heinle, P. P. E. Meuwissen, et al. Vector Processing as an Enabler for Software-Defined Radio in Handheld Devices. *EURASIP Journal on Application Signal Processing*, 2005:2613–2625, January 2005.

[24] 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. In *IEICE TRANS. ELECTRON*, volume E85, pages 359 –367, february 2002.

[25] Freescale Semi. Inc. *MSC8156: Six Core High Performance DSP*, 2011.

[26] J. Glossner, D. Iancu, M.N Moudgill, G. Nacer, S. Jinturkar, S. Stanley, and M. Schulte. The Sandbridge SB3011 Platform. *EURASIP J. Embedded Syst.*, 2007(1):16–16, January 2007.

[27] G. Cichon, P. Robelly, H. Seidel, E. Matus, M. Bronzel, and G. Fettweis. Synchronous Transfer Architecture (STA). In *Lecture Notes on Computer Science*, pages 126–130. Springer-Verlag, 2004.

[28] H. Corporaal. *Microprocessor Architectures: From VLIW to TTA*. John Wiley & Sons, Inc., New York, NY, USA, 1997.

[29] F. Clermidy, C. Bernard, R. Lemaire, J. Martin, I. Miro-Panades, Y. Thonnart, P. Vivet, and N. Wehn. MAGALI: A Network-on-Chip based Multi-Core System-on-Chip for MIMO 4G SDR. In *IEEE International Conference on IC Design and Technology (ICICDT), 2010*, pages 74 –77, june 2010.

[30] N.-u.-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*, pages 39 –46, sept. 2008.

[31] http://www.openairinterface.org.

[32] http://www.gaisler.com/leonmain.html.

[33] http://www.xilinx.com.

[34] http://ecos.sourceware.org/ecos/docs-2.0/ref/hal-smp-support.html.

[35] http://www.mutekh.org.

[36] A. Becoulet. *Conception d'un système d'exploitation supportant nativement les architectures multiprocesseurs hétérogènes à mémoire partagée*. PhD thesis, L'Universite Pierre et Marie Currie - Paris VI, 2010.

[37] N.-u.-I. Muhammad. *Flexible Baseband Architecture Design & Implementation for Wireless Communication Systems*. PhD thesis, Télécom ParisTech, 2010.

[38] VSIA consortium: `http://www.vsi.org/`.

[39] Vsi Alliance Virtual Component Interface Standard Version 2 (OCB 2 2.0).

[40] http://www.ict-sacra.eu.

[41] Stephen Ezell. Explaining International IT Application Leadership: Intelligent Transport Systems, january 2010.

[42] A. Lüebke. *Evaluation Criteria for Multi Channel Operation*. Technical report, Car 2 Car Communication Consortium, November 2007.

[43] 2008/671/EC. *Commision Decision on the Harmonized Use of Radio Spectrum in the 5875-5905 MHz Frequency Band for Safety-Related Applications of Intelligent Transport Systems (ITS)*, August 2008.

[44] ETSI, Draft ESTI EN 302 571 V1.1.1. *Intelligent Transport Systems (ITS); Radiocommunications Equipment Operating in the 5855 MHz to 5925 MHz Frequency Band; Harmonized EN Covering Essential Requirements of Article 3.2 of the R&TTE Directive*, September 2008.

[45] IEEE Std. 802.11p/D9.0. *Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 7: Wireless Access in Vehicular Environments*, july 2009.

[46] D. Jiang and L. Delgrossi. IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments. In *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE*, pages 2036 –2040, may 2008.

[47] IEEE. *IEEE 802.11a: Wireless LAN Medium Access Control and Physical Layer Specifications, High-Speed Physical Layer in the 5 GHz Band*, september 1999.

[48] D. Jiang, V. Taliwal, A. Meier, W. Holfelder, and R. Herrtwich. Design of 5.9 GHz DSRC-Based Vehicular Safety Communication. *Wireless Communications, IEEE*, 13(5):36 –43, october 2006.

[49] http://www.simtd.org.

[50] S. Eichler. Performance Evaluation of the IEEE 802.11p WAVE Communication Standard. In *Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th*, pages 2199 –2203, 30 2007-oct. 3 2007.

[51] K. Bilstrup, E. Uhlemann, E.G. Strom, and U. Bilstrup. Evaluation of the IEEE 802.11p MAC Method for Vehicle-to-Vehicle Communication. In *Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th*, pages 1 –5, sept. 2008.

[52] A. Paier, R. Tresch, A. Alonso, D. Smely, P. Meckel, Y. Zhou, and N. Czink. Average Downstream Performance of Measured IEEE 802.11p Infrastructure-to-Vehicle Links. In *IEEE International Conference on Communications Workshops (ICC), 2010*, pages 1 –5, may 2010.

[53] M. Wellens, B. Westphal, and P. Mahonen. Performance Evaluation of IEEE 802.11-based WLANs in Vehicular Scenarios. In *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*, pages 1167 –1171, april 2007.

[54] F. Martelli, M.E. Renda, and P. Santi. Measuring IEEE 802.11p Performance for Active Safety Applications in Cooperative Vehicular Systems. In *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, pages 1 –5, may 2011.

[55] G. Acosta-Marum and M.A. Ingram. Six Time- and Frequency- Selective Empirical Channel Models for Vehicular Wireless LANs. *Vehicular Technology Magazine, IEEE,* 2(4):4 –11, dec. 2007.

[56] T. M. Fernández-Caramés, M. González-López, and L. Castedo. FPGA-Based Vehicular Channel Emulator for Real-Time Performance Evaluation of IEEE 802.11p Transceivers. *EURASIP J. Wirel. Commun. Netw.*, 2010:4:1–4:10, April 2010.

[57] M. Abbasi. Characterization of a 5 GHz Modular Radio Frontend for WLAN based on 802.11p. Master's thesis, TU Wien, December 2008.

[58] V. Shivaldova. Implementation of IEEE 802.11p Physical Layer Model in SIMULINK. Master's thesis, TU Wien, Juni 2010.

[59] S. Shooshtary. Development of a MATLAB Simulation Environment for Vehicle-to-Vehicle and Infrastructure Communication Based on IEEE 802.11p. Master's thesis, TU Wien, December 2008.

[60] Q. Chen, D. Jiang, V. Taliwal, and L. Delgrossi. IEEE 802.11 Based Vehicular Communication Simulation Design for NS-2. In *Proceedings of the 3rd International Workshop on Vehicular Ad Hoc Networks (VANET)*, pages 50 – 56, september 2006.

[61] NEC. *LinkBird-MX Version 3 Datasheet*.

[62] E. Lambers, M. Klassen, A. Kippelaar, et al. *DSRC Mobile WLAN Component*. NXP Semiconductors.

[63] R. K. Schmidt, T. Leinmuller, and B. Boddeker. V2X Kommunikation. February 2009.

[64] R. Lasowski, T. Leinmüeller, and M. Strassberger. OpenWAVE Engine / WSU - A Platform for C2C-CC. In *Proceedings of the 15th World Congress on Intelligent Transport Systems*, 2008.

[65] P. Fuxjaeger, A. Costantini, P Castiglione, et al. IEEE 802.11p Transmission using GNU Radio. In *WSR2010, 6th Karlsruhe Workshop on Software Radios*, march 2010.

[66] http://www.aktiv-online.org/english/aktiv-cocar.html.

[67] A. Fort, J.-W. Weijers, V. Derudder, W. Eberle, and A. Bourdoux. A Performance and Complexity Comparison of Auto-Correlation and Cross-Correlation for OFDM Burst Synchronization. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03), 2003*, volume 2, pages II – 341–4 vol.2, april 2003.

[68] L. Bernado, N. Czink, T. Zemen, and P. Belanovic. Physical Layer Simulation Results for IEEE 802.11p Using Vehicular Non-Stationary Channel Model. In *IEEE International Conference on Communications Workshops (ICC), 2010*, pages 1 –5, may 2010.

[69] R. Ghaffar and R. Knopp. Low Complexity Metrics for BICM SISO and MIMO Systems. In *Vehicular Technology Conference (VTC 2010-Spring), 2010 IEEE 71st*, pages 1 –6, may 2010.

[70] http://www.ftw.at.

[71] T. Zetterman, A. Piipponen, K. Raiskila, and S. Slotte. Multi-Radio Coexistence and Collaboration on an SDR Platform. *Analog Integrated Circuits and Signal Processing*, 69:329–339, 2011. 10.1007/s10470-011-9713-7.

[72] *Multi-Radio Scheduling and Resource Sharing on a Software Defined Radio Computing Platform*, 2009.

[73] L. Stolz, M. Feilen, and W. Stechele. An Optimized Software-Defined Digital Audio Broadcasting (DAB) Receiver for x86 Platforms. In *WSR2012, 7th Karlsruhe Workshop on Software Radios*, 2012.

[74] Matthias Ihmig, Nicolas Alt, and Andreas Herkersdorf. Implementation and Fine-grain Partitioning of a DAB SDR Receiver on an FPGA-DSP Platform. In *WSR2010, 6th Karlsruhe Workshop on Software Radios*, 2010.

[75] O. Schliebusch, G. Ascheid, A. Wieferink, R. Leupers, and H. Meyr. Application Specific Processors for Flexible Receivers. In *Proc. of National Symposium of Radio Science (URSI)*, Poznan (Poland), apr 2005.

[76] S. Eberli. *Application-Specifc Processor for MIMO-OFDM Software-Defined Radio*. PhD thesis, ETH Zürich, 2009.

[77] M.A. Said, O.A. Nasr, and A.F. Shalash. Embedded Reconfigurable Synchronization & Acquisition ASIP for a Multi-Standard OFDM Receiver. *Eurasip Journal on Embedded Systems*, 2012, 2012. 10.1186/1687-3963-2012-2.

[78] O. Gustafsson, K. Amiri, D. Andersson, et al. Architectures for Cognitive Radio Testbeds and Demonstrators; An Overview. In *Proceedings of the Fifth International Conference on Cognitive Radio Oriented Wireless Networks Communications (CROWNCOM), 2010*, pages 1 –6, june 2010.

[79] STEricsson. Low-Power Embedded Vector DSP, EVP VD32041 32-bit Embedded-Vector Processor for SoCs.

[80] Texas Instruments Inc. *C6000 High Performance Multicore DSP*, 2011.

[81] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner. SODA: A High-Performance DSP Architecture for Software-Defined Radio. *Micro, IEEE*, 27(1):114 –123, jan.-feb. 2007.

[82] D. Novo, W. Moffat, V. Derudder, and B. Bougard. Mapping a Multiple Antenna SDM-OFDM Receiver on the ADRES Coarse-Grained Reconfigurable Processor. In *Signal Processing Systems Design and Implementation, 2005. IEEE Workshop on*, pages 473 – 478, nov. 2005.

[83] E. Matu, H. Seidel, T. Limberg, P. Robelly, and G. Fettweis. A GFLOPS Vector-DSP for Broadband Wireless Applications. In *Custom Integrated Circuits Conference, 2006. CICC '06. IEEE*, pages 543 –546, sept. 2006.

[84] C. Ebeling, C. Fisher, G. Xing, M. Shen, and H. Liu. Implementing an OFDM Receiver on the RaPiD Reconfigurable Architecture. *Computers, IEEE Transactions on*, 53(11):1436 – 1448, nov. 2004.

[85] http://www.tensilica.com.

[86] C. Rowen, P. Nuth, and S. Fiske. A DSP Architecture optimized for Wireless Baseband. In *System-on-Chip, 2009. SOC 2009. International Symposium on*, pages 151 –156, oct. 2009.

[87] D. Liu, A. Nilsson, E. Tell, D. Wu, and J. Eilert. Bridging Dream and Reality: Programmable Baseband Processors for Software-Defined Radio. *Communications Magazine, IEEE*, 47(9):134 –140, september 2009.

[88] E. Tell. *Design of Programmable Baseband Processors*. PhD thesis, Linköping University, 2005.

[89] A. Nilsson, E. Tell, and D. Liu. An 11mm^2 70mW Fully-Programmable Baseband Processor for Mobile WiMAX and DVB-T/H in 0.12μm CMOS. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 266 –612, feb. 2008.

[90] O. Schliebusch, H. Meyr, and R. Leupers. *Optimized ASIP Synthesis from Architecture Description Language Models*. Springer, Dordrecht, The Netherlands, 2007.

[91] A. Fauth, J. Van Praet, and M. Freericks. Describing Instruction Set Processors using nML. In *Proceedings of the 1995 European conference on Design and Test*, EDTC '95, pages 503–, Washington, DC, USA, 1995. IEEE Computer Society.

[92] G. Hadjiyiannis, S. Hanono, and S. Devadas. ISDL: an Instruction Set Description Language for Retargetability. In *Proceedings of the 34th annual Design Automation Conference*, DAC '97, pages 299–302, New York, NY, USA, 1997. ACM.

[93] P. Marwedel. The Mimola Design System: Tools for the Design of Digital Processors. In *Proceedings of the 21st Design Automation Conference*, DAC '84, pages 587–593, Piscataway, NJ, USA, 1984. IEEE Press.

[94] A. Halambi, P. Grun, V. Ganesh, and A. Khare. EXPRESSION: A Language for Architecture Exploration through Compiler/Simulator Retargetability. In *In Proceedings of the European Conference on Design, Automation and Test*, pages 485–490, 1999.

[95] A. Hoffmann, H. Meyr, and R. Leupers. *Architecture Exploration for Embedded Processors with LISA*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[96] A. Hoffmann, O. Schliebusch, A. Nohl, G. Braun, O. Wahlen, and H. Meyr. A Methodology for the Design of Application Specific Instruction set Processors (ASIP) using the Machine Description Language LISA. In *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, ICCAD '01, pages 625–630, Piscataway, NJ, USA, 2001. IEEE Press.

[97] P. Radosavljevic, J.R. Cavallaro, and A. de Baynast. ASIP Architecture Implementation of Channel Equalization Algorithms for MIMO Systems in WCDMA Downlink. In *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, volume 3, pages 1735 – 1739 Vol. 3, sept. 2004.

[98] M. Hamdy, O. A. Nasr, and A. F. Shalash. ASIP Design of a Reconfigurable Channel Estimator for OFDM Systems. In *International Conference on Microelectronics (ICM), 2011*, pages 1 –5, dec. 2011.

[99] T. Schuster, B. Bougard, P. Raghavan, R. Priewasser, D. Novo, L. Van der Perre, and F. Catthoor. Design of a Low Power Pre-Synchronization ASIP for Multimode SDR Terminals. In *Proceedings of the 7th international conference on Embedded computer systems: architectures, modeling, and simulation*, SAMOS'07, pages 322–332, Berlin, Heidelberg, 2007. Springer-Verlag.

[100] D. L. Iacono, J. Zory, E. Messina, N. Piazzese, G. Saia, and A. Bettinelli. ASIP Architecture for Multi-Standard Wireless Terminals. In *Proceedings of the conference on Design, automation and test in Europe: Designers' forum*, DATE '06, pages 118–123, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[101] C. Bachmann, A. Genser, J. Hulzink, M. Berekovic, and C. Steger. A Low-Power ASIP for IEEE 802.15.4a Ultra-Wideband Impulse Radio Baseband Processing. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 1614 –1619, april 2009.

[102] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, 4th edn.* Morgan Kaufmann, 2007.

[103] T. Bösch. *Adaptive Stream Processor for Network Multimedia Consumer Electronic Devices*. PhD thesis, ETH Zürich, 2004.

[104] W.A. Abu-Al-Saud and G.L. Stuber. Efficient Sample Rate Conversion for Software Radio Systems. In *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, volume 1, pages 559 – 563 vol.1, nov. 2002.

[105] T. Hentschel and G. Fettweis. Sample Rate Conversion for Software Radio. *Communications Magazine, IEEE*, 38(8):142 –150, aug 2000.

[106] Dallas Semiconductor. *Clock (CLK) Jitter and Phase Noise Conversion. Application Note AN-3359*.

[107] B. Brannon. *Samples Systems and the Effects of Clock Phase Noise and Jitter. Application Note AN-756*. Analog Devices.

[108] B. Brannon and A. Barlow. *Aperture Uncertainty and ADC System Performance. Application Note AN-501*. Analog Devices.

[109] Analog Devices. *High Resolution 6 GHz Fractional-N Frequency Synthesizer. ADF4157*.

[110] K. Gentile. *Direct Digital Synthesis (DDS) with a Programmable Modulus. Application Note AN-953*. Analog Devices.

[111] Analog Devices. *Low Power 250 MSPS 10-Bit DAC 1.8 V CMOS Direct Digital Synthesizer. AD9913*.

[112] Analog Devices. *A Technical Tutorial on Digital Signal Synthesis*.

[113] I. Lokken. *The Ups and Downs of Arbitrary Sample Rate Conversion*.

[114] T. Hentschel, M. Henker, and G. Fettweis. The Digital Front-End of Software Radio Terminals. *Personal Communications, IEEE*, 6(4):40 –46, aug 1999.

[115] T. Ramstad. Digital Methods for Conversion between Arbitrary Sampling Frequencies. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 32(3):577 – 591, jun 1984.

[116] R.E. Crochiere and L.R. Rabiner. Interpolation and Decimation of Digital Signals; A Tutorial Review. *Proceedings of the IEEE*, 69(3):300 – 331, march 1981.

[117] P.P. Vaidyanathan. Multirate Digital Filters, Filter Banks, Polyphase Networks, and Applications: a Tutorial. *Proceedings of the IEEE*, 78(1):56 –93, jan 1990.

[118] F.J. Harris, C. Dick, and M. Rice. Digital Receivers and Transmitters using Polyphase Filter Banks for Wireless Communications. *Microwave Theory and Techniques, IEEE Transactions on*, 51(4):1395 – 1412, apr 2003.

[119] R. Crochiere and L. Rabiner. Further Considerations in the Design of Decimators and Interpolators. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 24(4):296 – 311, aug 1976.

[120] H. G. Göckler. Polyphase Realization of Fractional Sample Rate Converters. In *ECCTD'99, Stresa, Italy*, volume 1, pages 405–408, 1999.

[121] J.G. Proakis and D.K. Manolakis. *Digital Signal Processing. Principles, Algorithms, and Applications*. Prentice Hall, 4th edition, 2007.

[122] T. Saramaki and T. Ritoniemi. An Efficient Approach for Conversion between Arbitrary Sampling Frequencies. In *Circuits and Systems, 1996. ISCAS '96., Connecting the World., 1996 IEEE International Symposium on*, volume 2, pages 285 –288 vol.2, may 1996.

[123] A.I. Russell and P.E. Beckmann. Efficient Arbitrary Sampling Rate Conversion with Recursive Calculation of Coefficients. *Signal Processing, IEEE Transactions on*, 50(4):854 –865, apr 2002.

[124] F.M. Gardner. Interpolation in Digital Modems. I. Fundamentals. *Communications, IEEE Transactions on*, 41(3):501 –507, mar 1993.

[125] L. Erup, F.M. Gardner, and R.A. Harris. Interpolation in Digital Modems. II. Implementation and Performance. *Communications, IEEE Transactions on*, 41(6):998 –1008, jun 1993.

[126] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck. *Discrete-time Signal Processing (2nd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[127] A. Shahed hagh ghadam and M. Renfors. Farrow Structure Interpolators based on Even Order Shaped Lagrange Polynomials. In *Image and Signal Processing and Analysis, 2003. ISPA 2003. Proceedings of the 3rd International Symposium on*, volume 2, pages 745 –748 Vol.2, sept. 2003.

[128] F. Harris. Performance and Design Considerations of the Farrow Filter when used for Arbitrary Resampling of Sampled Time Series. In *Conference Record of the Thirty-First Asilomar Conference on Signals, Systems amp; Computers, 1997*, volume 2, pages 1745 –1749 vol.2, nov. 1997.

[129] G.J. Cook, Y.H. Leung, and Y. Liu. On the Design of Variable Fractional Delay Filters with Laguerre and Kautz Filters. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003*, volume 6, pages VI – 281–4 vol.6, april 2003.

[130] L. Lundheim and T. Ramstad. An Efficient and Flexible Structure for Decimation and Sample Rate Adaptation in Software Radio Receivers. In *ACTS Mobile Communications Summit, Sorrento 1999*, 1999.

[131] F. Sheikh and S. Masud. Sample Rate Conversion Filter Design for Multi-Standard Software Radios. *Digit. Signal Process.*, 20(1):3–12, January 2010.

[132] E. Hogenauer. An Economical Class of Digital Filters for Decimation and Interpolation. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 29(2):155 – 162, apr 1981.

[133] T Wang and C. Li. Sample Rate Conversion Technology in Software Defined Radio. In *Canadian Conference on Electrical and Computer Engineering, 2006. CCECE '06*, pages 1355 –1358, may 2006.

[134] M. Henker, T. Hentschel, and G. Fettweis. Time-Variant CIC-Filters for Sample Rate Conversion with Arbitrary Rational Factors. In *The 6th IEEE International Conference on Electronics, Circuits and Systems, 1999. Proceedings of ICECS '99*, volume 1, pages 67 –70 vol.1, 1999.

[135] M. Unser. Splines: a perfect Fit for Signal and Image Processing. *Signal Processing Magazine, IEEE*, 16(6):22 –38, nov 1999.

[136] X. Huang. Arbitrary Ratio Sample Rate Conversion using B-Spline Interpolation for Software Defined Radio. In *Auswireless Conference*, 2006.

[137] J. Smith and P. Gossett. A Flexible Sampling-Rate Conversion Method. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '84.*, volume 9, pages 112 – 115, mar 1984.

[138] J. Smith. Bandlimited Interpolation Algorithm - Tutorial.

[139] A. Tkacenko. Variable Sample Rate Conversion Techniques for the Advance Receiver. IPN Progress Report 42-168.

[140] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.