MULTI-STANDARD ADAPTIVE WIRELESS COMMUNICATION RECEIVERS

ADAPTIVE APPLICATIONS MAPPED ON HETEROGENEOUS DYNAMICALLY RECONFIGURABLE HARDWARE

Gerard K. Rauwerda

Members of the dissertation committee:

prof. dr. ir.	G.J.M. Smit	University of Twente (promoter)
prof. dr. ir.	C.H. Slump	University of Twente
dr.	J.L. Hurink	University of Twente
prof. dr.-ing.	N. Wehn	Universität Kaiserslautern, Germany
prof. dr. ir.	T. Krol	University of Twente
prof. dr.	H. Corporaal	Technical University Eindhoven
prof. dr. ir.	A.J. Mouthaan	University of Twente (chairman and secretary)

Typeset with LaTeX.
Printed by Ipskamp PrintPartners, Enschede, The Netherlands.

ISBN 978-90-365-2607-4
ISSN 1381-3617 (CTIT PH.D.-THESIS SERIES NO. 08-109)

MULTI-STANDARD ADAPTIVE WIRELESS COMMUNICATION RECEIVERS

ADAPTIVE APPLICATIONS MAPPED ON HETEROGENEOUS DYNAMICALLY RECONFIGURABLE HARDWARE

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. W.H.M. Zijm,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 11 januari 2008 om 15.00 uur

door

Gerhardus Keimpe Rauwerda

geboren op 18 januari 1978
te Smallingerland

Dit proefschrift is goedgekeurd door:

prof. dr. ir. G.J.M. Smit (promotor)

Abstract

Today the world is overwhelmed with portable devices like handheld computers, mobile telephones and portable navigation systems. These devices will eventually be integrated into multi-functional devices that can perform all kinds of functions in one single system. Ultimately, portable devices will exist that can perform a wide variety of functions and provide rich information to the user.

One of the challenges for implementing these integrated multi-functional devices is to define a hardware architecture that is powerful enough to process complex algorithms, flexible enough to process all kinds of algorithms, and energy efficient enough because the portable devices are battery-powered.

This work focuses on the implementation of adaptive multi-standard multi-mode wireless communication systems that are implemented on dynamically reconfigurable hardware. The coarse-grained reconfigurable MONTIUM architecture is used to illustrate the mapping of wireless communication standards. Based on the mapping of different wireless communication algorithms, modifications of the MONTIUM architecture have been proposed.

Baseband processing and channel decoding of different wireless communication systems have been investigated. Orthogonal Frequency Division Multiplexing (OFDM) and Wideband CDMA (WCDMA) wireless communication techniques have been mapped on a dynamically reconfigurable System-on-Chip (SoC) architecture that contains several MONTIUM Tile Processors (TPs).

We showed that a single heterogeneous reconfigurable SoC platform can support various standards with a performance similar to an Application Specific Integrated Circuit (ASIC) implementation. The digital baseband processing of a HiperLAN/2 receiver and an Universal Mobile Telecommunications System (UMTS) receiver were mapped on MONTIUM TPs. Furthermore, the Viterbi and Turbo decoder algorithms were mapped on the same reconfigurable hardware.

All implementations of the baseband processing and channel decoding algorithms that are mapped on the MONTIUM TP have been verified against floating-point reference models. Performance simulations on the implemented algorithms show hardly any differ-

ence in accuracy between the floating-point reference models and the MONTIUM-based implementations.

As expected, an ASIC implementation of the algorithms is more energy efficient than an implementation in reconfigurable hardware. However, the ASIC implementation is fixed and the functionality of the ASIC cannot be changed. The power consumption and the configuration size of the MONTIUM TP depends on the implemented Digital Signal Processing (DSP) algorithms. The normalized dynamic power consumption of the MONTIUM-based Rake receiver in 0.13 μm CMOS technology is estimated at 0.470 mW/MHz. For the Viterbi decoder, the normalized dynamic power consumption is estimated at 0.309 mW/MHz on the same MONTIUM architecture.

The configuration sizes of the different DSP algorithms implemented on the MONTIUM TP are typically about 1 kB of configuration data. The MONTIUM TP can typically be configured as Rake or HiperLAN/2 receiver in less than 5 μs. In 7 μs the MONTIUM TP is configured as Viterbi or Turbo decoder. The characteristics of the algorithms can be semi-instantly changed through partial reconfiguration of the MONTIUM TP in the order of nanoseconds.

The small configuration sizes of the MONTIUM-based algorithms enable new opportunities for implementing real-adaptive applications. The standards level of adaptivity allows the terminal to adapt the communication standard that is used to satisfy the Quality of Service (QoS) requirements and the wireless channel conditions at a certain location. The algorithm-selection level of adaptivity allows the terminal to select the algorithms that satisfy the QoS requirements in the given communication environment in the most efficient manner. The algorithm-parameter level of adaptivity allows the terminal to change the parameters of a specific algorithm.

Exploiting the different kinds of adaptivity results in energy efficient wireless communication receivers that are capable of adapting the radio to the required QoS in different wireless communication environments. The flexibility, performance and low configuration overhead of the coarse-grained reconfigurable MONTIUM architecture enables the implementation of real-adaptive wireless communication receivers, which can switch their functionality instantly.

Samenvatting

Tegenwoordig wordt de wereld overspoeld met mobiele apparaten zoals zakcomputers, mobiele telefoons en draagbare navigatieapparatuur. Deze apparaten zullen uiteindelijk geïntegreerd worden tot een multi-functioneel apparaat, welke alle functies kan uitvoeren in een enkel systeem. Op den duur zullen deze mobiele apparaten iedere functie kunnen vervullen en de gebruiker van iedere gewenste informatie voorzien.

Eén van de uitdagingen om deze geïntegreerde multi-functionele apparaten te realiseren is het ontwikkelen van een hardware architectuur, die krachtig genoeg is om complexe algoritmes uit te voeren, die flexibel genoeg is om verschillende algoritmes uit te voeren, en die energie-efficient is aangezien draagbare apparaten gevoed worden met batterijen.

Het werk in dit proefschrift richt zich op de implementatie van adaptieve multi-standaard multi-mode draadloze communicatiesystemen, welke geïmplementeerd worden op dynamisch herconfigureerbare hardware. De grofkorrelig herconfigureerbare MONTIUM architectuur is gebruikt om verschillende draadloze communicatiestandaarden af te beelden. De afbeelding van verschillende draadloze communicatiealgoritmes op de MONTIUM architectuur heeft geleid tot voorstellen tot verbetering van deze architectuur.

Basisband signaalverwerking en kanaaldecodering van verschillende draadloze communicatiesystemen zijn onderwerp van onderzoek. Orthogonal Frequency Division Multiplexing (OFDM) en Wideband CDMA (WCDMA) draadloze communicatiealgoritmes zijn afgebeeld op een dynamisch herconfigureerbare Systeem-op-Chip (SoC) architectuur, welke meerdere MONTIUM Tegel Processors (TPs) bevat.

We hebben aangetoond dat een enkel heterogeen, herconfigureerbaar SoC platform gebruikt kan worden voor diverse standaarden met een prestatie welke gelijk is aan een Applicatie Specifieke geIntegreerde Chip (ASIC) oplossing. De digitale basisband signaalverwerking van een HiperLAN/2 ontvanger en een Universeel Mobiel Telecommunicatie Systeem (UMTS) ontvanger zijn afgebeeld op MONTIUM TPs. Tevens zijn de Viterbi en Turbo decoderingsalgoritmes afgebeeld op dezelfde herconfigureerbare hardware.

Alle implementaties van de basisband signaalverwerkings- en kanaaldecoderingsalgoritmes, welke afgebeeld zijn op de MONTIUM TP, zijn getest en vergeleken met drijvende komma referentie modellen. Simulaties met de geïmplementeerde algoritmes laten nauwelijks verschillen in de nauwkeurigheid van resultaten zien tussen de drijvende komma referentie modellen en de MONTIUM implementaties.

Zoals verwacht is een ASIC implementatie van de algoritmes energie-efficienter dan een implementatie gebaseerd op herconfigureerbare hardware. Echter, een ASIC implementatie is vast gedefinieerd en de functionaliteit kan niet gewijzigd worden. Het energieverbruik en de grootte van de configuratie van de MONTIUM TP hangen af van de geïmplementeerde digitale signaalverwerkingsalgoritmes. Het geschatte, genormalizeerde, dynamische vermogensverbruik van de Rake ontvanger, welke is afgebeeld op de MONTIUM in 0.13 μm CMOS technologie, bedraagt gemiddeld 0.470 mW/MHz. Voor de MONTIUM gebaseerde Viterbi decoder is het genormalizeerde, dynamische vermogensverbruik geschat op 0.309 mW/MHz voor dezelfde MONTIUM architectuur.

De grootte van de configuratie voor de verschillende algoritmes, welke zijn afgebeeld op de MONTIUM TP, bedraagt typisch ongeveer 1 kB. De MONTIUM TP kan geconfigureerd worden als Rake ontvanger of HiperLAN/2 ontvanger in minder dan 5 μs. De MONTIUM TP configureren als Viterbi of Turbo decoder kost 7 μs. Eigenschappen van de algoritmes kunnen gewijzigd worden in enkele nanoseconden door partieel herconfigureren van de MONTIUM TP.

De kleine groottes van de configuraties van MONTIUM gebaseerde algoritmes leiden tot nieuwe mogelijkheden om adaptieve toepassingen te realiseren die hun functionaliteit instantaan kunnen wijzigen. Adaptiviteit op het standaarden niveau maakt het mogelijk dat een mobiele ontvanger een communicatiestandaard kan kiezen om te voldoen aan de gevraagde kwaliteit van een verbinding (QoS) onder verschillende kanaalomstandigheden. Adaptiviteit op het niveau van algoritme-selectie stelt een mobiele ontvanger in staat om onder iedere kanaal-omstandigheid op de meeste efficiente manier te voldoen aan de gevraagde kwaliteit van een verbinding (QoS). Op algoritme-parameter niveau is het mogelijk om de parameters van een algoritme te wijzigen.

Door gebruik te maken van de verschillende vormen van adaptiviteit is het mogelijk een energie-efficiente draadloze communicatieontvanger te ontwikkelen welke de functionaliteit aanpast aan de gevraagde kwaliteit van de verbinding (QoS) onder verschillende kanaal-omstandigheden. De flexibiliteit, prestaties en kleine configuratieoverheadkosten van de grofkorrelig herconfigureerbare MONTIUM architectuur stellen ons in staat om adaptieve draadloze communicatieontvangers te ontwikkelen, welke de functionaliteit instantaan kunnen wijzigen.

Acknowledgements

Being a PhD student is always seen as a though and hard way of life. In fact, I encountered that being a PhD student is one big journey. Pursuing a PhD is setting on a journey through research, life and the world! I discovered what research is, discovered many nice places in the world during conference trips and I met nice people.

First of all, I am very grateful to Gerard Smit and Thijs Krol. Thijs and Gerard offered me the position to pursue my PhD. Gerard is always encouraging people and he always knows to highlight the positive aspects of experiments, even if the results are disappointing. I respect the way how Gerard manages his activities. Although he is occupied, he always manages to critically review papers and reports.

Paul Heysters and Lodewijk Smit were my first colleague PhD students who conducted their research in the same area as my research. Paul introduced me into the life of international conferences. We had a couple of good times in Las Vegas. Paul set the basis for the MONTIUM and he educated me many details of his architecture. Lodewijk always likes to ask critical questions. Our conference trip to Australia was a great experience. I am very grateful to the fact that Paul and Lodewijk founded (together with me) Recore Systems in 2005. Paul and Lodewijk gave me the freedom to finish my PhD during the establishment of Recore Systems, which I appreciate enormously.

My research was conducted in the AWgN project in close collaboration with people from the Electrical Engineering department of our university. I had a pleasant time working with Jordy Potman, Fokke Hoeksema and Kees Slump in the AWgN project. I am Jordy very grateful for providing his SASUMTSSIM simulator. The simulator made the verification of my Rake receiver implementation a lot easier. Furthermore, the bi-annual meetings with people from industry, the user-group, were always good events to acquire inspiration and get valuable reflections on our work.

I really liked the great atmosphere in the Computer Architecture for Embedded Systems (CAES) group. We had many coffee breaks and lunch walks that generated many new ideas and interesting discussions. During the last years many faces of students and employees have appeared in the CAES group. Therefore, it is impossible to thank everyone individually.

In 2004 I enjoyed the working environment of the Atmel Design Center in Ulm, Germany. I would like to thank Werner Brugger for giving me that opportunity to stay for 5 months in Ulm. Matthias Hofinger and Pieter Maier showed me the typical habits in Ulm such as *Nabada*. During Nabada all inhabitants of Ulm seem to sail around or swim in the Danube.

I still appreciate the special friendship with my old school friends from Hallum. Although we spread all over The Netherlands, we still are in contact with each other. I would like to thank my friends from Hallum for their interest and support. It is always nice to know who are your real friends. I met many people during the time being a PhD student. It is impossible to thank everyone personally for their support in one or another way. Thank you all for your interest, motivation and support. It is a good feeling to know that I can always count on you! Of course, special thanks go out to Robin de Graaf and Jacob Hulder for being my paranimphs.

Summer 2006 was very sunny. Especially, my heart was full of sunshine. Since then I really know what *love* is. I would like to thank Marloes Baijens for her love. She motivated me and kept me on track during the last year of writing this thesis. Finally, I would like to thank my parents, Kees Rauwerda and Wieke Vellinga, and my brother, Peter Rauwerda, for their love and continuous interest in my work. They always supported me during my studies and in all decisions I made.

Gerard Rauwerda
Enschede, December 2007

Table of Contents

List of Figures

List of Tables

List of Acronyms

2G	Second Generation
3G	Third Generation
4G	Fourth Generation
4S	SMART CHIPS FOR SMART SURROUNDINGS
AAC	Advanced Audio Coding
ACH	Access feedback Channel
ACS	Add Compare Select
ADC	Analog-to-Digital Converter
AE	Array Element
AFC	Automatic Frequency Control
AGU	Address Generation Unit
AHB	Advanced High-performance Bus
ALU	Arithmetic Logic Unit
AM	Amplitude Modulation
AMBA	Advanced Microcontroller Bus Architecture
ARM	Advanced RISC Machines
ASIC	Application Specific Integrated Circuit
AWgN	ADAPTIVE WIRELESS NETWORKING
AWGN	Additive White Gaussian Noise
BCH	Broadcast Channel
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
CAES	Computer Architecture for Embedded Systems
CCI	Communication and Control Interface

CCU	Communication and Configuration Unit
CD	Compact Disc
CDL	Configuration Description Language
CDMA	Code Division Multiple Access
CIF	Common Interleaved Frame
CIR	Channel Impulse Response
CM	Configuration Manager
CMOS	Complementary Metal Oxide Semiconductor
CPCH	Common Packet Channel
CPICH	Common Pilot Channel
CU	Capacity Unit
DAB	Digital Audio Broadcasting
DAC	Digital-to-Analog Converter
DCH	Dedicated Channel
DDC	Digital Down Converter
DECT	Digital Enhanced Cordless Telecommunications
DFT	Discrete Fourier Transform
DiL	direct link[1]
DL	downlink[2]
DLP	Data-Level Parallelism
DMA	Direct Memory Access
DPCCH	Dedicated Physical Control Channel
DPCH	Dedicated Physical Channel
DPDCH	Dedicated Physical Data Channel
D-QPSK	Differential QPSK
DRM	Digital Radio Mondiale
DS-CDMA	Direct Sequence CDMA
DSCH	Downlink Shared Channel
DSP	Digital Signal Processing / Processor
DVB	Digital Video Broadcasting
DVFS	Dynamic Voltage and Frequency Scaling
EEP	Equal Error Protection
EMC	Electromagnetic Compatibility
ETSI	European Telecommunications Standards Institute

[1] The DiL is a connection between two mobile terminals without using an intermediate base station.
[2] The DL direction is from base station to mobile receiver.

FAC	Fast Access Channel
FACH	Forward Access Channel
FAU	Function Accelerator Unit
FCH	Frame Channel
FDD	Frequency Division Duplex
FDMA	Frequency Division Multiple Access
FEC	Forward Error Correction
FER	Frame Error Rate
FFT	Fast Fourier Transform
FIB	Fast Information Block
FIC	Fast Information Channel
FIR	Finite Impulse Response
FM	Frequency Modulation
FNC	Function
FPGA	Field Programmable Gate Array
FU	function unit
GHz	Gigahertz
GPP	General Purpose Processor
GPS	Global Positioning System
HE-AAC	High Efficiency AAC
HiperLAN/2	High Performance Radio LAN type 2
HR	Hardware Radio
HS-DSCH	High Speed Downlink Shared Channel
HS-PDSCH	High Speed Physical Downlink Shared Channel
Hz	Hertz
I	In-phase
IBOC	In-Band On-Channel
IC	Integrated Circuit
ICI	intercarrier interference
IEEE	Institute of Electrical and Electronics Engineers
IF	Intermediate Frequency
iFFT	inverse FFT
ILP	Instruction-Level Parallelism
I/O	Input/Output
IP	Intellectual Property
I/Q	In-phase/Quadrature

IRQ	Interrupt Request
ISI	intersymbol interference
ISR	Ideal Software Radio
ITRS	International Technology Roadmap for Semiconductors
kB	Kilobyte
kbps	Kilobits per second
kHz	Kilohertz
ksps	kilosymbols per second
LAN	Local Area Network
LCH	Long transport Channel
LLR	Log-Likelihood Ratio
LO	local oscillator
LOS	Line-of-sight
LTE	Long Term Evolution
LUT	Look-up Table
LW	long wave
MAC	Medium Access Control
MAC	Multiply-Accumulate
MAI	Multiple Access Interference
MAN	Metropolitan Area Network
MAP	maximum a posteriori
Mbps	Megabits per second
MBps	Megabytes per second
MC-CDMA	Multi-Carrier CDMA
MCI	Multiplex Configuration Information
Mcps	Megachips per second
MHz	Megahertz
MLC	multi-level coding
MLM	Max-Log-MAP
MMSE	Minimum Mean Squared Error
MP2	MPEG-1 Layer 2
MPEG	Moving Pictures Expert Group
MRC	Maximal Ratio Combining
MSB	Most Significant Bit
MSC	Main Service Channel
MSD	multistage decoder

MSPS	Megasamples per second
MW	medium wave
NML	Native Mapping Language
NoC	Network-on-Chip
NSC	Non Systematic Convolutional
OFDM	Orthogonal Frequency Division Multiplexing
OS	Operating System
PAC	Processing Array Cluster
PAE	Processing Array Element
PCH	Paging Channel
PER	Packet Error Rate
PFA	Prime Factor Algorithm
PP	Processing Part
PPA	Processing Part Array
Q	Quadrature
QAM	Quadrature Amplitude Modulation
QoS	Quality of Service
QPSK	Quadrature Phase Shift Keying
RACH	Random Access Channel
RAM	Random Access Memory
RCH	Random Channel
RE	Register Exchange
RF	Radio Frequency
RISC	Reduced Instruction Set Computer
RSC	Recursive Systematic Convolutional
RTL	Register Transfer Level
SaS	Signals and Systems
SB	status bit
SC-FDMA	Single-Carrier FDMA
SCH	Synchronisation Channel
SCH	Short transport Channel
SCR	Software Controlled Radio
SDC	Service Description Channel
SDR	Software Defined Radio
SDRAM	Synchronous Dynamic RAM
SF	spreading factor

SIMD	Single Instruction Multiple Data
SISO	Soft-Input-Soft-Output
SNR	Signal-to-Noise Ratio
SoC	System-on-Chip
SRAM	Static RAM
SW	short wave
TB	Traceback
TDD	Time Division Duplex
TDM	Time Division Multiplexing
TDMA	Time Division Multiple Access
TFCI	Transport Format Combination Indicator
TLP	Thread-Level Parallelism
TP	Tile Processor
TPC	Transmit Power Control
TSMC	Taiwan Semiconductor Manufacturing Company
TTI	Transmission Time Interval
UEP	Unequal Error Protection
UL	uplink[3]
UMTS	Universal Mobile Telecommunications System
USR	Ultimate Software Radio
VHDL	VHSIC Hardware Description Language
VHF	Very High Frequency
VHSIC	Very High Speed Integrated Circuit
VLIW	Very Large Instruction Word
WCDMA	Wideband CDMA
WLAN	Wireless LAN
WMAN	Wireless MAN
XOR	Exclusive OR
XPP	EXTREME PROCESSING PLATFORM
ZF	zero-forcing

[3] The UL direction is from mobile receiver to base station.

Chapter 1

Introduction

Wireless communication systems are progressing towards multi-standard multi-mode communication systems. These communication systems are capable of dynamically adapting to various conditions while achieving optimal performance. The focus of the work in this thesis is on the implementation of adaptive multi-standard multi-mode wireless communication systems on heterogeneous reconfigurable hardware.

This chapter introduces the ADAPTIVE WIRELESS NETWORKING *(AWgN) project and outlines the work described in this thesis. The objectives of this work are given. Furthermore, the motivation for performing research towards adaptive multi-standard multi-mode wireless communication receivers is described.*

1.1 Introduction

This thesis addresses fundamental issues in the architecture and design of adaptive wireless communication systems. The research presented in this thesis has been conducted within the ADAPTIVE WIRELESS NETWORKING (AWgN) project[1]. Section 1.2 outlines the AWgN project and describes the focus areas of this thesis.

Section 1.3 describes the context of the AWgN project and companion projects. The research problem within the AWgN context is given in Section 1.3.1. The main contributions of this work to the field of reconfigurable computing and Software Defined Radio (SDR) are summarized in Section 1.3.2. Finally, the structure of the thesis is given in Section 1.3.3.

1.2 ADAPTIVE WIRELESS NETWORKING (AWgN)

The AWgN project aims to develop methods and technologies that can be used to design efficient, adaptable and reconfigurable base stations and terminals for Third Generation (3G) and Fourth Generation (4G) wireless communication systems, as for example Universal Mobile Telecommunications System (UMTS). To achieve this, the project consists of two activities:

- *Adaptive DSP algorithms*
  The goal of the *Adaptive DSP algorithms for UMTS* activity of the AWgN project is to deliver a set of adaptive Digital Signal Processing (DSP) algorithms that can be used in UMTS communication systems; [P2, P18]

- *Mapping DSP algorithms*
  The goal of the *Mapping DSP algorithms to a reconfigurable architecture* activity of the AWgN project is to map a set of adaptive DSP algorithms, for multi-mode wireless communication systems, on a heterogeneous reconfigurable architecture. The reconfigurable architecture is heterogeneous in the sense that signal processing is performed in General Purpose Processors (GPPs), in bit-level reconfigurable hardware, in word-level reconfigurable hardware or in Application Specific Integrated Circuit (ASIC) parts.

The work described in this thesis comprises the *Mapping DSP algorithms* activity of the AWgN project, which is carried out in the *Computer Architecture for Embedded Systems* (CAES) group[2] within the Computer Science department of the University of Twente. The work in the *Adaptive DSP algorithms* activity is carried out in the *Signals and Systems* (SaS) group[3] within the Electrical Engineering department of the University of Twente.

[1] The AWgN project (TTC.5956) has been supported by the Freeband Knowledge Impulse programme, a joint initiative of the Dutch Ministry of Economic Affairs, knowledge institutions and industry.

[2] `http://caes.cs.utwente.nl/`

[3] `http://www.sas.el.utwente.nl/`

Objectives

In the *Mapping DSP algorithms* activity of the AWgN project we focus on the downlink (DL) receiver of wireless communication systems. The main emphasis is on mapping DSP algorithms on a heterogeneous reconfigurable System-on-Chip (SoC) architecture. The following objectives are identified:

1. Study the state-of-the-art in baseband processing algorithms for Orthogonal Frequency Division Multiplexing (OFDM) and Wideband CDMA (WCDMA) wireless communication systems in order to define a set of algorithms that are typical for 3G / 4G multi-standard wireless communication systems;
   This objective is covered by Chapter 2, 4, 5 and 6.

2. Show that the set of typical wireless communication algorithms can be mapped efficiently on a heterogeneous reconfigurable SoC architecture;
   This objective is covered by Chapter 4, 5 and 6.

3. Identify opportunities for exploiting adaptivity in wireless communication receivers. Investigate how adaptive features of the wireless communication receiver can be used to efficiently implement the algorithms on the heterogeneous reconfigurable SoC architecture;
   This objective is covered by Chapter 4, 5, 6 and 7.

4. Define specifications for a SoC architecture for implementation of adaptive multi-standard multi-mode wireless communication devices.
   This objective is covered by Chapter 7.

Motivation

Evolution of technology (e.g. in DSPs and reconfigurable computing) will allow to move digitization closer to Radio Frequency (RF) in the radio access network. Furthermore, Software Defined Radio (SDR) receivers appear which are no longer implemented in dedicated ASICs, but are built as a programmable solution. SDR techniques will evolve with availability of low-cost enabling components. SDR has a very high potential to enable a major leap to obtain faster provision of more flexible, advanced mobile communication services.

Multi-standard network elements in the radio access network combined with advanced network management will allow dynamically assigned services, will provide Quality of Service (QoS) support for user applications, will use the spectrum and network resources more efficiently and will push integration of services and networks in the global, business and domestic environment. SDR may change the scope and role of standardization, allowing more freedom for implementation of new services (see Chapter 2).

For industry active in this field the main driving forces for using Software Defined Radios are:

- *cost reduction*: avoiding costly redesigns of ASICs;
- *flexibility*: using the same hardware for different traffic patterns and new (or revised) standards;
- *time-to-market*: reusing already designed hardware architectures (Intellectual Property (IP) reuse) reduces the design time.

It is expected that the combination of high-level design tools and reconfigurable hardware architectures will enable designers to develop highly flexible, efficient and adaptive base stations and wireless communication terminals. Performance and power gains will be achieved by applying dynamically reconfigurable hardware architectures instead of programmable GPP architectures. In this approach ASIC design is replaced by dynamic reconfiguration and reprogramming. The concepts of reconfigurable hardware architectures are described in Chapter 3.

The technological challenges to realize SDR for wireless communication systems are enormous:

- Highly *efficient* system architectures have to be developed that are *scalable*, *flexible* and *adaptive* to applications with varying processing capacity requirements (QoS);
- The communication between the various processing entities has to be *high performance*, *flexible* and *low power*;
- The middleware and (distributed) Operating Systems (OSs) have to support real-time requirements and (distributed) multi-tasking capability with minor overhead;
- The design tools will have to deal with the heterogeneity of the underlying hardware architecture, and the *adaptivity* of applications;
- Mapping of algorithms on the hardware architecture has to be done carefully, as this is closely related to the *efficiency* of the system.

Adaptivity

A key issue of systems that have to support (mobile) wireless communication systems is that they have to be adaptive. These systems have to adapt to:

- *changing environmental conditions*
  e.g. changing number of users in a cell or varying Signal-to-Noise Ratio (SNR) due to signal reflections and user movements;
- *changing user demands*
  e.g. the user might request for a service with better quality or a service with higher throughput (QoS);

- *changing / evolving standards*
  e.g. wireless communication standards evolve quickly. Systems have to be capable to adjust their functions with adapted or additional features.

Furthermore, these systems have to be extremely *efficient* when these are used in battery-operated terminals and *cost-effective* as these are used in consumer products as well as in base stations. Although *energy efficiency* is a major issue in mobile wireless communication terminals because they draw their energy from small batteries, energy consumption is also an issue in base stations from a technical (e.g. costly cooling of chips and power supplies) as well as from an environmental and operational point of view (e.g. environmental pollution and energy costs).

Reconfigurability

There are quite a number of good reasons for using reconfigurable hardware architectures in future wireless communication terminals and base stations:

- Although reconfigurable systems are known to be less power efficient compared to ASIC implementations they can have considerable benefits. Wireless communication systems work in a very dynamic environment, this means that depending on the distance between the receiver and transmitter or cell occupation more or less processing power is needed. When the system can adapt to the environment – at run-time – significant efficiency can be gained;

- Wireless communication standards evolve quickly; this means that systems need the flexibility and adaptivity to adapt to slight changes in the standards. By using reconfigurable hardware architectures instead of ASICs costly redesigns can be avoided;

- 3G and 4G communication systems based on e.g. the UMTS standard have a QoS based transmission scheme. These communication systems are highly flexible and adaptable to services and quality which is required by the user;

- 4G communication systems shift toward multi-standard communication systems, which have different communication standards integrated in one device;

- Reconfiguration of hardware enables the implementation of new or adapted services on existing terminals (hardware reuse by *long-term reconfiguration*);

- Traditional DSP algorithms are rather static. The recent emergence of new applications that require sophisticated adaptive, dynamic algorithms based on signal and channel statistics to extract optimum performance has drawn renewed attention to run-time reconfigurability (*short-term reconfiguration*).

1.3 This thesis

The work described in this thesis mainly focuses on the mapping of SDR applications and DSP algorithms on a heterogeneous reconfigurable SoC architecture with coarse-grained reconfigurable hardware elements. All work has been conducted as part of the *Mapping DSP algorithms* activity of the AWgN project.

1.3.1 Problem statement

The central problem addressed in this thesis is the implementation of adaptive multi-standard multi-mode wireless communication systems on heterogeneous reconfigurable System-on-Chip (SoC) architectures. The MONTIUM architecture is used to illustrate the feasibility of mapping Software Defined Radio (SDR) applications on coarse-grained reconfigurable hardware.

1.3.2 Contribution

The scope of the work presented is given by the AWgN project. The work resulted in several publications in international conferences, books and journals. All publications are listed on page 219. Contributions are made in different areas of reconfigurable computing and Software Defined Radio:

- We identified important DSP algorithms in wireless communication systems that are based on Orthogonal Frequency Division Multiplexing (OFDM) and Wideband CDMA (WCDMA) techniques. These investigations establish the requirements for implementing multi-standard multi-mode wireless communication receivers in heterogeneous reconfigurable System-on-Chips (SoCs);
  Chapter 4, 5 and 6, Publication [P18, P20]

- We showed that a single SoC platform can support various standards with a performance and energy consumption similar to an ASIC implementation:

  1. The digital baseband processing of the HiperLAN/2 receiver has been mapped on the MONTIUM TPs as an illustration for OFDM wireless communication receivers;
     Chapter 4, Publication [P6, P20]
  2. The Rake receiver has been mapped on the MONTIUM TP to illustrate the mapping of WCDMA wireless communication receivers;
     Chapter 5, Publication [P9, P20]
  3. The Viterbi and Turbo decoder have been mapped on the MONTIUM TP in order to complement the baseband processing algorithms;
     Chapter 6, Publications [P12, P15, P20]

- We modified the Arithmetic Logic Units (ALUs) of the MONTIUM Tile Processor (TP) by adding Add Compare Select (ACS) support;
  Chapter 6, Publication [P12]

- We showed that a coarse-grained reconfigurable architecture with partial reconfiguration capability can be used efficiently for adapting receiver settings dynamically and we showed that energy can be saved by adaptivity;
  Chapter 5, Publication [P9, P20]

- We developed a prototype implementation of a heterogeneous reconfigurable System-on-Chip (SoC). The SoC contains four coarse-grained reconfigurable MONTIUM Tile Processors (TPs) and one General Purpose Processor (GPP). The prototype chip is an instant of the CHAMELEON SoC template, intended to be used for digital broadcasting receivers (e.g. DAB).
  Chapter 7

1.3.3 Thesis outline

This chapter introduced the subject of this thesis in the scope of the AWgN project. The next chapters discuss different issues concerning SDR and the mapping of DSP algorithms on reconfigurable hardware. Every chapter starts with an introduction and is concluded with a summary and discussion.

Chapter 2 Chapter 2 discusses the context of the AWgN project with respect to Software Defined Radio (SDR). The basic framework of SDR is introduced and trends in wireless communication systems are investigated. Especially, all levels of adaptivity in SDR systems are discussed in the perspective of the AWgN project.

Chapter 3 Chapter 3 discusses various hardware architectures to implement SDR applications. The heterogeneous reconfigurable CHAMELEON SoC template is introduced as well as the coarse-grained reconfigurable MONTIUM TP. In this thesis the MONTIUM TP is used to illustrate the mapping of DSP algorithms on coarse-grained reconfigurable hardware.

Chapter 4 Chapter 4 focuses on OFDM-based wireless communication systems. The implementation of OFDM techniques is analyzed for Wireless LAN and digital broadcasting standards (i.e. HiperLAN/2, DAB and DRM). The principles of OFDM are explained in the context of different wireless communication standards.

The baseband processing functions of the HiperLAN/2 receiver have been mapped on multiple MONTIUM TPs, which are part of the CHAMELEON SoC template. The mapping of the HiperLAN/2 receiver on the MONTIUM TP has been verified by exhaustive BER performance simulations. Similar mappings have been proposed for the DAB and DRM receiver.

Chapter 5 WCDMA-based communication systems are the topic in Chapter 5. The general framework of Code Division Multiple Access (CDMA) communication systems is introduced, based on the UMTS standard. The mapping of the Rake receiver, which is an important DSP kernel in WCDMA receivers, is discussed in detail and mapped on the MONTIUM TP. The integration of the MONTIUM-based Rake receiver in the CHAMELEON SoC template is explained.

Simulations verify the BER performance of the Rake receiver under different conditions. The energy consumption of the MONTIUM TP performing the Rake functions has been estimated by simulation and compared to alternative ASIC Rake implementations.

Chapter 6 Channel decoding is the topic of Chapter 6. The Viterbi and Turbo channel decoding algorithms have been studied because they typically appear in a multi-standard wireless communication receiver. The Viterbi and Turbo decoder have been mapped on the MONTIUM TP and verified against reference implementations.

Hardware modifications of the MONTIUM have been proposed to support Add Compare Select (ACS) operations, which are typically used in channel decoding algorithms.

Chapter 7 Chapter 7 addresses the design of a heterogeneous reconfigurable System-on-Chip (SoC). A prototype SoC has been designed, which is an instant of the CHAMELEON SoC.

Chapter 8 Chapter 8 concludes this thesis with final conclusions.

Chapter 2

Software Defined Radio

This chapter introduces the Software (Defined) Radio concept. Many radio functions are migrating from the analog domain to the digital domain nowadays. The digitization of wireless communication receivers is accompanied with the establishment of Software (Defined) Radios.

Software (Defined) Radios introduce additional adaptivity features in wireless communication receivers. Levels of adaptivity can be identified in wireless communication standards, wireless communication functions and in wireless communication algorithms. Exploiting the different kinds of adaptivity results in energy efficient wireless communication receivers that are capable of adapting the radio to the required Quality of Service in different wireless communication environments.

Parts of this chapter have been published as a book chapter [P18].

2.1 Introduction

In this thesis wireless communication receivers are proposed according to the Software (Defined) Radio concept. This chapter introduces the concepts of Software Radio in wireless communication systems (Section 2.2).

The digitization of wireless communication systems goes hand in hand with the establishment of Software Radios. The impact of Software (Defined) Radio on wireless communication systems is analyzed in Section 2.3. The trends of multi-standard multi-mode communication systems and the large number of emerging and evolving wireless communication standards are addressed.

In Section 2.4 the energy budget of wireless receivers is addressed. Implementation of Software Defined Radio (SDR) requires a flexible, but energy efficient hardware architecture. Adaptivity is the key challenge in Fourth Generation (4G) wireless communication systems to reduce power consumption of the systems. Section 2.5 discusses the different types of adaptivity that can be identified in wireless communication systems.

The key elements of this chapter are summarized in Section 2.6.

2.2 Software Defined Radio

Nowadays wireless communication systems are identified as Software (Defined) Radios, since most radio functions make a shift from the analog to the digital domain. Moreover, the digitization of wireless communication systems is accompanied by the implementation of radio functions in software.

Joseph Mitola was one of the early pioneers investigating the opportunities of employing Software (Defined) Radios [75, 76]. He is considered the godfather of Software Defined Radio (SDR) and he also established the concept of Cognitive Radio [77]. According to Mitola: *"A software radio is a radio whose channel modulation waveforms are defined in software. That is, waveforms are generated as sampled digital signals, converted from digital to analog via a wideband Digital-to-Analog Converter (DAC) and then possibly up-converted from Intermediate Frequency (IF) to Radio Frequency (RF). The receiver, similarly, employs a wideband Analog-to-Digital Converter (ADC) that captures all of the channels of the software radio node. The receiver then extracts, down-converts and demodulates the channel waveform using software on a general purpose processor."*

Many definitions of Software Radio exist in the area of wireless communication systems. The SDR Forum identified different levels of flexibility in radio architectures [98]:

Level 0 *Hardware Radio* (HR)
The radio is implemented using hardware components only and cannot be modified except through physical intervention;

Level 1 *Software Controlled Radio* (SCR)
Only the control functions of an SCR are implemented in software – thus only limited functions are changeable using software. Typically this extends to inter-

connects, power levels, etc. but not to frequency bands and / or modulation types, etc;

Level 2 *Software Defined Radio* (SDR)
SDRs provide software control of a variety of modulation techniques, wide-band or narrow-band operation, communications security functions (such as hopping), and waveform requirements of current and evolving standards over a broad frequency range. The frequency bands covered may still be constrained at the front-end requiring a switch in the antenna system;

Level 3 *Ideal Software Radio* (ISR)
ISRs provide dramatic improvement over an SDR by eliminating the analog amplification or heterodyne mixing prior to analog-digital / digital-analog conversion. Programmability extends to the entire system with analog-digital / digital-analog conversion only at the antenna, speaker and microphones;

Level 4 *Ultimate Software Radio* (USR)
USRs are defined for comparison purposes only. It accepts fully programmable traffic and control information and supports a broad range of frequencies, air-interfaces & applications software. It can switch from one air interface format to another in milliseconds, use GPS to track the users location, store money using smartcard technology, or provide video so that the user can watch a local broadcast station or receive a satellite transmission.

SCR, SDR, ISR and USR are jointly referred to as *Software Radio*. Related with the development of Software Defined Radio (SDR) is the emerging interest in Cognitive Radio. A Cognitive Radio is a radio or system that is aware of its operational environment and can be trained to dynamically and autonomously adjust its radio operating parameters accordingly [77]. However, *cognitive* does not necessarily imply relying on software, e.g. cordless (DECT) telephones have long been able to select the best authorized channel based on relative channel availability.

Software Defined Radio (SDR) denotes wireless communication systems that are characterized by an analog front-end followed by a programmable, digital baseband processing part. The SDR concept for wireless communication receivers as used throughout this thesis is depicted in Figure 2.1.

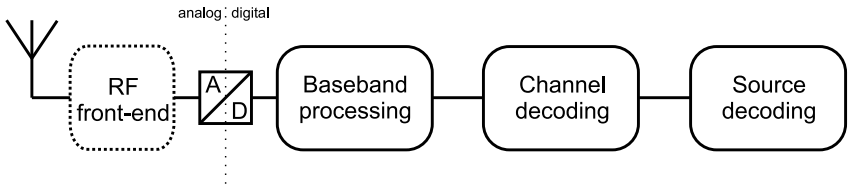

Figure 2.1: *Software Defined Radio (SDR) receiver architecture.*

In the analog front-end the radio signal is received, frequency converted, filtered and amplified. The filtered, amplified radio signal is converted to digital samples, which are the input of the digital baseband processing part. A programmable, digital baseband processing part enables reprogramming of functional modules that have to be performed, like modulation / demodulation techniques.

With the advancing developments in Software Radio, more functions in the analog front-end are replaced by digital equivalents. Hence, the Analog-to-Digital Converter (ADC) in the wireless communication receiver is shifting forward to the RF antenna with advancing levels of flexibility in the SDR Forum context; in Ideal Software Radio (ISR) and Ultimate Software Radio (USR) receivers, the ADC is directly placed at the RF antenna. Moving the ADC towards RF implies that a large dynamic resolution and large signal bandwidth are required [110]. Current developments in ADCs show that SDR receivers are feasible with acceptable power dissipation. The International Technology Roadmap for Semiconductors (ITRS) 2005 section on *System Drivers* depicts the recent ADC performance needs for important product classes and signals important research challenges in ADC design [63]. Improvements in ADC technology with respect to e.g. power consumption and performance are clearly not adequate for ISR and USR. The overall improvement in ADC technology is about 1.5 bits per 8 years for the same power consumption level and with equal sampling rate [114]. For ISR / USR receivers the Signal-to-Noise Ratio (SNR) as well as the sampling rate is insufficient. Hence, SDR receivers are the only feasible concept with reasonable power consumption nowadays.

2.2.1 Benefits of Software Defined Radio

A complete hardware based radio system (e.g. an ASIC solution) has limited functionality since parameters for each of the functional modules are fixed. A radio system built using SDR technology extends the utility of the system to a wide range of applications using different link-layer protocols and modulation / demodulation techniques. SDR provides an efficient and relatively inexpensive solution to the design of multi-mode, multi-band, multi-functional wireless devices that can be enhanced using software upgrades.

SDR-enabled devices (i.e. mobile terminals) can be dynamically programmed to reconfigure the characteristics of the device. So, the same hardware can be adapted to perform different functions at different times.

Another advantage of the SDR template is that real-adaptive systems can be implemented. Traditional algorithms in wireless communications are rather static. The recent emergence of new applications that require sophisticated adaptive, dynamic algorithms based on signal and channel statistics to achieve optimum performance has drawn renewed attention to run-time reconfigurability [87].

The benefits of SDR are addressed for different target groups [111]:

- *Users*
  - Enhanced roaming capabilities without changing the terminal;
  - Seamless interoperability between many different communication standards;

  - Optimized radio transmission characteristics according to the environment;
  - Download of protocol / standard extensions.

- *Network operators / Service providers*
  - Gradual and flexible upgrade of services (e.g. from 2G to 3G and 4G);
  - Improved Quality of Service (QoS): Radio transmission characteristics are optimized according to the environmental conditions and traffic demands as well as to the service;
  - Increased traffic and revenue: Not a number of hardware platforms but just one technology platform is able to deal with multiple radio access technologies;
  - Early market presence: Terminals will be able to incorporate new features dynamically as service technology continues to evolve.

- *Manufacturers / Industry*
  - Fast time-to-market: Reusing already designed hardware architectures;
  - Low development costs: Avoiding costly redesigns of Application Specific Integrated Circuits (ASICs);
  - Flexibility: SDR allows decoupling of service provision from standardization processes. Hence, there is no need to wait for finalized standards, but standards may evolve during product development.

2.2.2 Disadvantages of Software Defined Radio

Although complete hardware based radio systems have limited functionality because parameters for each of the functional modules are fixed, hardware based systems give advantages over SDR systems. Implementation of SDR systems requires flexible programmable hardware, which is generally less efficient than a dedicated fixed hardware solution (e.g. an ASIC). So, the high flexibility in SDR systems is disadvantageous for the power consumption of wireless communication systems.

In Chapter 3 different hardware architectures are investigated that can be used to implement SDR systems. The hardware architectures are classified in different categories ranging from fixed hardware to fully flexible hardware architectures.

2.3 Trends in wireless communication

There is a clear trend in wireless communication systems that many new digital wireless standards appear [105, 110] and, moreover, wireless communication standards evolve quickly.

The following types of wireless communication systems can be identified: *single-carrier*, *multi-carrier* and *spread spectrum* systems. The recent emerging new communication standards for Third Generation (3G) and Fourth Generation (4G) are all based on Orthogonal Frequency Division Multiplexing (OFDM) or Code Division Multiple Access (CDMA) techniques. Developments for 4G wireless communication systems show a shift towards the integration of OFDM and CDMA technologies in wireless communication systems.

Universal Mobile Telecommunications System (UMTS) Long Term Evolution (LTE) has been proposed as new 4G development, which incorporates the 3G UMTS standard and is extended with downlink (DL) communication channels based on OFDM [34]. In current 3G systems link adaptation has been achieved by exploiting channel variations in time, which provides an increase in spectral efficiency. Through the use of OFDM in the DL communication channels, link adaptation in the frequency domain can be exploited as well. Frequency domain adaptation becomes important with larger application bandwidth requirements. Using e.g. water-filling techniques, sub-channels with a low interference levels (i.e. high SNR) get more signal energy allocated than those with high interference levels (i.e. low SNR) [43].

Single-carrier techniques are proposed for high data rate uplink (UL) communication channels. Single-Carrier FDMA (SC-FDMA) is investigated for the UL communication channels in UMTS LTE and Wireless MAN (WMAN) communication systems [40, 80].

Other developments towards 4G wireless communication systems are the integration of CDMA and OFDM techniques in the physical layer. CDMA systems are migrating to Multi-Carrier CDMA (MC-CDMA) systems, which is typically an OFDM system with additional CDMA overlay [49]. The CDMA overlay provides a multiple access scheme to OFDM-based communication systems.

Hence, multiple transmission techniques are incorporated in the physical layer of emerging 4G wireless communication standards. Furthermore, existing wireless communication standards and digital broadcasting standards are integrated in 4G compatible communication devices, yielding multi-standard radios [81]. The abundance of standards incorporated in the 4G wireless communication system enables the system to efficiently operate in any situation by selecting the appropriate standard.

General characteristics of emerging and evolving 3G / 4G wireless communication standards are:

- Increasing signal bandwidth;
- Increasing data throughput;
- More complex functions / algorithms;
- More robust error protection;
- Parameterizable, flexible functions;
- Link adaptation facilities.

2.4 Efficient baseband processing and channel decoding

Figures presented in [19, 70] show that error decoding in a wireless (OFDM) receiver is as computationally intensive as baseband processing. This means that one should consider optimized implementations of both baseband processing and error correction algorithms for multi-mode communication systems in Software Radios.

Figure 2.2 depicts the typical energy consumption of a Wireless LAN (WLAN) OFDM receiver, as presented in [19]. The digital processing part of a typical SDR receiver is on average responsible for 60% of the total power budget. In [70] it has been reported that 50% or more of the computational complexity in the digital processing part of the wireless receiver is due to error correction algorithms, like Viterbi decoding. This complexity counts for about 60% of the total energy consumption in the digital processing part of a typical wireless receiver.

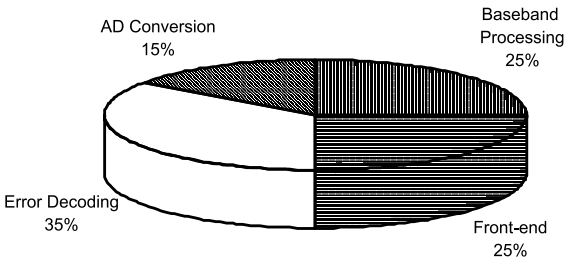

Figure 2.2: *Energy consumption in typical WLAN OFDM receiver [19].*

Developments in the analog front-end and ADC design show slow progress with respect to power consumption [114]. Consequently, effective power reduction should be achieved in the digital processing part of wireless communication receivers. So, new hardware oriented design methodologies are proposed in ASIC design for e.g. Viterbi decoders [45] and Turbo decoders [69].

Power reduction by exploiting variations in system characteristics due to changing noise conditions are the focus in [53, 65]. The latter can be achieved using dynamic reconfiguration in reconfigurable hardware. Applying these measures in coarse-grained reconfigurable hardware, i.e. the MONTIUM Tile Processor (TP), is discussed in Chapter 6 for channel decoding algorithms.

Power reduction in the baseband processing part of the wireless communication receiver can be achieved by adapting parameters of the baseband algorithms. Furthermore, the implementation of baseband processing functions can be adapted by changing to alternative algorithms. Adaptive implementations of baseband processing functions in OFDM and WCDMA receivers are the focus of Chapter 4 and 5.

The integration of multiple standards in wireless communication receivers and the abundance of emerging wireless communication standards requires flexible hardware architectures. Traditional SDR approaches are implemented on *homogeneous* flexible architectures, e.g. General Purpose Processor (GPP) or Digital Signal Processor (DSP) [96].

Since baseband processing in the wireless receiver is computationally intensive, the terminal's hardware architecture has to be very powerful. Moreover, as wireless terminals are battery-powered, the importance of energy efficiency of the hardware architecture is emphasized. A common objection to the traditional homogeneous flexible architecture is its relative energy inefficiency. However, *heterogeneous* reconfigurable hardware, consisting of processing elements with *different* granularities, is designed with flexibility, performance and energy efficiency in mind. Heterogeneous reconfigurable architectures are introduced in Chapter 3.

2.5 Adaptivity

Trends in wireless communication systems show many opportunities for exploiting adaptivity in Digital Signal Processing (DSP) algorithms of wireless communication systems. Three levels of adaptivity can be identified: *standards level*, *algorithm-selection level* and *algorithm-parameter level*. The Software Radio concept is a key enabler to efficiently employ the different levels of adaptivity.

Standards level The support of multiple wireless communication standards introduces a first level of adaptivity in the wireless terminal because the terminal can switch between wireless communication standards. For example when packet data transport is performed over UMTS and a WLAN hotspot becomes available, the terminal can switch from UMTS to a WLAN standard. This is referred to as *standards level* adaptivity. Standards level adaptivity has an impact on the digital signal processing in the wireless terminal because the wireless communication standard defines the DSP functions that have to be performed to implement the standard.

Algorithm-selection level Although a wireless communication standard usually defines the DSP functions which have to be performed to implement the standard, it usually does not define the algorithms that have to be used to implement these functions. The communication system can therefore select the best algorithm from a set of algorithms that implement the same DSP function. Therefore, this second level of adaptivity is referred to as *algorithm-selection level* adaptivity.

Figure 2.3 shows the *algorithm-selection level* adaptivity applied in a wireless terminal. Assume that a certain function in a UMTS terminal requires equal link quality independent of the number of interfering users. This function can be performed by different DSP algorithms, which require different processing power levels while supplying equal link quality levels under different conditions, e.g. different Multiple Access Interference (MAI) levels:

1. *Algorithm 1* requires least processing power and can efficiently be used to perform the function in case a few interfering users are present in a cell;

2. The conditions of the wireless channel become worse when the MAI due to interfering users increases. Hence, the terminal receiver needs to spend more effort to

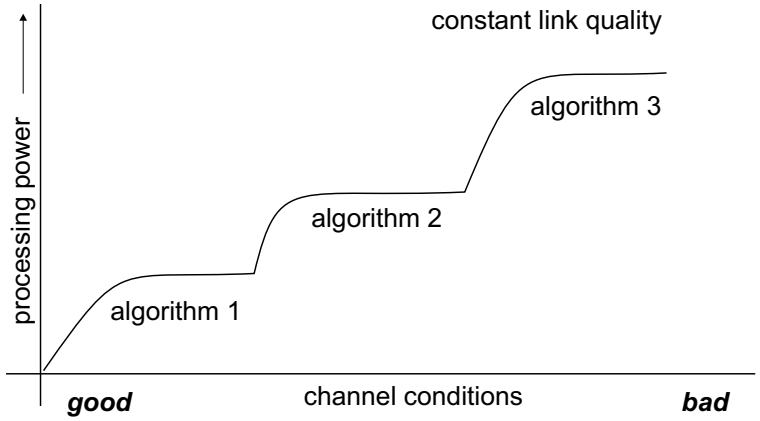

Figure 2.3: *Algorithm-selection – Processing power / channel conditions trade-off.*

supply the same link quality level. However, the link quality level cannot be met by *algorithm 1* any more and the receiver should switch to *algorithm 2* which performs more complex receiver algorithms. The complex algorithm requires more processing power;

3. When the channel conditions become even worse, the receiver needs to switch to the most sophisticated algorithm. *Algorithm 3* needs the most processing power to fulfill the quality requirements of the supplied wireless link.

Traditional wireless communication receivers in the mobile terminal are typically designed for worst-case conditions. Therefore, *algorithm 3* would always be implemented in a Hardware Radio (HR) terminal to satisfy the link quality level required. Since the conditions of the wireless channel are typically better than the worst-case scenario, dramatic power reductions can be achieved by applying more power efficient DSP algorithms.

Algorithm-parameter level For a specific algorithm, there are also opportunities for adaptivity by changing parameters of the algorithm. This third level of adaptivity is called *algorithm-parameter level* adaptivity.

An example of algorithm-parameter level adaptivity is given by the amount of iterations of a Turbo decoder, which depends on the SNR of a given channel, or the amount of Rake fingers used to receive the desired signal, which depends on the number of signal reflections in a mobile communication environment [102].

2.6 Summary

This chapter introduced different aspects of Software (Defined) Radio. The digitization of wireless communication systems enables the implementation of most radio functions in software, resulting in Software Radios. Different levels of Software Radios can be identified, ranging from Software Controlled Radio (SCR) to Ideal Software Radio (ISR). In SCR only the control functions of the wireless communication receiver are implemented in software. Whereas, in ISR programmability extends to the entire wireless communication receiver with the Analog-to-Digital Converter (ADC) and Digital-to-Analog Converter (DAC) directly applied at the RF antenna.

Trends identified in 3G and 4G wireless communication systems are the fast appearance of new wireless communication standards as well as the quick evolution of existing wireless communication standards. The multitude of 3G / 4G wireless communication standards results in the concept of multi-standard multi-mode wireless communication devices. Moreover, 3G and 4G wireless communication systems are provided with adaptive functions. The abundance of wireless communication standards and the adaptivity features result in wireless communications terminals that can adapt to Quality of Service (QoS) requirements and to the wireless channel conditions at a certain location.

An adaptive wireless terminal only makes sense when it better suits the needs of a user at an acceptable complexity and efficiency compared to a traditional non-adaptive terminal. The standards level of adaptivity allows the terminal to adapt the communication standard that is used to satisfy the QoS requirements and the wireless channel conditions at a certain location. The algorithm-selection level of adaptivity allows the terminal to select the algorithms that satisfy the QoS requirements in the given communication environment in the most efficient manner. The algorithm-parameter level of adaptivity allows the terminal to change the parameters of a specific algorithm. So, a reconfigurable and adaptive terminal is able to track the QoS requirements and communication environment on a much finer grained scale than a traditional non-adaptive terminal.

Chapter 3

Hardware Architectures for Software Defined Radio Receivers

Trends in semiconductor technology and issues related to hardware architectures are investigated in this chapter. Hardware architectures are characterized with respect to performance, flexibility and energy efficiency.

Coarse-grained reconfigurable hardware architectures are discussed. Furthermore, the design methodologies for implementing Digital Signal Processing (DSP) functions in Software Defined Radio (SDR) applications are analyzed.

Special focus is on the coarse-grained reconfigurable MONTIUM *Tile Processor (TP). The* MONTIUM *TP has been applied in a tiled System-on-Chip (SoC) as target architecture for mapping multi-standard adaptive wireless communication receivers.*

3.1 Introduction

In the previous chapter the trends in Software Defined Radio (SDR) have been investigated. In this chapter the hardware architectures that are used to implement the digital signal processing of SDR are discussed.

Section 3.2 analyzes the trends in semiconductor technology. In hardware architectures a trade-off between flexibility, performance and energy efficiency can be distinguished. Flexible hardware architectures typically achieve bad performance and are energy inefficient. Best performance and best energy efficiency is obtained in dedicated hardware designs. The different hardware architectures are characterized in Section 3.3.

A tiled System-on-Chip (SoC) template is discussed in Section 3.4. The SoC typically contains heterogeneous reconfigurable processing tiles. The MONTIUM Tile Processor (TP) that is integrated in the tiled SoC is applied in this thesis as the target architecture for mapping multi-standard adaptive wireless communication receivers.

Section 3.5 discusses various coarse-grained reconfigurable architectures that target Digital Signal Processing (DSP) applications in the wireless communication and multimedia domain. Special attention is given to the coarse-grained reconfigurable MONTIUM TP in Section 3.5.1.

Section 3.6 summarizes the main contribution of this chapter.

3.2 Trends in semiconductor technology

In [78], Gordon Moore looked ahead into the future of semiconductor technology. This investigation became the basis of the well-known *Moore's law*. Initially Moore analyzed the trend in number of components per Integrated Circuit (IC). This analysis finally resulted in *Moore's law*, predicting that the number of transistors for minimum transistor costs on an IC doubles every year. *Moore's law* was reformulated in 1975, stating that the transistor density of ICs doubles every 24 months [79]. Nowadays, *Moore's law* is known in its present form that computing performance doubles every 18 months.

Originally *Moore's law* was intended to be an observation that made a forecast on the future of semiconductor technology. However, *Moore's law* became more accepted with the years and has become the main motivation in semiconductor industry to double the transistor density every 18 months. In this sense, *Moore's law* has become a self-fulfilling prophecy.

With the scaling of semiconductor technology, the transistor density of ICs increases. Hence, present designs that are implemented in semiconductor technology will become smaller and require less silicon area. Reducing the feature size[1] in semiconductor technology gives opportunities for implementing more complex designs, since the transistor budget is increased.

[1] The feature size denotes the smallest detail that can be produced in silicon.

Another advantage of decreasing feature size is an increasing clock frequency of the IC designs, yielding more processing power[2]. On the other hand the semiconductor industry faces power problems due to the technology scaling. The increase in transistor density has also an effect on the power density. Moreover, an increase in clock frequency induces an increase in power consumption.

The problem of power dissipation is faced by the semiconductor industry and mitigated by e.g. introducing parallelism in hardware designs. Parallel hardware architectures and multi-core architectures are designed to cope with the *memory wall*, *power wall* and *Instruction-Level Parallelism (ILP) wall* in current hardware designs [52].

The *memory wall* states that there exists a gap between the memory bandwidth and the data path bandwidth (i.e. compute bandwidth). Therefore, locality of reference is heavily applied in multi-core architectures. Many small local memories in the processors of multi-core architectures, which run individually at low clock frequency, reduce the gap between memory bandwidth and compute bandwidth. Moreover, the multi-core approach increases the total memory bandwidth of the entire System-on-Chip (SoC). The *ILP wall* points out that there exists only limited ILP in applications. By increasing the granularity of parallelism, e.g. by introducing multi-core architectures, parallelism can be exploited at a higher level in the application (e.g. Thread-Level Parallelism (TLP)). The lower clock frequencies obtained by multi-core architectures will reduce the dynamic power consumption as well, reducing the *power wall*.

To demolish the walls, multi-core hardware architectures spread the processing load over multiple cores which perform their operations concurrently. Consequently, similar overall performance, or even higher, can be achieved at lower clock frequencies.

The semiconductor business faces also a productivity gap; the complexity of hardware designs overruns the productivity of hardware designers. Intellectual Property (IP) reuse is a solution that helps in mitigating the productivity gap. More complex hardware designs can be built in shorter time by reusing IP hardware building blocks. IP reuse can increase the productivity by 200% [33]. Hence, IP reuse reduces the product design time and shortens the time-to-market.

Moreover, the increasing complexity of hardware designs leads to problems with respect to the testability of ICs at design time [63]. IP reuse is a compromise by partitioning the overall hardware design in smaller less complex hardware IP building blocks. Testing the hardware design of smaller hardware IP building blocks requires less effort and is less costly. Hardware IP building blocks only have to be tested once, before they can be reused. As a consequence the costs of testing the overall hardware design are reduced dramatically by reusing hardware IP.

[2] Processor performance is not only determined by clock speed, but by the combination of frequency and the amount of operations / instructions executed per clock cycle. This misunderstanding is commonly referred to as the *Megahertz Myth*.

3.3 Hardware architectures

Different hardware architectures are available in the embedded systems domain to perform Digital Signal Processing (DSP) functions and algorithms: *General Purpose Processors (GPPs)*, *Digital Signal Processors (DSPs)*, *(re-)configurable hardware* and *application specific hardware*. The application specific hardware is designed for a dedicated function and is usually referred to as Application Specific Integrated Circuit (ASIC). The ASIC is, like its name suggests, an application specific processor which has been implemented in an IC.

These hardware architectures have different characteristics in relation to *performance*, *flexibility* or *programmability* and *energy efficiency* [54]. Figure 3.1 depicts the trade-off in flexibility and performance for different hardware architectures. Generally, more flexibility implies a less energy efficient solution. A classification of different hardware architectures that are used to implement wireless communication receivers is shown in Figure 3.2. The most important characteristics of the hardware architectures are discussed in the following sections.

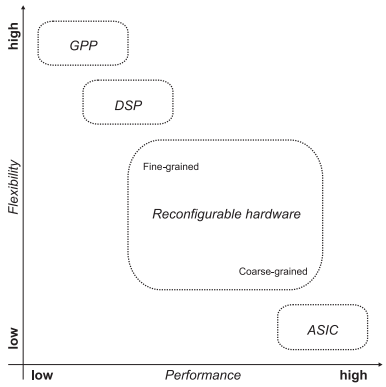

Figure 3.1: *Flexibility versus performance trade-off for different hardware architectures.*

3.3.1 General Purpose Processor

The GPP is the most flexible hardware architecture. The processor can be programmed to perform almost any algorithm. Although the GPP can perform almost any algorithm, it cannot perform every algorithm efficiently. GPPs are well suited for control-oriented functions. Due to the large overhead in control, these processors are not very energy efficient.

Basically, two kinds of GPP architectures exist which differ in their internal memory organization:

- The *Von Neumann architecture* is characterized by the fact that both instructions and data are located together in the same local memory of the processor. The

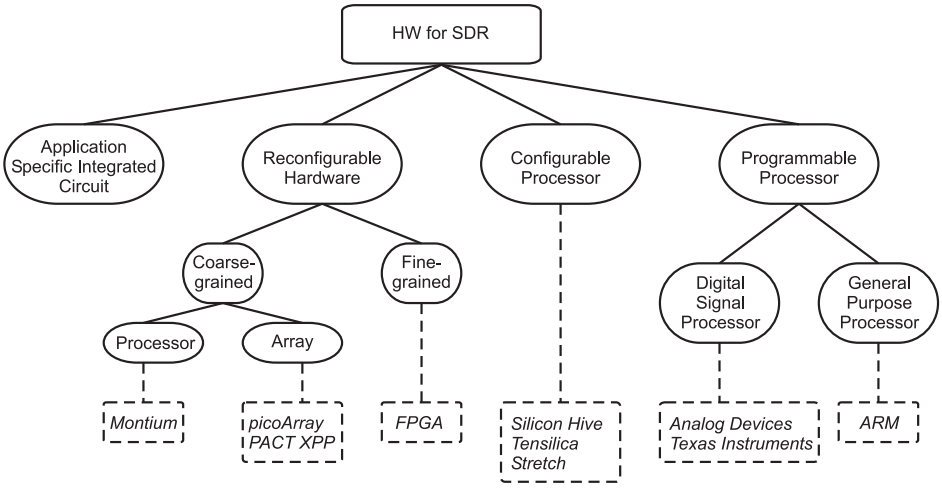

Figure 3.2: *Classification of hardware architectures for Software Defined Radio.*

disadvantage of this architecture is that data and instructions cannot be fetched concurrently, but have to be scheduled in time. The data throughput between the memory and the processor is limited in the *Von Neumann architecture*, which is referred to as the *Von Neumann bottleneck*;

- The *Harvard architecture* has separated the memory for instructions and data. The separation of data and instruction memories allows parallel fetch of instructions and data. Furthermore, pipelining can be implemented easier with separated data and instructions memories. This architecture is better suitable to build high performance processors.

Caches are used when higher clock speeds for the GPP are required. Caches are placed between the data / instruction memory and the data / instruction processing unit, to bridge the gap in clock speed of the processing unit and the memory[3].

The Arithmetic Logic Unit (ALU) inside the GPP is flexible and supports many simple operations. The ALU generally does not contain any specialized dedicated functionality. Complex operations, e.g. *Multiply-Accumulate (MAC)* operations, are performed on the GPP in an inefficient way, as the complex operation has to be composed from many simple instructions, e.g. *add* and *multiply* instructions.

[3] Usually clock speeds in the data path of a processor are higher than the clock speeds of the memory architecture. *Moore's law* typically applies for the speed of the data path and the capacity of memories, but the speed of memories increases only by 7% annually [52, 121].

3.3.2 Digital Signal Processor

A DSP is a processor that is specifically designed to perform DSP operations. DSPs can be programmed to perform general purpose operations, but these are performed inefficiently. The DSP is related to the GPP, but implemented with dedicated hardware to support functions that appear in typical DSP algorithms. Typical functions are *Multiply-Accumulate (MAC)*, *modulo* and *bit reverse* operations. Characteristics of DSPs are:

- Designed for real-time signal processing;
- Designed according to the *Harvard architecture*;
- Special instructions for parallel operations, e.g. SIMD, VLIW;
- Pipelined architecture;
- Parallel multiplier and accumulator;
- Single cycle operations;
- Dedicated memory address calculation unit;
- Fixed-point arithmetic with support for saturation arithmetic;
- Special memory structures to fetch multiple data items or instructions;
- Direct Memory Access (DMA) support.

In comparison with GPPs DSPs are optimized for high speed processing by taking advantage of parallelism in DSP applications. Therefore, the DSP hardware architecture has been optimized to exploit Data-Level Parallelism (DLP) and Instruction-Level Parallelism (ILP), which is supported by special Single Instruction Multiple Data (SIMD) and Very Large Instruction Word (VLIW) instructions[4].

DSPs achieve better performance than GPPs. However, both processor types have large overhead in control, as every instruction needs to be fetched from memory. This overhead does not benefit the energy efficiency of DSPs.

3.3.3 Configurable processor

Configurable processors are a new kind of flexible processors which integrate flexible logic in the GPP architecture. The configurable processor maintains similar ease of application development as GPPs, but achieves better performance due to embedded flexible logic inside the data path of the processing unit. Manufacturers of configurable processors are e.g. Silicon Hive, Stretch and Tensilica [100, 104, 108].

Configurable processors are typically developed in synergy with the intended application. Once the application is known, the application is analyzed and computationally

[4] The general characteristics of DSPs are described. Therefore, not all instructions are supported in any DSP.

intensive kernels are identified. Based on the computationally intensive kernels, an instant of the configurable processor is synthesized at design time which has an optimized instruction set for those kernels. In fact configurable processors are configurable at design time and only programmable at run-time. Hence, an optimized instruction set has been defined at design time.

The optimized instructions in the configurable processor are mainly obtained by implementing specialized hardware inside the processing unit of the processor architecture. The specialized hardware can increase the performance of the processor dramatically for a particular application domain. At design time hardware / software co-designers have to make a trade-off between flexibility and performance; processors can be designed that are only dedicated for one dedicated application by just removing all general purpose instructions from the instruction set and only keep dedicated instructions.

Different directions are investigated in creating configurable processors: Tensilica provides configurable Reduced Instruction Set Computer (RISC) processors which can be synthesized with an optimized instruction set at design time. Moreover, the software-configurable processors of Stretch are based on configurable Tensilica RISC processors. However, Stretch has extended the instruction set of the configurable processor with configurable logic. These processors are always configured with a default general purpose instruction set. The configurable processors of Silicon Hive, however, are fully stripped at design time and only incorporate the essential instructions. Hence, synthesized Silicon Hive processors appear to be more ASIC than GPP.

Configurable processors obtain much better performance than GPPs. Since these processors are derived from GPP architecture templates, they still suffer from substantial control overhead.

3.3.4 Reconfigurable hardware

Whereas the instruction set of configurable processors is defined at design time, reconfigurable hardware can be customized to the instantaneous needs of an application at run-time. One characteristic of reconfigurable hardware is the ability to change the logic inside the data path of the hardware. The main motivation for reconfigurable hardware is that the hardware adapts to the needs of a specific application or algorithm instead of adapting the algorithm to the hardware.

The main difference between configurable processors and reconfigurable hardware is that configurable processors are designed for a given application. The configurable processor has a fixed, optimized instruction set for specific DSP functions. Reconfigurable hardware on the other hand defines a hardware template which can be configured for different applications within an application domain. Reconfigurable hardware is very flexible in adapting the data path to a desired function, and the configurable instruction set can (mostly) instantly be (re-)configured. In fact, the instruction set of reconfigurable hardware is (re-)configured at run-time. This means that only a small subset of all possible instructions is active after the hardware is configured. Thus, only the necessary logic and interconnects inside the data path of the hardware are configured.

Different levels of flexibility are identified in reconfigurable hardware [109]. The granularity of reconfigurable hardware defines the size of the smallest functional block that can be configured for a specific operation. Reconfigurable hardware becomes more flexible as the detail of configurability increases, i.e. finer grained. However, fine-grained reconfigurable hardware involves larger configuration overhead. Hence, reconfigurable hardware involves a flexibility / costs trade-off. Increasing the flexibility results in e.g. increased power consumption, configuration overhead and chip area.

Fine-grained Fine-grained reconfigurable devices are bit-level programmable. Because of the configurability at bit-level, the configuration overhead is large. Fine-grained reconfigurable devices are perfectly suited for prototyping and to implement encryption algorithms. Typical examples of fine-grained reconfigurable hardware are Field Programmable Gate Arrays (FPGAs). However, modern FPGA devices have already embedded coarse-grained processing elements for better efficiency [7, 122].

Coarse-grained Coarse-grained reconfigurable devices are reconfigurable at word-level. Multipliers, adders, etc. are hardwired in these devices. Because only coarse functional blocks have to be configured, the configuration overhead is small. These architectures are more suited for data-oriented functions, like algorithms performed in the DSP domain.

Because of the introduction of coarse functional processing blocks, the flexibility in coarse-grained reconfigurable hardware is reduced. Therefore, coarse-grained reconfigurable hardware is generally optimized for a particular domain of DSP applications and is therefore referred to as *domain-specific* reconfigurable hardware. Coarse-grained reconfigurable architectures are promising to implement Software Defined Radio (SDR) applications [103].

Coarse-grained reconfigurable hardware implicitly incorporates code compaction in contrast to fine-grained reconfigurable hardware. Lower configuration overhead results in faster configuration of the reconfigurable hardware, giving new opportunities for implementing real-adaptive applications that can change their functions instantly. Hence, coarse-grained reconfigurable hardware enables new directions in system design of e.g. SDR applications by exploiting dynamic reconfiguration.

3.3.5 Application Specific Integrated Circuit

Application Specific Integrated Circuits (ASICs) represent the most energy efficient hardware architecture. An ASIC incorporates a dedicated hardware design that is customized for a specific function or application.

The dedicated hardware comprises only fixed circuitry, which results in limited or even no flexibility in a given application. By designing only fixed functions in the ASIC no additional complex circuitry has to be implemented for flexibility purposes. Because of the limited flexibility, the dedicated functions can be implemented efficiently in the ASIC.

The energy efficiency of ASIC designs is advantageous, the inflexibility is disadvantageous. Applications in wireless communication systems embrace more and more dynamic requirements as discussed in Section 2.3. Wireless communication standards are evolving quickly, and they also require flexibility to instantly adapt to the communication environment. Hence, flexibility is already required when a communication standard is still in the definition phase and also when the application is employed. Once an ASIC has been designed, the specifications of the dedicated application cannot be changed anymore. Evolving standards can result in different specifications making an ASIC design valueless.

3.4 Heterogeneous reconfigurable System-on-Chip

In [54] the CHAMELEON System-on-Chip (SoC) template has been proposed, which contains processing tiles of different kind. Figure 3.3 depicts the proposed tiled SoC template with heterogeneous processing elements.

The idea of heterogeneous processing elements is that one can match the granularity of the algorithms with the granularity of the hardware. Four processor types can be distinguished: *General Purpose Processor*, *fine-grained* reconfigurable hardware (e.g. FPGA), *coarse-grained* reconfigurable hardware (e.g. MONTIUM Tile Processor) and *dedicated* hardware (e.g. ASIC). Matching the granularity of the reconfigurable hardware with the algorithm provides flexibility at the right level.

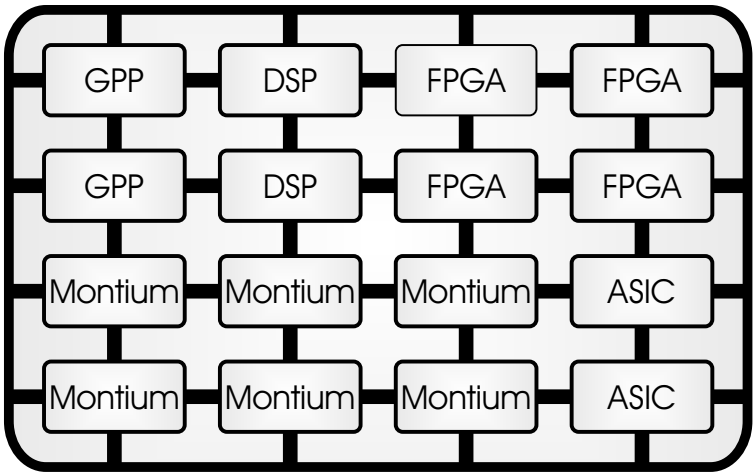

Figure 3.3: *The* CHAMELEON *SoC template.*

The different tile processors in the SoC are interconnected by a Network-on-Chip (NoC). Both SoC and NoC are dynamically reconfigurable, which means that the pro-

grams running on the processing tiles as well as the communication channels are defined at run-time[5].

The configuration of the processing tiles and the configuration of the NoC is coordinated by a special *coordination function*. This *coordination function* can be implemented on a GPP tile, which is programmed with a run-time Operating System (OS). The run-time OS schedules the DSP tasks at run-time on the heterogeneous reconfigurable SoC.

The advantages of designing SoCs according to a tiled approach are [54]:

- Tiles of the same type can be duplicated when the number of transistors grow in newer CMOS technology;
- Replications of tiles eases the verification process;
- Tiles do not grow in complexity with new CMOS technology;
- Relatively small tiles allow extensive optimizations;
- Locality of reference is exploited;
- Computational performance scales about linearly with the number of tiles;
- Unused tiles can be switched off to reduce energy consumption of the entire SoC;
- Individual clock frequency per tile and individual power supply voltage per tile enable Dynamic Voltage and Frequency Scaling (DVFS) .

Explicit parallelism By employing the tiled SoC approach, as proposed in Figure 3.3, many kinds of parallelism are explicitly applied:

- *Thread-Level Parallelism (TLP)* is addressed by the multi-core approach as different tiles can run different tasks;
- *Data-Level Parallelism (DLP)* is achieved by the MONTIUM processing tiles, which employ parallelism in the data path;
- *Instruction-Level Parallelism (ILP)* is addressed by the MONTIUM processing tiles as multiple data path instructions can be executed concurrently.

Dynamic Voltage and Frequency Scaling DVFS is an important measure to control the power consumption of embedded systems. In CMOS design, the power consumption, P, is given by:

$$P \propto V^2 \cdot f \tag{3.1}$$

[5] The CHAMELEON SoC template that contains coarse-grained reconfigurable MONTIUM processing tiles is used as target architecture for mapping multi-standard adaptive wireless communication receivers in Chapter 4, 5 and 6 of this thesis.

The power consumption in CMOS depends quadratically on the supply voltage, V. The main idea of DVFS is that the supply voltage should be kept as low as possible. Besides, the maximum operating frequency, f, is tightly coupled to the supply voltage level. This means that by scaling the clock frequency of hardware, the supply voltage can be scaled as well, resulting in a cubic decrease of the power consumption. The energy consumption, on the other hand, decreases quadratically because the time to complete an operation increases due to the reduced clock frequency.

3.5 Coarse-grained reconfigurable architectures

Several coarse-grained reconfigurable architectures have been developed and designed by both academia and industry. In the following sections coarse-grained reconfigurable architectures intended for Digital Signal Processing in Software Defined Radio (SDR) are discussed. For a more extensive overview of coarse-grained reconfigurable architectures we refer to [51, 103, 109].

The MONTIUM Tile Processor is used in this thesis as the target architecture for our experiments on mapping multi-standard adaptive wireless communication receivers. Other recent coarse-grained reconfigurable processors that are discussed and that target the SDR wireless communication domain are the PICOARRAY, the EXTREME PROCESSING PLATFORM (XPP) and the Silicon Hive processor template.

3.5.1 MONTIUM Tile Processor

The MONTIUM is an example of a coarse-grained reconfigurable processor and targets the 16-bit Digital Signal Processing (DSP) algorithm domain. The MONTIUM architecture origins from research at the University of Twente [54][6]. The coarse-grained reconfigurable processor has been further developed by Recore Systems [93].

Architecture

A single MONTIUM processing tile is depicted in Figure 3.4. At first glance the MONTIUM architecture bears a resemblance to a VLIW processor. However, the control structure of the MONTIUM is very different. The lower part of Figure 3.4 shows the Communication and Configuration Unit (CCU) and the upper part shows the coarse-grained reconfigurable MONTIUM Tile Processor (TP).

Communication and Configuration Unit The CCU implements the interface for off-tile communication. The definition of the off-tile communication interface depends on

[6] The work described in Chapter 4, 5 and 6 of this thesis is based on the MONTIUM architecture that is described in [54]. The MONTIUM architecture has been further developed and optimized, however, the basic principles of the architecture as described in [54] remain valid. The future SoC in Chapter 7 has been built with an optimized MONTIUM TP architecture [93].

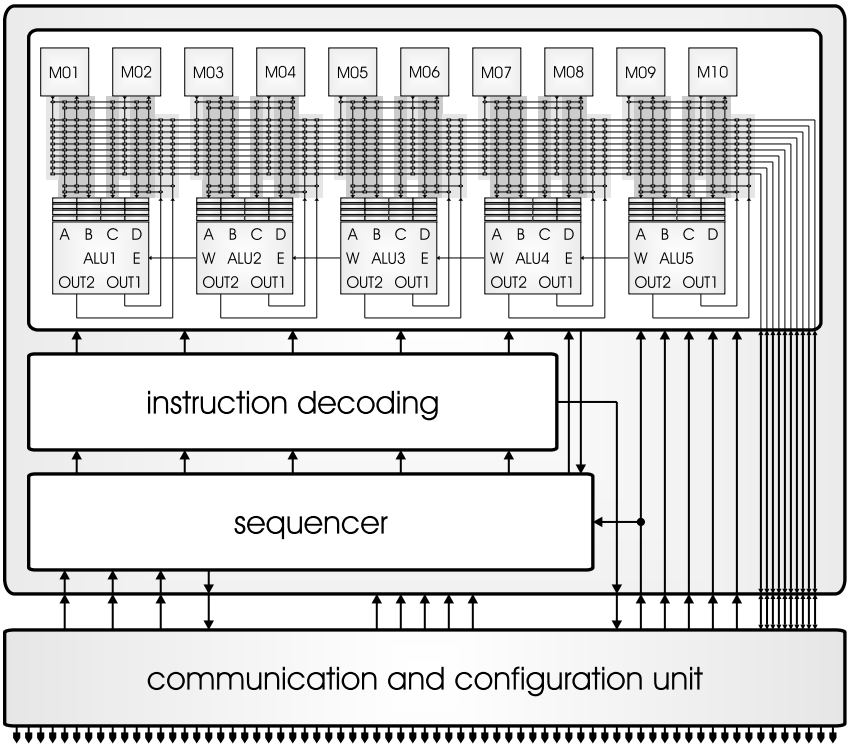

Figure 3.4: *The* MONTIUM *coarse-grained reconfigurable processing tile.*

the NoC technology that is used in a SoC in which the MONTIUM processing tile is integrated [20].

The CCU enables the MONTIUM TP to run in *streaming* as well as in *block* mode. In *streaming* mode the CCU and the MONTIUM TP run in parallel. Hence, communication and computation overlap in time in *streaming* mode. In *block* mode the CCU first reads a block of data, then starts the MONTIUM TP, and finally after completion of the MONTIUM TP the CCU sends the results to the next processing unit in the SoC (e.g. another MONTIUM processing tile or external memory).

The CCU implements the network interface controller between the NoC and the MONTIUM TP. The CCU provides configuration and communication services to the MONTIUM TP:

- *Configuration* of the MONTIUM TP and parts of the CCU itself;
- *Block communication* to move data into or from the MONTIUM TP memories and registers (using DMA);

- *Streaming communication* to stream data into and / or out of the MONTIUM TP while the TP is computing.

MONTIUM Tile Processor The Tile Processor (TP) is the computing part of the MONTIUM processing tile. The MONTIUM TP can be configured to implement a particular DSP algorithm.

Figure 3.4 reveals that the hardware organization of the MONTIUM TP is very regular. The five identical ALUs (ALU1 through ALU5) in a tile can exploit spatial concurrency to enhance performance. This parallelism demands a very high memory bandwidth, which is obtained by having 10 local memories (M01 through M10) in parallel. The small local memories are also motivated by the locality of reference principle.

The data path has a width of 16-bits and the ALUs support both signed integer and signed fixed-point arithmetic. The ALU input registers provide an even more local level of storage. Locality of reference is one of the guiding principles applied to obtain energy efficiency in the MONTIUM TP.

A vertical segment that contains one ALU together with its associated input register files, a part of the interconnect and two local memories is called a Processing Part (PP). The five PPs together are called the Processing Part Array (PPA).

A relatively simple sequencer controls the entire PPA. The sequencer selects configurable PPA instructions that are stored in the instruction decoding block of Figure 3.4. For (energy) efficiency it is imperative to minimize the control overhead.

The PPA instructions, which comprise ALU, AGU, memory, register file and interconnect instructions, are determined by a DSP application designer at design time. All MONTIUM TP instructions are scheduled at design time and arranged into a MONTIUM sequencer programme. By statically scheduling the instructions as much as possible at compile time, the MONTIUM sequencer does not require any sophisticated control logic which minimizes the control overhead of the reconfigurable architecture.

Figure 3.5 shows a block diagram of the ALU that is used in the MONTIUM TP. A single ALU has four 16-bit inputs. Each input has a private input register file that can store up to four operands. The input register file cannot be bypassed, i.e. an operand is always read from an input register. Input registers can be written by various sources via a flexible interconnect. An ALU has two 16-bit outputs, which are connected to the interconnect.

The ALU is entirely combinational and there are no pipeline registers within the ALU. The function units (FUs) in *level 1* of the ALU can be configured to perform general arithmetic and logic operations that are available in languages like C (except multiplication and division). Neighbouring ALUs can also communicate directly on *level 2*. The West-output of an ALU connects to the East-input of the ALU neighbouring on the left.

The MONTIUM TP has no fixed instruction set, but the instructions are configured at configuration-time. During configuration of the MONTIUM TP, the CCU writes the configuration data (i.e. instructions of the ALUs, memories and interconnects, etc.; sequencer and decoder instructions) in the configuration memory of the MONTIUM TP. The size of

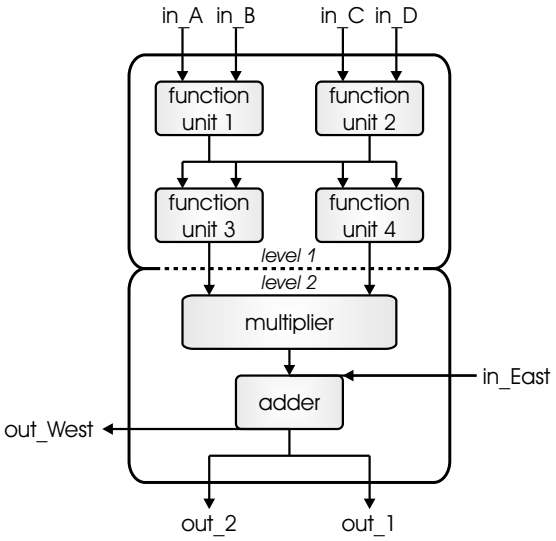

Figure 3.5: *Schematic overview of the ALU inside the* MONTIUM *TP.*

the total configuration memory of the MONTIUM TP is about 2.9 kB. However, configuration sizes of DSP algorithms mapped on the MONTIUM TP are typically in the order of 1 kB. For example, a 64-point Fast Fourier Transform (FFT) has a configuration size of 946 bytes.

By sending a configuration file containing configuration RAM addresses and data values to the CCU, the MONTIUM TP can be configured via the NoC interface. The configuration memory of the MONTIUM TP is implemented as a 16-bit wide Static RAM (SRAM) memory that can be written by the CCU. By only updating certain configuration locations of the configuration memory, the MONTIUM TP is partially reconfigured.

In the considered MONTIUM TP implementation, each local SRAM is 16-bit wide and has a depth of 1024 addresses, which results in a storage capacity of 2 kB per local memory. The total data memory inside the MONTIUM TP adds up to a size of 20 kB. A reconfigurable Address Generation Unit (AGU) accompanies each local memory in the PPA of the MONTIUM TP. It is also possible to use the local memory as a Look-up Table (LUT) for complicated functions that cannot be calculated using an ALU, such as sine or division (with one constant). The memory can be used in both integer or fixed-point LUT mode.

Design methodology

Good development tools are essential for quick implementation of applications in reconfigurable architectures. The intention of the MONTIUM development tools is to start with a high-level description of an application (in C / C++ or MATLAB) and translate this

description to a MONTIUM TP configuration [48]. For efficient implementation applications can be implemented on the MONTIUM TP using an assembly-like language, called MONTIUM Configuration Description Language (CDL).

The development tools comprise, among other things, a stand-alone MONTIUM simulator, a MONTIUM assembler and a MONTIUM configuration editor [54]. DSP applications are implemented on the MONTIUM TP using the proprietary MONTIUM CDL.

The MONTIUM design methodology to map DSP applications on the MONTIUM TP is divided in three steps:

1. The high-level description of the DSP application is analyzed and computationally intensive DSP kernels are identified;

2. The identified DSP kernels or parts of the DSP kernels are mapped on one or multiple MONTIUM TPs that are available in a SoC. The DSP operations are programmed on the MONTIUM TP using MONTIUM CDL. For a number of typical DSP algorithms, dedicated MONTIUM configuration generators are available (e.g. Finite Impulse Response (FIR) filters, Fast Fourier Transforms (FFTs));

3. Depending on the layout of the SoC in which the MONTIUM processing tiles are applied (refer to Section 3.4), the MONTIUM processing tiles are configured for a particular DSP kernel or part of the DSP kernel. Furthermore, the channels in the NoC between the processing tiles are defined.

3.5.2 PICOARRAY

The PICOARRAY defines a parallel processing approach within the application domain of wireless communication systems. The PICOARRAY typically targets the baseband processing (in base stations) of Third Generation (3G) / Fourth Generation (4G) wireless communication systems. The architecture aims to combine flexibility, ease of development and cost efficiency for the wireless communication market [86].

The target applications of the PICOARRAY typically consist of a mixture of DSP functionality, which can be both *streaming* and *block* based. Additionally, these wireless communication systems require distributed control of the DSP functions.

Architecture

The PICOARRAY is a tiled processor architecture, containing several hundreds of heterogeneous processors, connected through a compile-time scheduled interconnect [30, 31, 84]. The heterogeneous processors are connected together using a deterministic interconnect. The heterogeneous processors within the PICOARRAY, also referred to as Array Elements (AEs), are organized in a two-dimensional grid. The AEs communicate over 32-bit unidirectional buses and programmable bus switches. The inter-processor communication protocol implemented by the interconnect is based on a Time Division Multiplexing (TDM) scheme.

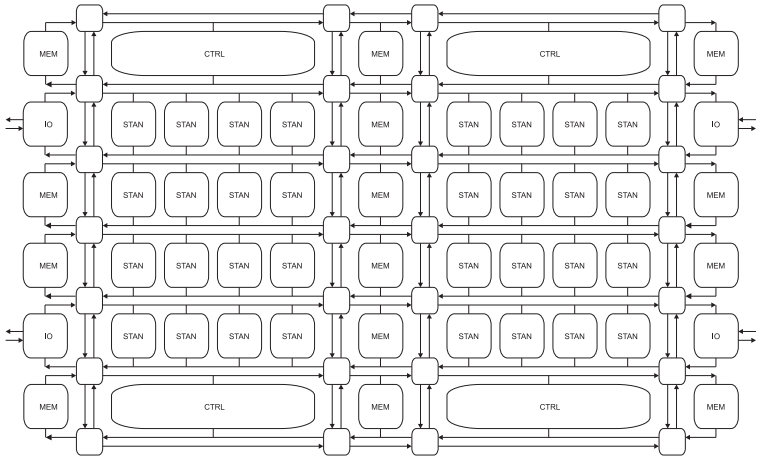

Figure 3.6: *The* PICOARRAY *composed of Array Elements and interconnect [84, 86].*

The GPP AEs in the PICOARRAY are implemented in three RISC processor variants which share a common core instruction set. The three processor variants have varying amounts of memory and additional instructions to implement certain wireless baseband control and DSP functions. *Standard*, *Memory* and *Control* variants of the RISC processors can be identified. All of the processors in the PICOARRAY are 16-bit, and use 3-way VLIW scheduling. The processors in the PICOARRAY are programmed and initialized via a special configuration bus. The small memory of the RISC processors is divided in separate instruction and data banks (i.e. Harvard architecture).

Function Accelerator Units (FAUs) are AEs in the PICOARRAY which include configurable hardware for accelerating a number of computationally intensive tasks, e.g. correlation and trellis processing. The hardware in the FAU can be configured to specific functions, like Viterbi, Turbo or Reed Solomon decoding.

Furthermore, the PICOARRAY core is accompanied by system interface peripherals. The interface peripherals include a host interface and SRAM interface as well as high speed I/O interfaces, which link several PICOARRAYS together or connect to external systems. Several PICOARRAYS linked together enable the creation of scalable parallel data processing systems.

Design methodology

The design of the PICOARRAY and the accompanied tool chain strongly reflect the application domain with the following attributes:

- application partitioning in DSP kernels is traditionally done by the user;
- use of stream based processing;

- hand-crafted coding in an assembly language is often used to achieve compactness of code and efficiency.

The PICOARRAY is typically programmed using a mixture of VHDL, C and assembly language. The PICOARRAY design methodology to create PICOARRAY-based applications is distinguished in a number of steps [31, 84]:

1. *System decomposition*
   System decomposition is done by hierarchically breaking down the target application into components. The components consist of processes connected by signals. VHDL is used to describe the structure of the overall system, including the relationship between processes and signals;

2. *Component coding*
   The components of the decomposed system contain multiple processes connected by signals. Each individual process is programmed in C or in assembly language. The PICOARRAY C compiler deals only with the code for a single Array Element;

3. *System integration*
   The individual coded AEs are finally integrated in the PICOARRAY. During the system integration the processes implemented on the AEs are placed on the PICOARRAY. So, a specific processor is assigned to each entity in the design and all signals which link entities are routed. If appropriate, the design can be partitioned over multiple PICOARRAYS, which can be linked together to create a scalable system.

3.5.3 Silicon Hive

Silicon Hive develops configurable processors that address a trade-off between performance, power consumption, area and flexibility for several application domains, including SDR baseband processing [100]. The AVISPA is a typical instant of a configurable processor and targets typical DSP kernels for SDR systems [67, 88, 89].

The configurable processors combine:

- *minimal control overhead* by moving processor control to a C compiler;
- *extreme parallelism* at different levels: SIMD for DLP, VLIW for ILP and multi-core for TLP;
- *application domain customizing* by combining general purpose RISC operations with computationally efficient custom operations.

Accordingly, the configurable processors strike a balance between high computational efficiency and flexibility by three main measures:

1. The configurable processors couple extreme parallelism to modest clock frequencies. In this way high performance is achieved at low power dissipation, resulting in a high computational efficiency;

2. A C compiler has been developed that is capable of controlling the processor instead of adding control logic to the processor. Hence, the required control hardware logic in the configurable processors is virtually negligible, since the compiler is handling almost all control at design time;

3. Many general purpose RISC operations are replaced by (single) customized operations, which are implemented in the processor's data path with dedicated logic. So, the computational performance is increased, while the power dissipation has been reduced.

3.5.4 eXTREME PROCESSING PLATFORM

The eXTREME PROCESSING PLATFORM (XPP) is a run-time reconfigurable data processing architecture. The XPP provides parallel processing power for high bandwidth data like e.g. video or audio processing. The XPP targets for streaming DSP applications in the multimedia and telecommunications domain [13, 83].

Architecture

The XPP architecture is based on a hierarchical array of coarse-grained, adaptive computing elements, called Processing Array Elements (PAEs). The PAEs are clustered in Processing Array Clusters (PACs). All PAEs in the XPP architecture are connected through a packet-oriented communication network. Figure 3.7 shows the hierarchical structure of the XPP array and the PAEs clustered in a PAC.

Different PAEs are identified in the XPP array: *ALU-PAEs*, *RAM-PAEs* and *FNC-PAEs*. The ALU-PAE contains a multiplier and is used for DSP operations. The RAM-PAE contains a Random Access Memory (RAM) to store data. The Function (FNC)-PAE is a unique sequential VLIW-like processor core. The FNC-PAEs are dedicated to the control flow and sequential sections of applications. Every PAC contains ALU-PAEs, RAM-PAEs and FNC-PAEs.

The PAEs operate according to the data flow principle; a PAE starts processing data as soon as all required input packets are available. If a packet cannot be processed, the pipeline stalls until the packet is processed. So, it is possible to map a signal flow graph to the ALU-PAEs.

Each PAC is controlled by a Configuration Manager (CM). The CM is responsible for writing configuration data into the configurable object of the PAC. Multi-PAC XPP arrays contain additional CMs for concurrent configuration data handling, arranged in a hierarchical tree of CMs. The top CM, called supervising CM, has an external interface which connects the supervising CM to an external configuration memory.

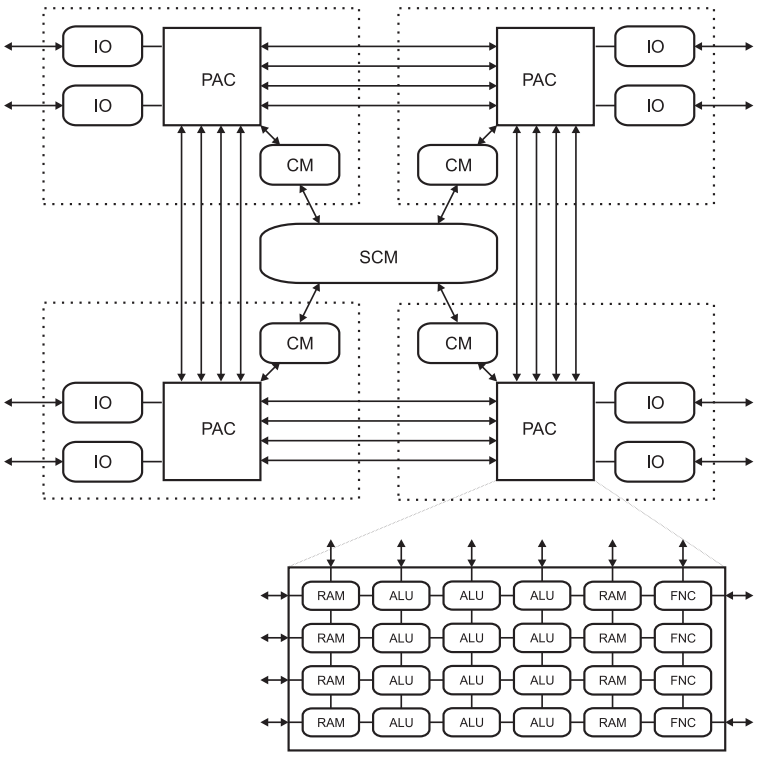

Figure 3.7: *The structure of an* EXTREME PROCESSING PLATFORM *array composed of four Processing Array Clusters [13].*

Design methodology

DSP algorithms are directly mapped onto the XPP array according to their data flow graphs. The flow graph nodes define the functionality and operations of the PAEs, whereas the edges define the connections between the PAEs. Basically, the XPP array is programmed using the Native Mapping Language (NML). In NML descriptions, the PAEs are explicitly allocated and the connections between the PAEs are specified. Optionally, the allocated PAEs are placed onto the XPP array. NML also includes statements to support configuration handling. Configuration handling is an explicit part of the application description.

A vectorizing C compiler is available to translate C functions to NML modules. The vectorizing compiler for the XPP array analyzes the code for data dependencies, vectorizes those code sections automatically and generates highly parallel code for the XPP array. The vectorizing C compiler is typically used to program *regular* DSP operations which are mapped on *ALU-PAEs* and *RAM-PAEs* of the XPP array.

Furthermore, a coarse-grained parallelization into several FNC-PAE threads is very

useful when *irregular* DSP operations exist in an application. This allows to even run irregular, control-dominated code in parallel on several FNC-PAEs. The FNC-PAE C compiler is similar to a conventional RISC compiler extended with VLIW features to take advantage of ILP within the DSP algorithms.

Given the discussed design tools, these steps in the XPP application design flow are recognized:

1. *Profiling & partitioning* of the application;
2. *Mapping critical code sections*:
   - *Regular* DSP code sections are mapped on *ALU-PAEs* and *RAM-PAEs* of the XPP array. The vectorizing C compiler translates C functions to NML modules. Optimizations to the modules are performed using NML;
   - *Irregular* DSP code sections are parallelized into several FNC-PAE threads and mapped on FNC-PAEs. The FNC-PAE C compiler generates binaries which can be optimized in assembly language;
3. *System integration* of the *regular* and *irregular* code sections in the EXTREME PROCESSING PLATFORM.

3.6 Summary

Different aspects of hardware architectures and developments related to semiconductor technology have been discussed in this chapter. The miniaturization of semiconductor technology faces several problems and brings new challenges in embedded systems design:

- Increasing transistor budget leads to more complex IC designs;
- Semiconductor technology suffers with increasing power density in IC designs;
- Introducing parallelism in multi-core hardware architectures:
  1. Exploit Thread-Level Parallelism (TLP),
  2. Data-Level Parallelism (DLP) and
  3. Instruction-Level Parallelism (ILP) in hardware architectures;
- Reducing the productivity gap by reusing Intellectual Property (IP);
- Solving the testability gap by reusing Intellectual Property (IP).

Different hardware architectures are identified that are characterized by different flexibility, performance and energy efficiency trade-offs. The General Purpose Processor (GPP) is the most flexible hardware architecture. It can be programmed to perform almost any

algorithm. Due to the large overhead in control, these processors are not very energy efficient. Application Specific Integrated Circuits (ASICs) on the other hand are not flexible at all, but implement dedicated application specific functions. ASICs obtain best computational performance and are energy efficient as well.

Reconfigurable architectures are positioned in between the GPP and the ASIC. Fine-grained reconfigurable architectures are bit-level programmable, while coarse-grained reconfigurable architectures are flexible at word-level. Coarse-grained reconfigurable architectures are designed for a particular application domain with an optimum trade-off in performance, flexibility and energy efficiency.

Modern reconfigurable processors exploit parallelism heavily to achieve high performance with reasonable power dissipation:

- Multi-core architectures are defined to exploit Thread-Level Parallelism (TLP);
- SIMD-like architectures are developed to exploit Data-Level Parallelism (DLP);
- VLIW techniques are performed to exploit Instruction-Level Parallelism (ILP).

These techniques are embedded in the analyzed coarse-grained reconfigurable architectures. The modern reconfigurable architectures that have been analyzed all involve the same design methodology:

1. *Profiling & partitioning* of the high-level DSP application description;
2. *Mapping the critical DSP kernels* on the target architecture;
3. *System integration* of all DSP kernels in the entire SoC.

Chapter 4

OFDM Communication Systems

This chapter covers Orthogonal Frequency Division Multiplexing (OFDM) communication systems. First the general framework of an OFDM system is introduced and different OFDM-based standards are given. Several implementations of typical Digital Signal Processing (DSP) kernels in OFDM systems are discussed.

The DSP kernels in the baseband processing of a HiperLAN/2 receiver have been mapped on MONTIUM *Tile Processors (TPs). The details of mapping these typical DSP kernels are discussed. Furthermore, the baseband processing of Digital Audio Broadcasting (DAB) and Digital Radio Mondiale (DRM) receivers has been investigated.*

The OFDM receiver is used to illustrate the feasibility of designing an adaptive multi-mode wireless communication receiver build on a heterogeneous reconfigurable System-on-Chip (SoC). General purpose and coarse-grained reconfigurable processing elements of the heterogeneous reconfigurable SoC are used for implementing the OFDM receiver functions. The baseband signal processing is mainly performed on MONTIUM *processing elements, whereas the General Purpose Processor (GPP) contributes in controlling the system.*

Major parts of this chapter have been published in [P1, P3, P4, P5, P6, P8, P10, P13, P18, P20].

4.1 Introduction

This chapter covers the implementation of Orthogonal Frequency Division Multiplexing (OFDM)-based wireless communication receivers on heterogeneous reconfigurable hardware. The heterogeneous reconfigurable System-on-Chip (SoC) template, as shown in Figure 3.3, is applied to illustrate the mapping of OFDM receivers. Special attention is given to the mapping of typical Digital Signal Processing (DSP) kernels in OFDM systems on the coarse-grained MONTIUM Tile Processor (TP). Section 4.2 discusses the basic principles of the OFDM communication system.

In Section 4.3 the High Performance Radio LAN type 2 (HiperLAN/2) communication receiver is analyzed. The HiperLAN/2 standard defines the basic functionality of the OFDM-based system. However, the standard does not define how to implement the functions of the receiver. Firstly, the general context of the HiperLAN/2 system is given in Section 4.3.1. Secondly, the DSP kernels of the HiperLAN/2 baseband processing are discussed in Section 4.3.2. The functions of the DSP kernels in the HiperLAN/2 receiver are described thoroughly to understand the integral system design of the HiperLAN/2 receiver. Thirdly, the computationally intensive parts of the baseband processing in the HiperLAN/2 receiver are implemented on the coarse-grained MONTIUM TP in Section 4.3.3. Finally, simulations are performed on the HiperLAN/2 baseband processing functions which are implemented on the heterogeneous reconfigurable SoC template. The simulations in Section 4.3.4 are performed to verify the MONTIUM-based DSP kernels.

To illustrate the generality of our approach we investigate in Section 4.4 and 4.5 OFDM receivers that are applied for Digital Audio Broadcasting (DAB) and Digital Radio Mondiale (DRM) systems, respectively. Problems that arise for mapping the receivers on the MONTIUM TP are investigated. Partial mappings of DSP kernels on the MONTIUM TP are discussed in these sections.

Section 4.6 summarizes the main contributions of this chapter. In Section 4.6.1 conclusions with respect to the mapping of OFDM receivers on heterogeneous reconfigurable SoC architectures are drawn. In order to utilize the dynamic reconfiguration capabilities of the heterogeneous reconfigurable SoC architecture, opportunities for adaptivity in multi-standard multi-mode wireless communication receivers are identified as well. Recommendations for hardware modifications are discussed in Section 4.6.2.

4.2 Orthogonal Frequency Division Multiplexing

The concept of Orthogonal Frequency Division Multiplexing (OFDM) has been known since 1966 [22], however, due to implementational complexity it has been widely adopted since the 1990s. OFDM is a special multi-carrier modulation technique, which utilizes multiple sub-carriers within a single channel. The modulation technique divides the high data rate information in several parallel bit streams and each of these bit streams modulates a separate sub-carrier.

A general view of the OFDM system is depicted in Figure 4.1. The data to be transmitted is passed through a serial-to-parallel convertor. The convertor splits the data in K streams, which are all modulated by a different carrier frequency, f_s. The spacing between the sub-carriers is Δf and the overall bandwidth of the K sub-carriers is $K \cdot \Delta f$.

The set of K orthogonal sub-carriers, called the OFDM signal, is transmitted over the communication channel. At the receiver side of the channel, the sub-carriers of the OFDM signal are demodulated. The data of the separate sub-carriers are recombined by using a parallel-to-serial convertor.

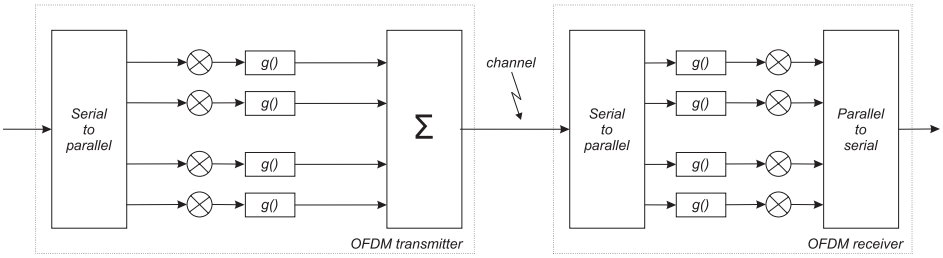

Figure 4.1: *Basic OFDM system structure.*

The key characteristic of OFDM systems is that they use a set of sub-carriers that are orthogonal to each other. Employing a set of orthogonal sub-carriers gives the advantage of overlapping sub-carriers' spectra, which increases the spectral efficiency. The common method for obtaining orthogonality of sub-carriers is to choose their frequency spacing, Δf, equal to the inverse of the sub-carrier's symbol duration, T:

$$\Delta f = \frac{1}{T} \tag{4.1}$$

Classical OFDM implementations were proposed using filter-banks. However, the implementation of the classical filter-banks approach becomes complex and costly for an increasing number of sub-channels [56]. In order to reduce the implementational complexity, the implementation of OFDM systems is based on the Discrete Fourier Transform (DFT). The set of orthogonal signals can be generated using the DFT, since the DFT results in an orthogonal set of signals [116]. In practice, the DFT operations are performed by the Fast Fourier Transform (FFT) algorithm [26], which implements the DFT operations more efficiently.

Figure 4.2 depicts a block diagram of a typical OFDM system based on the FFT algorithm. The OFDM transmitter combines the data streams of the separate sub-carriers using the inverse FFT (iFFT). The generated OFDM signal is sent over the communication channel. At the receiver the OFDM signal is converted from the time domain to the frequency domain, using the FFT. The output of the FFT contains the data streams of the separate sub-carriers.

According to (4.1) the sub-carrier's symbol duration, T, is of importance for the orthogonal characteristics of the OFDM system. When the OFDM system is implemented

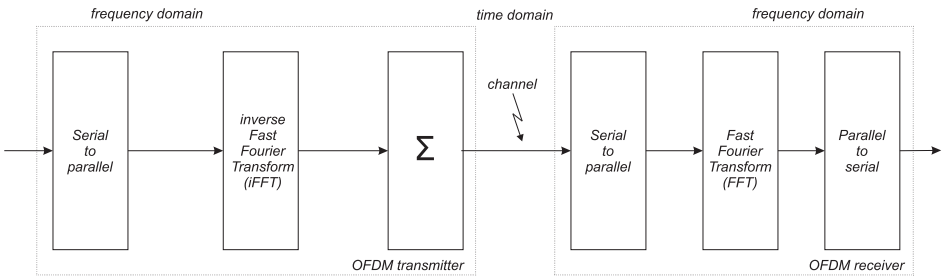

Figure 4.2: *Basic OFDM system structure based on FFT.*

based on the FFT, the FFT window period is equal to T. The data covered by the FFT window is referred to as the OFDM symbol in OFDM communication systems. An OFDM system with N orthogonal sub-carriers employs an N-point FFT/iFFT. Hence, during the OFDM symbol period N samples have to be processed by the FFT/iFFT causing the sampling period, T_s, to be T/N.

4.2.1 OFDM signal distortion

Wireless communication systems suffer from distortion of the transmitted signals through the wireless channel. The mechanisms behind the distortion are diverse, and can be attributed to reflection, diffraction and scattering, which lead to multipath fading at a specific location. Multipath channels cause two main problems for an OFDM system: intersymbol interference (ISI) and intercarrier interference (ICI). Due to multiple signal paths, multiple delayed signals originating from a single transmitter are combined at the receiver. Those signals interfere with each other, which is referred to as ISI. Since OFDM signals exist of many separate overlapping carriers, these carriers can also interfere with each other causing ICI.

Whereas wideband channels encounter frequency-selective fading, the many narrowband sub-channels in OFDM systems all individually encounter flat fading. This is one big advantage of employing OFDM systems.

To combat interference, as induced by a multipath channel, the OFDM signal is, generally, extended by a guard interval. Figure 4.3 shows the typical OFDM symbol structure with the cyclic extended guard interval. The guard interval has duration T_g. The length of the guard interval is chosen according to the typical delay spread of the wireless channel. Typically, the N_g samples of the cyclic extended guard interval, usually referred to as the *cyclic prefix*, are a copy of the last N_g samples of the time domain OFDM symbol.

The cyclic extended OFDM symbol consists of a prefix and a useful part. The total duration of the cyclic extended OFDM symbol is equal to T_{OFDM}:

$$T_{OFDM} = T_g + T, \tag{4.2}$$

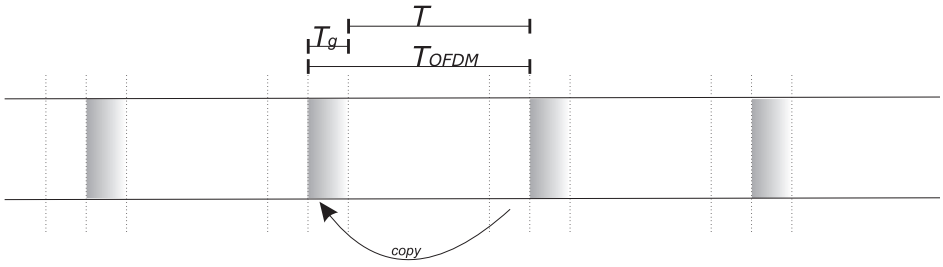

Figure 4.3: *OFDM symbol structure.*

where T denotes the duration of the useful OFDM symbol part and T_g denotes the guard time.

The N_g samples of the prefix are neglected at the receiver because of its redundancy. Furthermore, the cyclic extension is useful for synchronization of the OFDM symbol stream, since the cyclic extended OFDM symbol shows periodicity.

Because of their orthogonality characteristics, OFDM systems are sensitive to distortions in the frequency domain. The frequency spacing, as mentioned in (4.1), between the carriers of the OFDM symbol needs to be constant over the entire bandwidth in order to satisfy the existence of orthogonal spectral components in the OFDM signal.

Due to moving objects in the wireless environment, Doppler effects appear in the wireless channel. The Doppler effects result in time-varying frequency shifts of the spectral components in OFDM signals. Furthermore, non-linearities of the OFDM transmitter and receiver equipment lead to time-varying frequency shifts. Both effects ruin the orthogonality of the sub-carriers in the OFDM signal. The non-linearities of the equipment in OFDM systems are generally due to mismatches between the local oscillators (LOs) of the transmitter and receiver. Mismatches of the LO appear both in frequency and phase. Frequency mismatches are typically corrected by the OFDM receiver in the time domain, whereas phase mismatches are typically corrected in the frequency domain.

4.2.2 Generic OFDM receiver framework

OFDM-based communication systems are all designed according to a generic OFDM framework. Figure 4.4 shows this generic OFDM framework. In this framework the characteristic properties are based on specific OFDM standards. This means that, among other things, the number of sub-carriers and the length of the guard interval differ for each OFDM standard. The characteristics of an OFDM receiver for a particular standard can even differ if it has different modes defined. Characteristics of the OFDM-based standards, High Performance Radio LAN type 2 (HiperLAN/2), Digital Radio Mondiale (DRM)[1] and Digital Audio Broadcasting (DAB) are summarized in Table 4.1 [36, 37, 38].

[1] Only the characteristics of DRM for single spectrum occupancy are considered with channel bandwidth of 10 kHz.

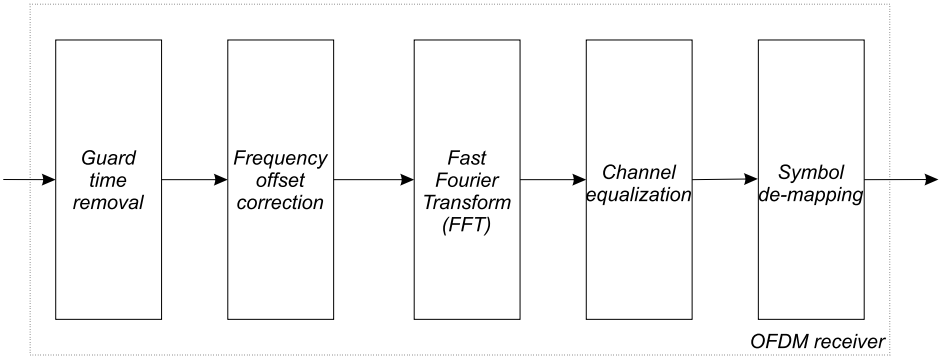

Figure 4.4: *Generic OFDM receiver framework.*

Many more characteristics of OFDM systems are of importance for the design of the communication systems. In Section 4.3, 4.4 and 4.5 the details, which are of importance for implementing an OFDM standard in embedded systems, are discussed for HiperLAN/2, DAB and DRM, respectively. Although these standards are all examples of OFDM systems, many differences exist in terms of sub-carrier modulation, number of modulated sub-carriers and OFDM frame structure.

4.3 High Performance Radio LAN type 2

High Performance Radio LAN type 2 (HiperLAN/2) [36] is a Wireless LAN (WLAN) access technology and is similar to the IEEE 802.11a standard at the physical level [62]. HiperLAN/2 operates in the 5 GHz frequency band. The air interface is based on Time Division Multiple Access (TDMA) and Time Division Duplex (TDD). In the TDMA system the Medium Access Control (MAC) frame is divided in multiple time slots. Multiple users can make use of the same MAC frame by utilizing different time slots. TDD means that time slots for both downlink (DL) and uplink (UL) are available within the same MAC frame [35].

4.3.1 HiperLAN/2 transport mechanism

Within the fixed duration of 2 ms, six different transport channels are identified in the MAC frame of HiperLAN/2. Transport channels describe the basic message format and each channel has a dedicated function in the MAC frame:

- *Broadcast Channel (BCH)*
  The BCH is used in DL direction and contains broadcast control channel information concerning the radio cell. The amount of data is fixed;

Table 4.1: *Characteristics of different OFDM standards [36, 37, 38].*

		Hiper LAN/2	DAB				DRM			
			I	II	III	IV	A	B	C	D
Bandwidth	$[MHz]$	16.25	1.536	1.536	1.536	1.536	0.01	0.01	0.01	0.01
# Carriers (FFT size), N		64	2 048	512	256	1 024	288	256	176	112
# Modulated carriers, K		52	1 536	384	192	768	225	205	137	87
Symbol time, T_{OFDM}	$[\mu s]$	4	1 246	312	156	623	26 667	26 667	20 000	16 667
Guard time, T_g	$[\mu s]$	0.8	246	62	31	123	2 667	5 333	5 333	7 333
Useful time, T	$[\mu s]$	3.2	1 000	250	125	500	24 000	21 333	14 667	9 333
sub-carrier spacing, Δf	$[kHz]$	312.5	1.0	4.0	8.0	2.0	0.0417	0.0469	0.0682	0.1071

- *Frame Channel (FCH)*
  Information that describes the structure of the MAC frame is sent by the FCH in DL direction. The amount of data is variable;

- *Access feedback Channel (ACH)*
  Terminals that have used the Random Channel (RCH) in the previous MAC frame are informed in the ACH about the result of their access attempts. The information is sent in DL direction;

- *Long transport Channel (LCH)*
  The LCH is used to transport user data information as well as for control information. The LCH is utilized in UL, DL and direct link (DiL) phase as well;

- *Short transport Channel (SCH)*
  The SCH is used to transport short control information in the UL, DL and DiL direction;

- *Random Channel (RCH)*
  Mobile terminals use the RCH to send control information to the Access Point, when the mobile terminal has no granted channel with the Access Point available. The requests are sent in UL direction.

Figure 4.5 shows the basic MAC frame structure for a single sector HiperLAN/2 system. A minimum requirement of the HiperLAN/2 standard is that the MAC frame at least contains the transport channels BCH, FCH, ACH and RCH. For transmitting user data a DL and UL phase is provided. Optionally, in direct mode, when data is transmitted between mobile terminals (without the base station involved), the DiL phase is provided between the DL and UL phase. The duration of the BCH is fixed, while the duration of the FCH, ACH, DL, DiL and UL phase and the number of RCHs are dependent on the traffic situation in the HiperLAN/2 system. The order of the transport phases within the MAC frame is fixed.

The transmission format of the transport channels is a burst, which consists of a preamble and a data part. The preamble is a sequence of predefined known Orthogonal Frequency Division Multiplexing (OFDM) symbols. The receiver uses the preamble to estimate the various distortions in the received OFDM signal. Moreover, the preamble creates means for synchronization with the MAC frame. Different preambles are employed in HiperLAN/2: preamble *A, B (short), B (long)*, and *C*. Which preamble is used, depends on the burst type. In HiperLAN/2 five different kinds of bursts are defined: *Broadcast, Downlink, Uplink (short), Uplink (long)*, and *Direct link*. All burst types use one or more preambles, preceding the data part of the burst. Figure 4.6 shows how the preambles are used for each burst type. Table 4.2 shows the relation between the transport channels and the burst types for all channel directions (i.e. DL, UL and DiL).

The task of the physical layer in HiperLAN/2 is to modulate bits that originate from the data link control layer at the transmitter side and to demodulate them at the receiver side. OFDM with an FFT size of 64 points is used in HiperLAN/2. 52 sub-carriers are utilized (i.e. modulated) in the OFDM signal, which occupies a bandwidth of $16.25\ MHz$. 4 of

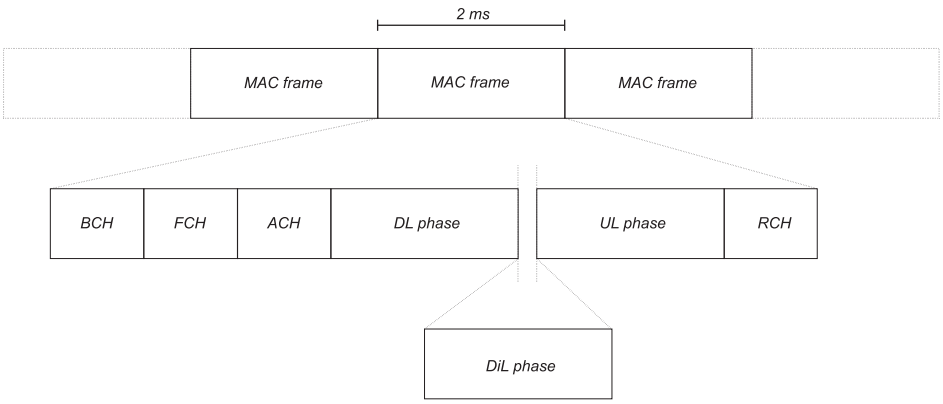

Figure 4.5: *Basic MAC frame structure for single sector HiperLAN/2 system.*

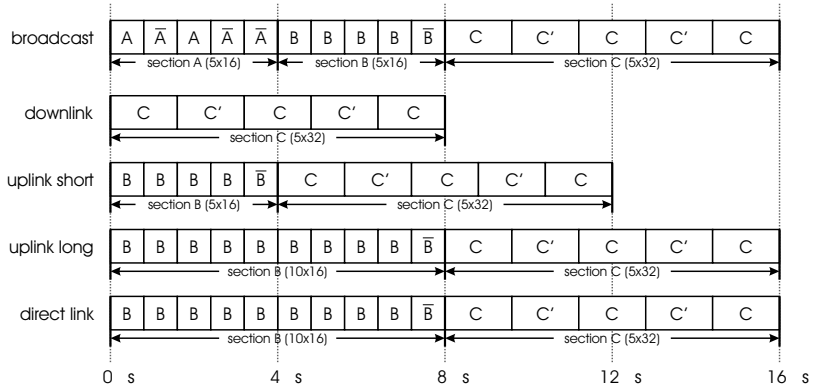

Figure 4.6: *HiperLAN/2 burst structures.*

these 52 sub-carriers are used to transmit pilot tones, the remaining 48 sub-carriers are used to transmit data. The channel bit rate of HiperLAN/2 at the physical level depends on the modulation type and is either 12, 24, 48 or 72 $Mbps$ for BPSK, QPSK, QAM-16 or QAM-64, respectively. Because of Forward Error Correction (FEC) coding, the net experienced user bit rate is maximally 54 $Mbps$ (Table 4.3). Only the Short transport Channel (SCH) and Long transport Channel (LCH) are able to employ all physical level operation modes, other transport channels employ BPSK and coding rate, $R = 1/2$, only.

Table 4.2: *Transport channels and their associated burst types in different channel directions.*

	downlink (DL)	uplink (UL)	direct link (DiL)
BCH	*Broadcast*	–	–
FCH	*Broadcast*	–	–
ACH	*Broadcast*	–	–
LCH	*Downlink*	*Uplink (short), Uplink (long)*	*Direct link*
SCH	*Downlink*	*Uplink (short), Uplink (long)*	*Direct link*
RCH	–	*Uplink (short), Uplink (long)*	–

Table 4.3: *Operation modes in HiperLAN/2 [36].*

Mode	Modulation	Coding Rate, R	Channel Data Throughput [$Mbps$]	User Data Throughput [$Mbps$]
I	BPSK	1/2	12	6
II	BPSK	3/4	12	9
III	QPSK	1/2	24	12
IV	QPSK	3/4	24	18
V	QAM-16	9/16	48	27
VI	QAM-16	3/4	48	36
VII	QAM-64	3/4	72	54

Timing requirements

The OFDM symbol is the basic data unit in the HiperLAN/2 system. At the physical level of the radio channel, all information is encapsulated in OFDM symbols. In HiperLAN/2 the OFDM symbols are transmitted with a period of 4 μs. The *OFDM symbol* time determines the basic timing constraints of the signal processing functions in the HiperLAN/2 receiver. The main timing properties that have to be considered in the HiperLAN/2 receiver are:

- *OFDM symbol* time;
- *MAC frame* time;
- *Propagation delay guard* time;
- *Sector switch guard* time;
- *Radio turn-around* time.

Due to the MAC frame structure, the HiperLAN/2 receiver needs to be flexible in order to support all different transport channels and their associated modulation schemes. Although the order of the transport phases within the MAC frame is fixed, the duration of

the different phases is variable. The duration of each HiperLAN/2 MAC frame, the *MAC frame* time, is fixed at 2 ms.

The BCH, FCH, ACH and DL transport phases within one HiperLAN/2 MAC frame are always transmitted in concatenated manner, without any intervals in between. The DL, DiL and UL transport phases are transmitted with some guard space in between. The guard spaces in between can either be a propagation delay guard or a sector switch guard. The propagation delay guard has been inserted to cope with wireless channel propagation delays. The minimum *propagation delay guard* time between two adjacent UL bursts, between two adjacent DiL bursts, and between adjacent DiL and UL bursts is 2 μs. The sector switch guard has been inserted to cope with systems implemented with multiple antenna sectors. The minimum *sector switch guard* time between adjacent bursts of different sectors within the MAC frame is 800 ns.

At the HiperLAN/2 system level, the *radio turn-around* time has been defined to cope with switching between transmit and receive mode. The TDD principle in the HiperLAN/2 system provides that both DL and UL phases exist in the same MAC frame. As a consequence, the HiperLAN/2 transceiver has to be able to switch from receive to transmit mode, and vice versa. The maximum radio turn-around time in HiperLAN/2 is 6 μs.

4.3.2 HiperLAN/2 receiver structure

A HiperLAN/2 receiver design can be based on the generic OFDM receiver framework from Figure 4.4. This framework is adapted to the specific characteristics of the HiperLAN/2 standard. Figure 4.7 shows the main building blocks of the HiperLAN/2 receiver. Only building blocks in the digital domain of the receiver are considered, starting at the output of the Analog-to-Digital Converter (ADC).

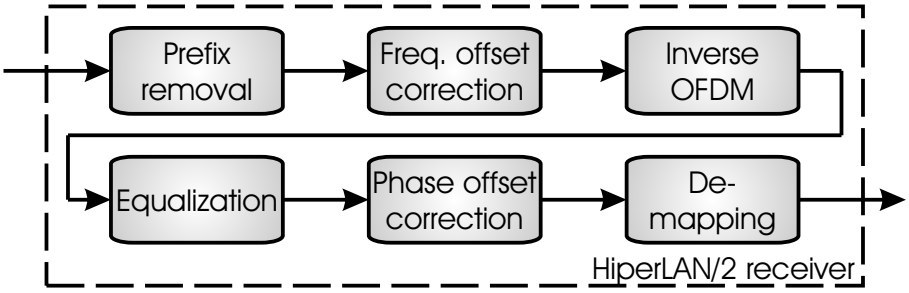

Figure 4.7: *HiperLAN/2 receiver block diagram.*

The sampling rate of the ADC is equal to 20 MHz, as is recommended by the HiperLAN/2 standard [36]. Hence, one OFDM symbol with a duration of 4 μs is represented by a block of 80 complex-number samples. Consequently, once per 80 samples the receiver has enough information to demodulate the received samples and output the resulting bits.

The receiver does not only convert the received signal to data bits by performing the inverse of the transmitter, but also has to correct distortions caused by the radio channel. The receiver can roughly be divided into two parts, a *time domain* and a *frequency domain* part.

The locations of the functions in the receiver architecture, shown in Figure 4.7, are based on a trade-off between the necessary resolution for a certain correction and the solution with the minimum number of operations. Furthermore, one tries to keep the corrections independent of each other by deciding the execution order of the functions. The most important functions (i.e. the most computationally intensive) will be described in order to implement the functionality in domain specific embedded processors (e.g. the MONTIUM architecture). Only the receiver part of the HiperLAN/2 physical layer has been considered, although the transmitter part can be described in a similar way.

The HiperLAN/2 receiver contains at least the following functions:

- OFDM symbol synchronization / OFDM prefix removal;
- OFDM symbol frequency offset correction;
- Inverse OFDM;
- OFDM symbol equalization;
- OFDM symbol phase offset correction;
- OFDM symbol de-mapping;
- De-interleaving;[2]
- Channel decoding;[2,3]
- De-scrambling.[2]

OFDM symbol synchronization

The existence of cyclic extended OFDM symbols, and preambles in the HiperLAN/2 system facilitates synchronization mechanisms in the receiver. The receiver first needs to synchronize with the transmitter. It does this by detecting the start of a transmission and then detecting the preamble sections by means of correlating the received stream of samples with the known preamble.

Because the sampling clock of the receiver hardware is not synchronized with the clock in the transmitter, the OFDM symbol window in the receiver may slowly wander away from the ideal OFDM symbol window. Therefore, the start position of the data of each received OFDM symbol has to be determined. The prefix information of an OFDM

[2] *De-interleaving*, *Channel decoding* and *De-scrambling* are not defined as baseband processing functions in this thesis.

[3] The implementation of channel decoding algorithms on reconfigurable hardware is discussed in Chapter 6.

Table 4.4: *Matched filter sizes applied in HiperLAN/2.*

Detection of	Size of matched filter
Preamble *A*	16 complex-number samples
Preamble *B*	16 complex-number samples
Preamble *C*	32 complex-number samples
Prefix	16 complex-number samples

symbol is used for this purpose. According to the structure of the HiperLAN/2 OFDM symbol in Figure 4.3, the first 16 samples and the last 16 samples of the OFDM symbol contain the same information. Therefore, the maximum of a sequence of correlations between 16 samples and 16 samples received 64 samples earlier reveals the location of the prefix (i.e. the start of the OFDM symbol). After the prefix has been detected, the prefix is removed from the OFDM symbol and the useful data is then processed by the remaining baseband processing functions. Detecting the preamble is usually referred to as the *prefix removal* or *guard time removal* function.

In the prefix removal function (and the MAC frame synchronization function) complex-number samples are detected with matched filters in order to synchronize the receiver. In Table 4.4 the sizes of the matched filters that are applied in the different functions of the HiperLAN/2 receiver are shown.

The cross-correlation operation applied in these matched filters is defined in (4.3). The output, $\tilde{z}$, from a matched filter with N samples and two complex-number inputs $\tilde{x}$ and $\tilde{y}$ is:

$$\tilde{z}\left[l\right] = \frac{1}{N} \sum_{i=0}^{N-1} \tilde{x}\left[i\right] \cdot \tilde{y}\left[l+i\right]^{*}, \tag{4.3}$$

where $\tilde{y}\left[i\right]^{*}$ denotes the complex conjugate of $\tilde{y}\left[i\right]$.

OFDM symbol frequency offset correction

Usually, the local oscillator (LO) of the HiperLAN/2 transmitter and receiver are not fully synchronized. There exists a difference in mix frequencies between transmitter and receiver, which is called frequency offset and causes intercarrier interference (ICI). These offsets are primarily caused by inaccurate oscillators in the analog RF front-end of the transmitter and receiver. The ICI can be removed by compensating for the frequency offset.

A time domain algorithm can be used to estimate the frequency offset based on the preambles in the HiperLAN/2 system. The algorithm searches for periodicity in the preambles. In the time domain a frequency offset causes a phase-shift. Hence, the frequency offset can be determined by taking the total accumulated phase-shift between a

known and a received signal. The frequency offset is estimated by computing the cross-correlation of preamble *C* [15, 97].

Figure 4.6 shows that preamble *C* consists of a prefix of 32 samples and 2 equal sequences of 64 samples. The sequence consists of 2 parts (*C* and *C'*), which are mirrored. In order to estimate the frequency offset, the first 16 samples of the two sequences are cross-correlated:

$$\widetilde{X} = \sum_{i=0}^{15} \tilde{r}\left[i\right] \cdot \tilde{r}\left[i+64\right]^{*}, \tag{4.4}$$

with $\tilde{r}\left[i\right]$ the i^{th} complex-number sample of the received preamble.

Introducing channel effects, will result in the received baseband signal before sampling:

$$\tilde{r}\left(t\right) = \left[\tilde{s}\left(t\right) \cdot e^{-j2\pi f_{\Delta} t + \Theta} * \tilde{h}\left(t,\tau\right)\right] + \tilde{n}\left(t\right), \tag{4.5}$$

where

- $\tilde{r}\left(t\right)$ denotes the received baseband signal;
- $\tilde{s}\left(t\right)$ denotes the originally transmitted baseband signal;
- $\tilde{h}\left(t,\tau\right)$ denotes the Channel Impulse Response (CIR);
- $\tilde{n}\left(t\right)$ defines the channel noise;
- f_{Δ} declares the frequency offset between transmitter and receiver ($f_{\Delta} = f_{Tx} - f_{Rx}$);
- Θ denotes a constant rotation, e.g. the phase offset;
- the convolution operation is defined by $*$.

In order to analyze the frequency offset estimation, the noise, $\tilde{n}\left(t\right)$, added to the received baseband signal will be neglected. Moreover, for analysis purpose, the channel will be considered flat, $\tilde{h}\left(t,\tau\right) = \delta\left(\tau\right)$:

$$\tilde{r}\left(t\right) = \tilde{s}\left(t\right) \cdot e^{-j2\pi f_{\Delta} t} \tag{4.6}$$

Combining equations (4.4) and (4.6) yields the correlation result, after sampling:

$$\begin{aligned} \widetilde{X} &= \sum_{i=0}^{15} \left(\tilde{s}\left[i\right] e^{-j2\pi f_{\Delta} i}\right) \cdot \left(\tilde{s}\left[i\right] e^{-j2\pi f_{\Delta}\left(i+64\right)}\right)^{*} \\ &= \sum_{i=0}^{15} \left|\tilde{s}\left[i\right]\right|^{2} \cdot e^{j2\pi f_{\Delta} 64} \end{aligned} \tag{4.7}$$

The angle of the correlation result, $\widetilde{X}$, determines the total accumulated phase offset over the correlation interval of 64 samples:

$$\angle\widetilde{X} = -2\pi f_{\Delta} 64 \tag{4.8}$$

Hence, the phase offset per sample, ϕ, that is introduced by frequency offset equals:

$$\phi = -2\pi f_{\Delta} \tag{4.9}$$

The time-varying phase offset, φ, is linearly proportional with the frequency offset, f_{Δ}:

$$\varphi = -2\pi f_{\Delta} t \mid_{\phi\in[-\pi,\pi\rangle} \Leftrightarrow f_{\Delta} = \frac{-\varphi}{2\pi t} \tag{4.10}$$

The cross-correlation computes the total phase offset accumulated over 64 samples, which corresponds to a time interval of 3.2 μs. The maximum frequency offset, f_{Δ}, that can be detected in this way by using the preamble *C* is $\pm$ 156 kHz.

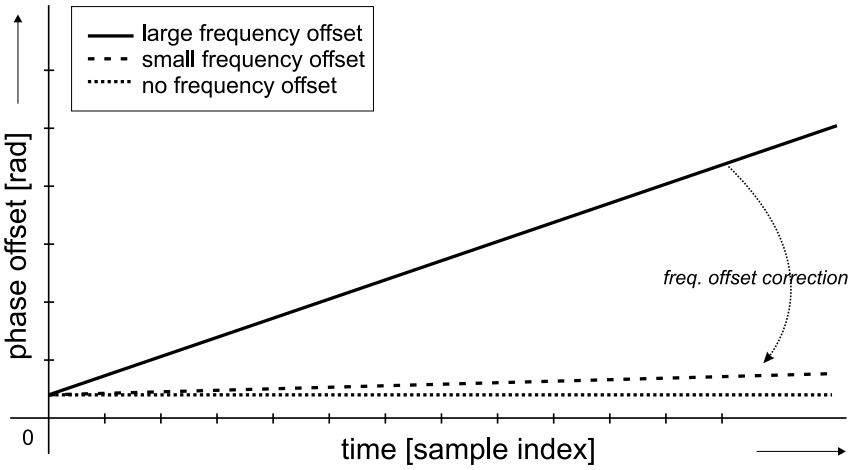

Figure 4.8: *The effect of frequency offset correction on phase offset.*

The frequency offset causes the phase of the received baseband signal to increase (or decrease) linearly with time. Ideal frequency offset correction would result in a phase of the baseband signal that is constant over time. However, in real-life the frequency offset cannot be determined exactly. As a result, the phase of the signal after correction is still time-variant.

The received baseband signal from (4.6) is simply compensated for frequency offset by multiplying the signal with the inverse frequency offset, as estimated in (4.10):

$$\hat{s}(t) = \tilde{r}(t) \cdot e^{-j\varphi} = \left(\tilde{s}(t) \cdot e^{-j2\pi f_{\Delta} t}\right) \cdot e^{-j\varphi} \simeq \tilde{s}(t) \tag{4.11}$$

The effects of frequency offset and (non-) ideal frequency offset correction are illustrated in Figure 4.8. The resulting time-variant phase needs to be corrected by a phase offset corrector in a later stage of the HiperLAN/2 receiver (i.e. equalization). Typically, the slightly changing phase offset is almost constant during one OFDM symbol of 4 μs. Using the pilot symbols in HiperLAN/2, the remaining phase offset can be corrected.

Inverse OFDM

64 time domain samples represent the useful data part of the HiperLAN/2 OFDM symbol that has to be demodulated. Before demodulation can take place, the sub-carrier values must be retrieved from the useful data part. The Fast Fourier Transform (FFT) is used to translate the information in this useful data part from the time domain to the frequency domain. The FFT is applied to the vector containing the 64 samples of the useful data part. The FFT efficiently implements the Discrete Fourier Transform (DFT) [26]:

$$\widetilde{C}[x] = \sum_{n=0}^{N-1} \tilde{r}[n] \cdot e^{-j2\pi \frac{xn}{N}}, \tag{4.12}$$

with $x = 0, \cdots, 63$ and $N = 64$ for the HiperLAN/2 receiver. $\tilde{r}[n]$ denotes the vector of received complex-number input samples in the time domain. $\widetilde{C}[x]$ denotes the vector of complex-number sub-carrier values in the frequency domain.

52 sub-carrier values – 48 complex-number data values and 4 pilot values – can be extracted from the output of the FFT operation. 12 of the 64 sub-carriers in the HiperLAN/2 system are not modulated, generating extra guard spectrum between adjacent HiperLAN/2 channels.

OFDM symbol equalization

The received complex-number values of the sub-carriers may still suffer from distortions that need to be corrected before de-mapping them to a bit stream. The distortions are introduced by the wireless radio channel:

- The reflections of the transmitted radio signals have different propagation times before they reach the receiver;
- Slow movement of the receiver introduces a Doppler shift distortion in the received signal.

These two effects together result in frequency-selective fading [92]. In each path the signal experiences Doppler shift due to motion in the environment. Such motion leads to frequency shifts in the signal's spectrum of individual reflected signals. These frequency shifts can be seen as time-varying phase shifts. At the antenna of the mobile receiver many reflected signals arrive, all with different phase shifts. Thus, their relative phases change all the time and the amplitude of the resulting composite signal is affected. So, the Doppler effects determine the rate at which the amplitude of the resulting composite signal changes.

In HiperLAN/2 the maximum speed of the terminal is assumed to be 3 m/s, which results in a maximum Doppler shift, f_d, of 52 Hz. The coherence time, T_C, is a statistical measure of the time duration over which the Channel Impulse Response is invariant. The Doppler shift and coherence time are inversely proportional to each other,

$$T_C \approx \frac{1}{f_d}, \tag{4.13}$$

yielding a coherence time of 20 ms for the HiperLAN/2 systems. Hence, the HiperLAN/2 channel can be considered constant during at least the time of one MAC frame with a length of 2 ms [92].

During equalization all carriers in the OFDM symbol are compensated for channel effects. The equalization in OFDM systems is usually performed in the frequency domain. In principle all carriers are multiplied with the inverse of the channel:

$$\widehat{C}\,[x] = \widetilde{C}\,[x] \cdot \frac{1}{\widehat{H}\,[x]}, \tag{4.14}$$

where $\widehat{H}\,[x]$ is the estimated channel transfer function of the x^{th} sub-carrier. $\widetilde{C}\,[x]$ denotes the received sub-carrier values at the output of the inverse OFDM function, while $\widehat{C}\,[x]$ gives the equalized sub-carrier values. In the HiperLAN/2 system 64 sub-carriers have been defined for $x = 0, \cdots, 63$.

The coefficients of the OFDM symbol equalizer in (4.14) are defined by the inverse of the estimated radio channel. This approach is referred to as *zero-forcing* (ZF) equalization. Drawback of the ZF equalizers is the fact that noise is heavily amplified when the channel encounters large attenuation. Under heavy noise distortions (i.e. low Signal-to-Noise Ratio (SNR)), it is suggested to use Minimum Mean Squared Error (MMSE) algorithms to determine the equalization coefficients instead [24, 25, 92]. The MMSE algorithms try to maximize the SNR at the output of the equalizer.

The equalization coefficients are determined by using information from the received preamble sections of the physical HiperLAN/2 bursts. Because all sub-carriers are modulated in preamble *C*, this preamble can efficiently be used to estimate the radio channel. By using preamble *C*, the channel transfer (i.e. the equalization coefficient) of every individual sub-carrier can be determined.

Since the coherence time of a HiperLAN/2 channel is about 20 ms, there is no need to continuously update the equalization coefficients. The HiperLAN/2 MAC frame has a duration of 2 ms, and therefore it is sufficient to determine the equalization coefficients only at the start of every MAC frame [15].

OFDM symbol phase offset correction

The various distortions of the transmitted signal caused by a wireless channel make it very difficult to estimate the frequency offset exactly at the receiver side. Therefore, the signal, after frequency offset correction, will still experience a small frequency offset. This small frequency offset causes a small, linearly changing phase difference between

the transmitted and received signals (Figure 4.8). This phase offset is small and is assumed to be constant during one OFDM symbol of 4 μs. Also, since the transmitter and receiver are not synchronized, there will be a (constant) phase offset between them.

4 pilot carriers are defined per OFDM symbol in the HiperLAN/2 system; three pilots in the OFDM symbol carry identical values, while the fourth pilot takes the negative value. The common rotation (i.e. phase offset) of the sub-carriers can be determined by calculating the mean rotation of the pilot carriers compared to their expected values:

$$\widehat{PO} = \frac{1}{4}\left(\frac{\widetilde{C}\,[57]}{P} + \frac{\widetilde{C}\,[43]}{P} + \frac{\widetilde{C}\,[7]}{P} - \frac{\widetilde{C}\,[21]}{P}\right), \tag{4.15}$$

with $\widetilde{C}\,[x]$ the received sub-carrier values of the pilots. P denotes the expected pilot value. This reference pilot signal is a cyclic extension of a 127-element random sequence [36]. $\widehat{PO}$ denotes the estimated common rotation of the sub-carriers. Once the common phase offset is known, one can apply phase offset correction to all received sub-carrier values. The correction for phase offset is similar to (4.14):

$$\widehat{C}\,[x] = \widetilde{C}\,[x] \cdot \frac{1}{\widehat{PO}}, \tag{4.16}$$

where $\widetilde{C}\,[x]$ denotes the equalized sub-carrier values. $\widehat{C}\,[x]$ yields the sub-carrier values after phase offset correction has been applied. Only the 48 sub-carriers containing data are applied for phase offset correction.

OFDM symbol de-mapping

In HiperLAN/2 four OFDM symbol mapping schemes are available: BPSK, QPSK, QAM-16 and QAM-64. Each of these schemes (or constellations) associates a different number of data bits per sub-carrier. The symbol mapping function in the HiperLAN/2 transmitter groups consecutive bits in the information stream and maps the consecutive bits to one symbol. The used symbol mapping constellations in HiperLAN/2 are shown in Figure 4.9 – 4.12. The de-mapping function assumes that the most likely symbol to be transmitted was the symbol that maps (given the constellation) to the complex-number value closest to the received complex-number value. This method of de-mapping is referred to as *hard decision* de-mapping[4].

[4] The counterpart of *hard decision* is *soft decision* de-mapping. During soft decision de-mapping the decision of the mapped symbol is made final during FEC decoding. Advantage of this approach is that all received information (i.e. the QAM symbol) is maintained during FEC decoding.

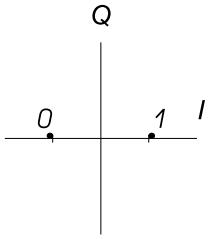

Figure 4.9: *HiperLAN/2 BPSK constellation.*

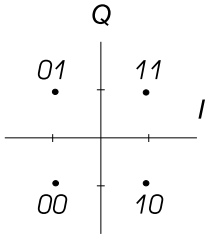

Figure 4.10: *HiperLAN/2 QPSK constellation.*

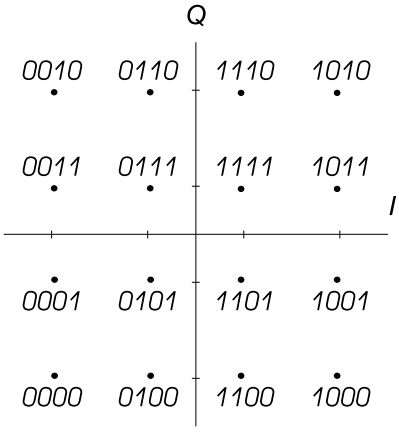

Figure 4.11: *HiperLAN/2 QAM-16 constellation.*

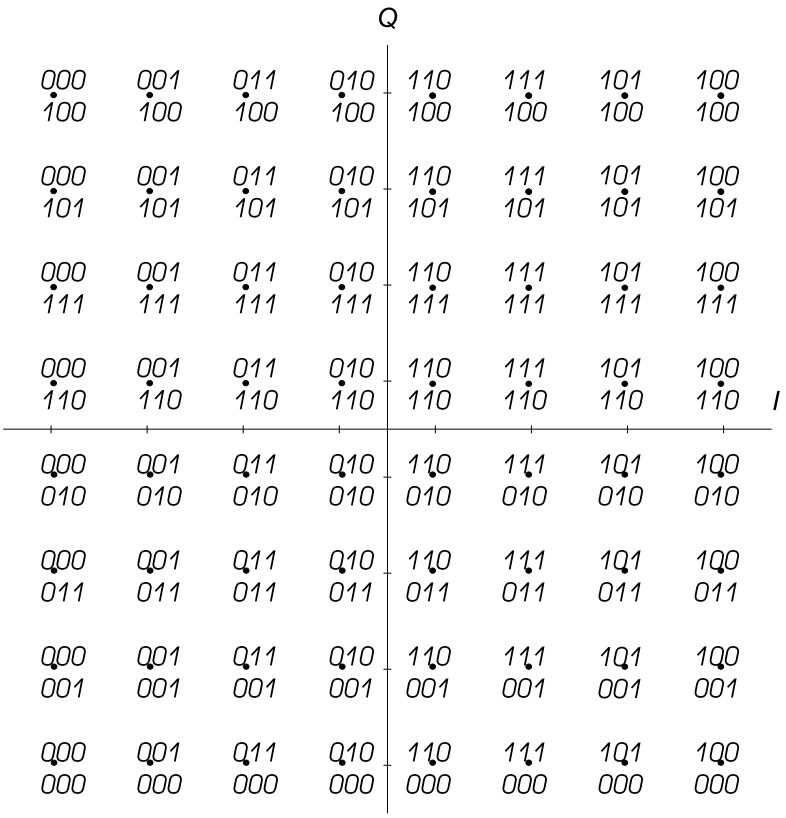

Figure 4.12: *HiperLAN/2 QAM-64 constellation.*

De-interleaving

In the HiperLAN/2 transmitter all encoded bits are interleaved by a block interleaver, which has a block size equal to the number of bits in a single OFDM symbol, N_{CBPS}. The interleaving is performed to ensure that:

1. adjacent coded bits are mapped onto non-adjacent sub-carriers;

2. adjacent coded bits are mapped alternately onto less and more significant bits of the constellation.

The two permutations of the block interleaver are given by the rules [36]:

$$i = \frac{N_{CBPS}}{16} \left(k \bmod 16\right) + \left\lfloor \frac{k}{16} \right\rfloor , \tag{4.17}$$

and

$$j = s \times \left\lfloor \frac{i}{s} \right\rfloor + \left(i + N_{CBPS} - \left\lfloor 16 \times \frac{i}{N_{CBPS}} \right\rfloor \right) \bmod s, \tag{4.18}$$

with $k = 0, \cdots, N_{CBPS}$, the index of the coded bit before the first permutation. i denotes the index after the first permutation, while j denotes the index after the second permutation. N_{CBPS} denotes the number of coded bits in a single OFDM symbol. s is determined by the number of coded bits mapped per sub-carrier, N_{BPSC}:

$$s = \max \left(\frac{N_{BPSC}}{2}, 1 \right) \tag{4.19}$$

The de-interleaver in the HiperLAN/2 receiver has to do the inverse operations of (4.17) and (4.18). The de-interleaving function in the HiperLAN/2 receiver can be performed with a convolutional de-interleaver instead of using a block de-interleaver. Block interleavers are considered to be a particular case of convolutional interleavers [42, 44, 91].

The interleaver functionality is not considered part of the baseband processing in the remainder of this chapter. Therefore, the implementation of the de-interleaver on reconfigurable hardware is not discussed.

Channel decoding

Forward Error Correction (FEC) coding is applied in HiperLAN/2. All transport channels in the MAC frame are separately encoded in the HiperLAN/2 transmitter. All bits in the transport channels are encoded with the same convolutional mother code, but the encoded bits are punctured depending on the transport channel. The transport channels during the DL, UL and DiL phases are processed by the channel encoder as a whole. The BCH, FCH and ACH transport channels are individually processed by the encoder.

The convolutional mother code, used in HiperLAN/2, is of rate $R = 1/2$ with constraint length $k = 7$. The generator polynomials of the mother code are $(133, 171)$ [36]. Depending on the employed physical mode, the convolutional mother code is punctured to achieve the according coding rates of the different physical modes from Table 4.3. Three different coding rates are applied in HiperLAN/2. Therefore, three puncturing rules have been applied in the encoder. These puncturing rules are referred to as *rate dependent puncturing*. Moreover, *rate independent puncturing* has been applied in the HiperLAN/2 encoder. The rate independent puncturing is performed prior to the rate dependent puncturing. The rate independent puncturing is applied to the tail bits, which force the convolutional encoder to return to the *zero* state.

In a HiperLAN/2 receiver, channel decoding is typically done by Viterbi decoding [112]. Chapter 6 is completely dedicated to channel decoding techniques.

De-scrambling

Scrambling is applied in the HiperLAN/2 transmitter to randomize the data bits. The randomization is done to avoid bit sequences of consecutive *ones* or *zeros* in the data bit stream. The scrambler is implemented using the generator polynomial:

$$X^7 \oplus X^4 \oplus 1, \tag{4.20}$$

with X the state of the scrambler. The same scrambler polynomial is applied for generating the pilot values in the HiperLAN/2 OFDM signal. The scrambler structure is used both to scramble the transmitted data and to de-scramble the received data.

4.3.3 HiperLAN/2 receiver implementation

In this section we describe how the computationally intensive parts of the baseband processing in the HiperLAN/2 receiver are implemented on reconfigurable hardware. The coarse-grained reconfigurable MONTIUM Tile Processor (TP) [54, 55] is used as target architecture for mapping the baseband Digital Signal Processing (DSP) algorithms. The physical layer of the HiperLAN/2 receiver is implemented on a General Purpose Processor (GPP) in combination with several MONTIUM TPs. Figure 4.13 shows the functional baseband processing blocks in the receiver that are implemented on the System-on-Chip (SoC)[5]. The solid arrows in Figure 4.13 indicate the data, which is processed in consecutive processing tiles. The input consists of data samples, which form OFDM symbols. The output of the baseband processing part results in encoded channel bits. The baseband processing functions are supported by a GPP, which is used for control purposes. The control streams between the processing tiles are shown as dashed arrows in Figure 4.13. The data streams between the processing tiles are mapped on channels of the Network-on-Chip (NoC). The NoC provides the on-chip communication network in the SoC.

Irregular tasks, which are outside the algorithm domain of the MONTIUM, are performed in software (i.e. on the GPP). The irregular processes in the HiperLAN/2 receiver are the channel estimation functions, like frequency offset estimation and computation of equalization coefficients. Because of its relatively large coherence time, the channel estimates have to be updated only once per MAC frame, i.e. once per 2 ms. Table 4.5 shows the results of partitioning the receiver's functionality over the MONTIUM TPs and the GPP.

OFDM symbol frequency offset correction

The frequency offset correction is implemented on one MONTIUM tile. During correction every complex-number sample is multiplied with the frequency offset correction factor. This correction factor is determined with a Look-up Table (LUT) based on the estimated frequency offset. The frequency offset is estimated in software by the GPP once per MAC

[5] The implementation of the prefix removal function on the MONTIUM TP is not covered in this thesis. A mapping of the function has been discussed in [113].

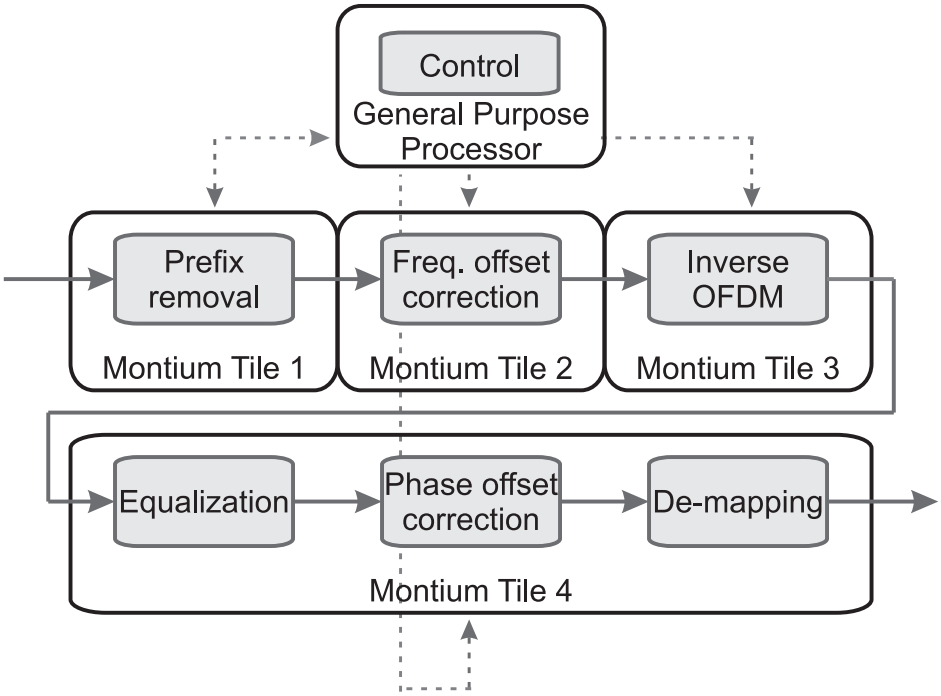

Figure 4.13: *HiperLAN/2 receiver mapped on a tiled reconfigurable SoC.*

frame. One OFDM symbol, containing 64 complex-number samples, can be corrected in 67 clock cycles when implemented on the MONTIUM.

A summary of the implementation properties for frequency offset correction implemented on the MONTIUM is given in Table 4.6. The amount of instruction contexts in the MONTIUM is an indication for the number of MONTIUM decoder instructions. The table only summarizes the number of clock cycles required for data processing. The communication overhead[6], e.g. loading data into the MONTIUM, is not taken into consideration in Table 4.6.

The implemented frequency offset correction is performed according to (4.11). Every received complex-number sample is rotated with the accumulated phase offset, which is caused by frequency offset. The accumulated phase offset is calculated for every sample based on the estimated phase offset per sample caused by frequency offset. The slope of the graph in Figure 4.8 indicates the phase offset per sample, which is caused by frequency offset.

The rotation for every complex-number sample is determined on a sample basis. First of all, the accumulated phase offset is calculated. The accumulated phase offset gives the

[6] The OFDM symbol frequency offset correction is implemented on the MONTIUM according to the block communication principle. However, the implementation can be adapted for streaming communication easily.

Table 4.5: *Reconfigurable hardware/software partitioning of the HiperLAN/2 functionality.*

	Implemented in	Block size	Multiplies per MAC frame	Additions per MAC frame
Determine frequency offset	software	32	64	64
Determine equalizer coefficients	software	52	0	0
Prefix removal	-	80	-	-
Frequency offset correction	MONTIUM	64	127 744	95 309
Inverse OFDM	MONTIUM	64	383 232	574 848
Equalizer, Phase offset, De-mapper	MONTIUM	52	203 184	104 082

address in the LUT, which contains the complex rotation factors. The memory address defines the phase of the rotation factor. Multiplication of the received sample with the rotation factor yields the sample that is corrected for frequency offset. By using two parallel LUTs, the LUT performs the translation from the polar coordinates to the cartesian coordinates system. One LUT contains the *cosine* information whereas the other LUT contains the *sine* information:

$$\widetilde{rotation} = \cos(\phi) + j\sin(\phi) \tag{4.21}$$

The estimated frequency offset, which is determined in the GPP, is actually determined as a phase offset per sample. This phase offset per sample takes values in the range

Table 4.6: *Computational requirements for frequency offset correction implemented on the* MONTIUM *per OFDM symbol.*

# clock cycles	67
# interconnect contexts	1
# memory contexts	4
# ALU contexts	2
# register contexts	4

$[-\pi, \pi\rangle$. In the MONTIUM the normalized phase offset, $\varphi_n = \varphi/\pi$, is used to determine the frequency offset correction coefficients because the normalized phase offset takes values in the range $[-1, 1\rangle$, which perfectly fits the fixed-point number representation used in the MONTIUM.

The normalized phase offset represents the rotation used to correct for frequency offset. The normalized phase is used as the look-up address. In the MONTIUM implementation two LUTs are used to generate the complex rotation factor as given in (4.22). Both the *real* and *imaginary* part of the rotation are determined using separate LUTs, with $\varphi_n = \varphi/\pi$:

$$e^{j\varphi_n} = \cos(\varphi) + j\sin(\varphi) \tag{4.22}$$

In the MONTIUM-based frequency offset correction implementation, ALU1 through ALU4 are involved in the complex multiplication. Every received sample is multiplied with the complex rotation factor. The received input samples are stored in memories MEM1 (real part) and MEM3 (imaginary part). The real and imaginary part of the frequency offset corrected samples are stored in MEM2 and MEM5, respectively. MEM9 and MEM10 are used in LUT mode to generate the rotation factor. The memory address for the LUTs is determined by ALU5, which keeps track of the total accumulated phase over the received samples of the OFDM symbol. The phase offset per sample due to frequency offset, which is determined by the frequency offset estimator, is added to the already accumulated phase for every sample. Consequently, the accumulated phase is linearly increasing or decreasing with time as shown in Figure 4.8. Figure 4.14 shows a detailed diagram of the implemented main loop on the MONTIUM during the frequency offset correction.

The locality of reference principle is directly applied to the mapping of the frequency offset correction on the MONTIUM. The mapping of the functionality to the Arithmetic Logic Units (ALUs) and memories is chosen in such a way that the data is closely located near the data processing units. Storing the data close to the data processing units avoids communication overhead, which is the main motivation to apply the locality of reference principle.

Basically, the frequency offset correction on the real part of the complex-number samples is applied by ALU1 and ALU2. ALU3 and ALU4 are involved in the imaginary part. According to the locality of reference principle, the output of the frequency offset correction can best be stored in MEM1, MEM2, MEM3 or MEM4 for the real part and MEM5, MEM6, MEM7 or MEM8 for the imaginary part. Therefore, MEM2 and MEM5 are used to store the real and imaginary part of the frequency offset corrected samples, respectively. These memory units are chosen to store the results, because the local interconnects between the ALUs and memories can be used. Using the local interconnects instead of the global interconnects, results in a more energy efficient mapping. Local interconnects only invoke local signal activity within a Processing Part (PP), whereas the global interconnects result in signal activity in the entire Processing Part Array (PPA).

Since ALU5 is applied as address generator for the LUTs and because it is common practice to use the local interconnects for energy efficiency, the LUTs have been situated

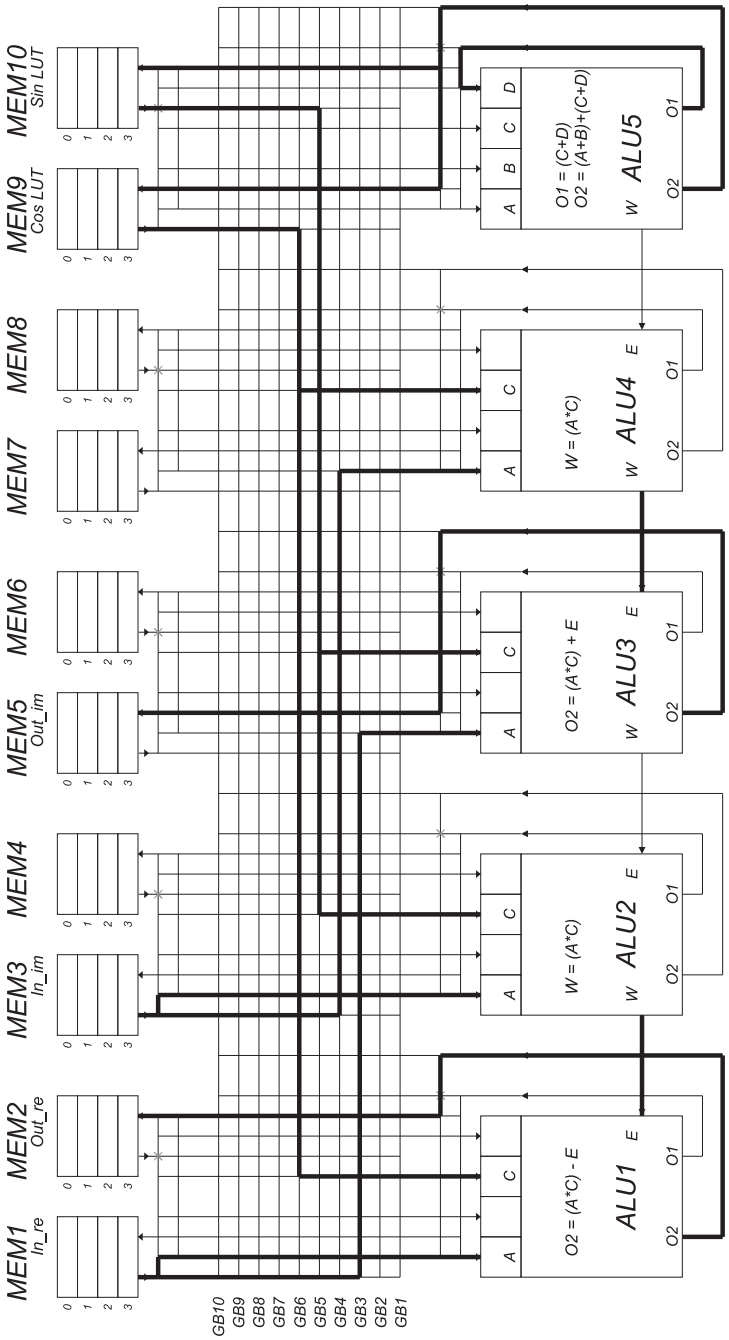

Figure 4.14: *Main loop of frequency offset correction mapped on the* MONTIUM.

Table 4.7: *Computational requirements for FFT-64 processing on the* MONTIUM.

# clock cycles	204
# interconnect contexts	17
# memory contexts	21
# ALU contexts	1
# register contexts	2

in the memories MEM9 and MEM10. The look-up rotation factor is routed via the global interconnect to ALU1 through ALU4, that are involved in the frequency offset correction.

Inverse OFDM

The inverse OFDM function in HiperLAN/2 is done using an FFT-64 operation. This operation is performed by radix-2 butterflies, which can be efficiently mapped on the MONTIUM. For FFT-64 signal processing, $\left(\frac{64}{2}\right)\log_2(64)$ *complex* multiplications have to be calculated. One complex multiplication can be implemented with 4 *real* multiplications, which can be calculated in the MONTIUM in a single clock cycle. On the MONTIUM architecture the FFT-64 can be performed in $\left(\frac{64}{2}+2\right)\log_2(64)$ clock cycles. The 2 extra clock cycles per FFT stage are due to pipelined memory accesses of the MONTIUM memories [54]. In general, an N-point FFT kernel, with N a *power of 2*, is performed in $\left(\frac{N}{2}+2\right)\log_2(N)$ clock cycles on the MONTIUM architecture.

The FFT algorithm uses ALU1 through ALU4 for Multiply-Accumulate (MAC) operations. All 10 memories of the MONTIUM are used during FFT computation. Two memories, MEM9 and MEM10, are used to store the real and imaginary part of the twiddle factors, respectively. The remaining memories in the MONTIUM, MEM1 to MEM8, are used to stored the (intermediate) results of the FFT algorithm. The size of the local memories in the MONTIUM determines the maximum FFT-size that can be supported; with a memory depth of 1024 addresses, the maximum supported FFT-size is $N_{max} = 2048$.

A summary of the MONTIUM-based FFT-64 implementation is given in Table 4.7. The amount of instruction contexts in the MONTIUM is an indication for the number of MONTIUM decoder instructions. From Table 4.7 can be concluded that FFT processing is especially complex with its memory management. The information in Table 4.7 only covers the data processing and not the communication overhead[7].

The FFT implementation on the MONTIUM is available in two 'flavours': *streaming* or *block* communication. The streaming communication alternative of the FFT algorithm achieves higher throughput than the block communication implementation. In streaming communication mode, data words are processed immediately as they become available from the on-chip network. The results after processing are again immediately sent over the on-chip network. Conversely, during block communication mode, the data words

[7] The FFT-64 operation is implemented on the MONTIUM according to the block communication principle. However, the implementation can be adapted for streaming communication easily.

are first stored in the local memories of the MONTIUM before they are processed. After data processing, the results are stored in the local memories of the MONTIUM. Via the on-chip network the data is stored in or retrieved from the local memories, resulting in extra communication time (i.e. communication overhead).

OFDM symbol equalization

The Channel Impulse Response (CIR) is assumed to be time-invariant during one complete MAC frame, because the coherence time of the wireless channel (20 ms) is much longer than the duration of a MAC frame (2 ms). Therefore, the equalizer is trained only once per MAC frame. The equalization coefficients are determined by dividing the predefined preamble *C* values, which are the known training values, by the received complex-number training values. The computation of the equalization coefficients is done on the GPP. This function is implemented in software (i.e. on the GPP) because it occurs infrequently, i.e. once every 2 ms, and because the MONTIUM is not optimized for complex-number divide operations. However, it is possible to do divide operations using the LUT functionality of the MONTIUM.

The complex-number equalization coefficients are stored in the local memories of the MONTIUM. During equalization each of the 52 complex-number sub-carrier values is multiplied with its corresponding complex-number equalization coefficient. The MONTIUM computes a complex-number multiplication in a single clock cycle using 4 ALUs. The 48 equalized sub-carriers containing data are stored in the local memories of the MONTIUM. The 4 pilot sub-carriers are not stored in memory, but used to determine the phase offset.

Figure 4.15 shows a detailed picture of the implemented main loop on the MONTIUM during OFDM symbol equalization.

The real and imaginary input samples are stored in memory MEM1 and MEM3, respectively. These input samples are multiplied with the complex-number equalization coefficients in ALU1 through ALU4. The real and imaginary part of the equalization coefficients are stored in memory MEM5 and MEM7, respectively. These coefficients are infrequently updated by the GPP by means of writing the new coefficients in the memories. Basically, the OFDM symbol equalization on the real part of the complex samples is applied by ALU1 and ALU2, whereas ALU3 and ALU4 are involved in the imaginary part. The equalized samples at the output of ALU1 and ALU3 are stored in memory MEM2 and MEM6, respectively. Analogous to frequency offset correction, the mapping of the equalization on the MONTIUM is based on the locality of reference principle.

OFDM symbol phase offset correction

After equalization the 4 pilot values are used to determine the phase offset correction factor. The phase offset correction factor is determined in the MONTIUM, because the phase offset can vary for every OFDM symbol and so the correction factor has to be determined on an OFDM symbol basis (once every 4 μs). Hence, estimation of the phase

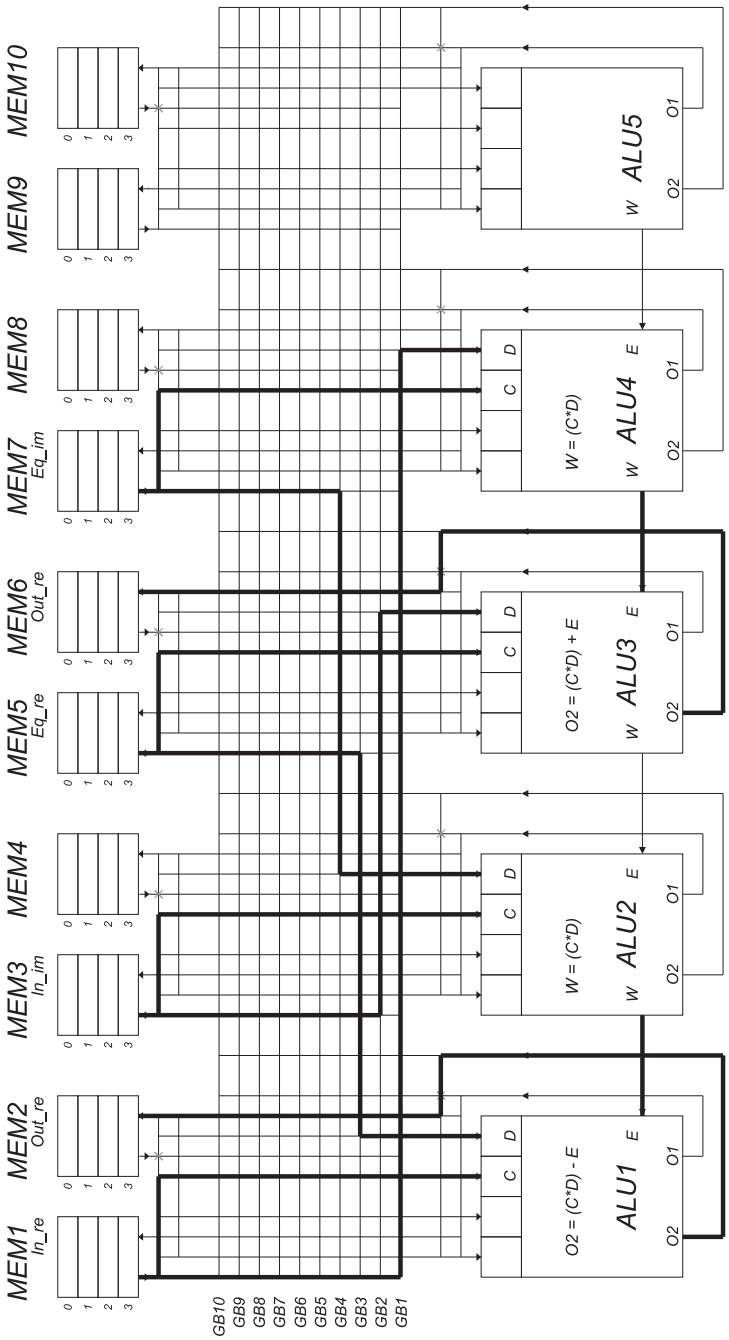

Figure 4.15: *Main loop of equalization mapped on the* MONTIUM.

offset correction factor in software (i.e. on the GPP) would create large communication overhead between the GPP and the MONTIUM tile.

The phase offset of the sub-carriers is estimated in the MONTIUM by calculating the mean rotation of the pilot carriers compared to their expected values, according to (4.15). According to (4.16), the inverse of the estimated phase offset has to be multiplied with the equalized sub-carrier values. The inverse operation is implemented in the MONTIUM with LUT functionality.

The reference signal transmitted in the pilot carriers of the OFDM signal is defined as a cyclic sequence of 127 elements [36]. The pilot value to be received is chosen in accordance with the OFDM symbol number. Hence, the pilot carrier values are repeated for every 127 OFDM symbols. This reference sequence of pilot values is stored in a local memory of the MONTIUM, i.e. MEM8.

Phase offset correction invokes also a complex multiplication, like equalization. As a consequence the equalizer and phase offset corrector use the same functionality of the MONTIUM. Hence, the same configurations of the ALUs used for OFDM symbol equalization are reused. The equalized samples are loaded from MEM2 and MEM6 and used as input for the phase offset correction. The phase offset correction is applied in ALU1 and ALU2 for the In-phase component of the OFDM signal and in ALU3 and ALU4 for the Quadrature component.

Figure 4.16 depicts the main loop of the phase offset correction mapped on the MONTIUM. The phase offset correction is combined with the QAM symbol de-mapping in a pipelined manner. The mapping of the phase offset estimation, which is performed before phase offset correction, is not shown.

OFDM symbol de-mapping

The corrected complex-number samples are translated into a bit stream in a pipelined, parallel manner. Hard-decision de-mapping is implemented using LUT functionality. A parameterizable de-mapper is implemented, which supports QPSK, QAM-16 and QAM-64 modulated signals by only changing the contents of the LUT in the memory of the MONTIUM.

The de-mapping of the received OFDM signal is performed in PP5, containing ALU5 and MEM9 and 10. In ALU5, the In-phase and Quadrature components of the corrected OFDM signal are combined in order to generate the memory address in the de-mapping LUT. The de-mapping LUT is situated in MEM9. The de-mapped bit sequence is stored in MEM10.

Figure 4.16 depicts the main loop of the pipelined phase offset correction and de-mapping mapped on the MONTIUM.

The OFDM symbol equalization, phase offset correction and hard-decision de-mapping are completely integrated in one MONTIUM. A summary of the pipelined implementation[8] is given in Table 4.8. The amount of instruction contexts in the MONTIUM

[8] The OFDM symbol equalization, phase offset correction and de-mapping is implemented on the MONTIUM according to the block communication principle. However, the implementation can be adapted for streaming communication easily.

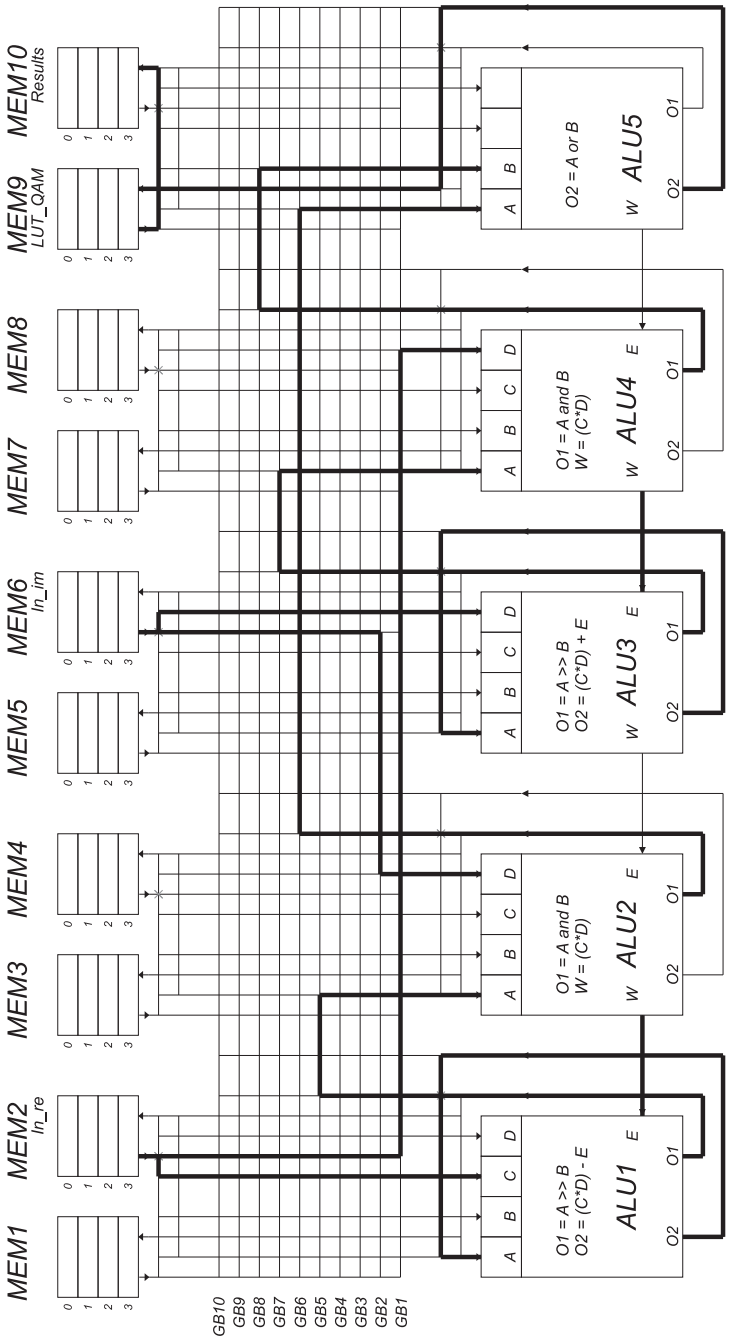

Figure 4.16: *Main loop of the pipelined phase offset correction and de-mapping mapped on the* MONTIUM.

Table 4.8: *Computational requirements for combined OFDM symbol equalization, phase offset correction and hard-decision de-mapping on the* MONTIUM *per OFDM symbol.*

# clock cycles	110
# interconnect contexts	2
# memory contexts	5
# ALU contexts	3
# register contexts	13

Table 4.9: *Properties of the HiperLAN/2 receiver implementation.*

		Frequency offset correction	Inverse OFDM	Equalizer, Phase offset, De-mapper
Execution time	[cycles]	67	204	110
Communication time	[cycles]	128	116	$<$100
Minimum system clock with streaming communication	[MHz]	17	51	28
Minimum system clock with block communication	[MHz]	49	80	53
Minimum processor clock with block communication (@ 100 MHz)	[MHz]	25	72	37
Configuration size	[bytes]	274	946	576
Configuration time	[cycles]	137	473	288

is an indication of the number of MONTIUM decoder instructions. The information in Table 4.8 only concerns the data processing and not the communication overhead.

HiperLAN/2 receiver implementation results

The HiperLAN/2 receiver is implemented on a tiled reconfigurable SoC according to Figure 4.13. The tiled reconfigurable SoC template is introduced in Chapter 3. The implementation characteristics of the DSP kernels in the HiperLAN/2 receiver are summarized in Table 4.9. The table shows the impact of the communication overhead in the NoC, which can be performed by streaming or block mode communication. Furthermore, the configuration overhead of the DSP kernels implemented on the coarse-grained MONTIUM is given.

Configuration The configuration sizes of the MONTIUM TPs are small for the different functions (Table 4.9). MONTIUM Tile 3 (see Figure 4.13), on which the inverse OFDM is performed, requires the largest configuration size. The configuration of Tile 3 contains less than 1 kB of configuration data. The configuration data is written into the configuration memory of the MONTIUM in about 500 clock cycles as every clock cycle 16-bits are written. Suppose that the MONTIUM is configured at a clock frequency of 100 MHz, then Tile 3 can be configured in 4.73 μs. Notice that the maximum radio turn-around time[9] of the HiperLAN/2 communication system is 6 μs [35], so the implemented HiperLAN/2 receiver can be considered as a real-time dynamically reconfigurable receiver.

Frequency scaling Since the DSP kernels in the HiperLAN/2 receiver are mapped on different MONTIUM TPs in the SoC architecture, frequency scaling can be easily applied. Frequency scaling in combination with voltage scaling is an important means to control the power consumption of embedded systems. The idea of dynamic voltage scaling is to keep the supply voltage as low as possible. The maximum operating frequency is tightly coupled to the supply voltage level. This means that by down scaling the clock frequency of hardware, the supply voltage can be lowered as well, resulting in a cubic decrease of the power consumption (Chapter 3).

All DSP operations in the physical layer of the HiperLAN/2 communication system are performed on OFDM symbols. Every OFDM symbol has a time period of 4 μs. So, it should be assured that each 4 μs a new OFDM symbol can be processed by the receiver.

Typically, the clock frequency of the NoC is fixed and the clock frequency of the processor tiles can be varied. In case of block mode communication and under the assumption that the clock frequency of the NoC is fixed at 100 MHz, this results in the situation that the clock frequency of the MONTIUM tile for frequency offset correction has to be at least 25 MHz.

In case of block mode communication and when both the clock frequency of the NoC and the reconfigurable processor tile can be adapted, the situation exists where the clock frequency of the entire SoC (i.e. processor tiles and NoC) can be scaled to its minimum value. The minimum clock frequency of the system would in this case be reduced to 49 MHz for frequency offset correction. However, this situation is fairly unlikely to happen because managing the adaptable clock frequency of the common NoC is complex.

Introducing the streaming communication mode variant of the DSP kernels in the HiperLAN/2 receiver provides a situation wherein the data processing and communication are performed in parallel. In this case the data words are processed immediately as they become available from the on-chip network. Consequently, the communication time is not a bottleneck. For example, the data processing for frequency offset correction is performed during 67 clock cycles and needs to complete in 4 μs. Hence, the minimum clock frequency of the MONTIUM tile is 17 MHz for frequency correction. In case of streaming communication, the clock frequency of the NoC should be at least 17 MHz.

[9] The radio turn-around time indicates the time to switch from transmit to receive mode in a communication transceiver, and vice versa.

Typically, the clock speed of the NoC is fixed to the maximum operating frequency of the processing tiles.

Energy consumption Using power estimation tooling, the dynamic power consumption of a typical MAC operation in the MONTIUM was estimated to be about $0.5\ mW/MHz$, realized in a $0.13\ \mu m$ CMOS technology. The area of a single MONTIUM TP is about $2\ mm^2$ in this technology [54].

4.3.4 HiperLAN/2 implementation verification

The implemented HiperLAN/2 receiver is tested by means of hardware / software co-simulations. A reference model of the HiperLAN/2 receiver was implemented in MATLAB (using 64-bit floating-point computations) in order to verify the MONTIUM implementation. Simulations were performed with both the reference model and the MONTIUM implementation of the HiperLAN/2 receiver. In each simulation a downlink (DL) burst, containing 500 OFDM symbols, was received. In the DL burst, the first two OFDM symbols of the MAC frame contain preamble *C* (Figure 4.6). This preamble *C* was used to estimate the frequency offset and the equalization coefficients. The OFDM symbols were QAM-16 modulated in the experiments, so in each experiment $95\,616$ bits were received.

Hardware / software co-simulation framework

Figure 4.17 shows a co-simulation environment comprising MATLAB [71] and MODELSIM [73], which is used for functional simulations. Using this co-simulation environment, we can simulate the baseband functionality of the HiperLAN/2 communication system in software (i.e. using MATLAB) and partly in hardware (i.e. using MODELSIM).

All functions of the HiperLAN/2 receiver that will be implemented in the GPP of the final SoC implementation, are simulated in MATLAB code. The hardware simulations are performed by MODELSIM, which simulates the RTL code of the MONTIUM TP.

Wireless channel model

To verify more realistic scenarios, the HiperLAN/2 receiver was simulated together with different wireless channels. The wireless channels introduce multipath and Additive White Gaussian Noise (AWGN). In [15] five types of channel models for HiperLAN/2 are given, which are derived from measurements in typical indoor and outdoor environments. These channel models are defined for standardized system analysis [72]. The parameters of the channel models are shown in Table 4.10. Figure 4.18 shows the wireless channel model used in our experiments. The multipath channel can be modelled by means of a tapped-delay model, where each tap represents a path of the wireless signal that traverses from transmitter to receiver.

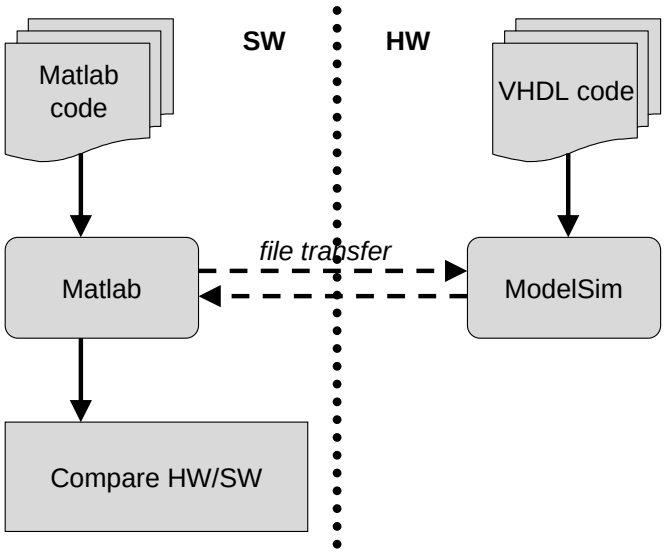

Figure 4.17: *The hardware/software co-simulation environment.*

Introducing all channel effects, as depicted in Figure 4.18, results in the received baseband signal before sampling (4.5):

$$\tilde{r}(t) = \left[\tilde{s}(t) \cdot e^{-j2\pi f_{\Delta} t + \Theta} * \tilde{h}(t,\tau)\right] + \tilde{n}(t)$$

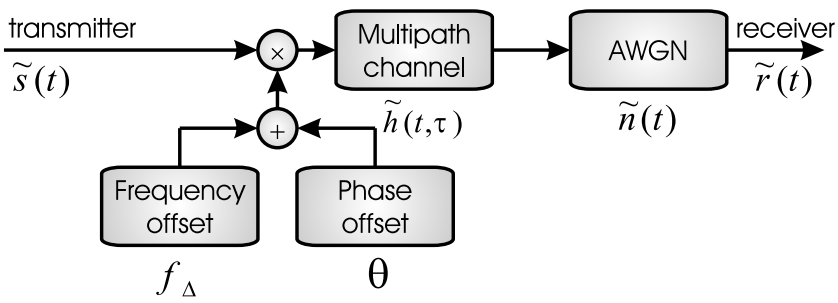

Figure 4.18: *The HiperLAN/2 wireless channel model.*

where the parameters of the channel model are implemented as follows:

- $\tilde{s}(t)$ denotes the originally transmitted baseband signal;
- $\tilde{h}(t,\tau)$ denotes the CIR;
- $\tilde{n}(t)$ defines the Additive White Gaussian Noise (AWGN);
- f_Δ denotes the frequency offset between transmitter and receiver ($f_\Delta = f_{Tx} - f_{Rx}$);
- Θ denotes the common phase offset.

The time-variant CIR defines the multipath contribution [92, 123]:

$$\tilde{h}(t,\tau) = \sum_{l=0}^{L-1} \tilde{\alpha}_l(t)\,\delta(\tau - \tau_l), \tag{4.23}$$

where

- τ_l defines the delay of the taps;
- $\tilde{\alpha}_l$ denotes the complex-number amplitude of the taps;
- δ is the Dirac pulse;
- L gives the number of taps in the delay-line.

The channel realizations in Table 4.10 are modelled with $L = 18$ taps. The taps are assumed to be statistically independent and complex Gaussian distributed with zero mean. Except for the first tap, which can have a Ricean distribution for channel type D, all taps have Rayleigh fading statistics. Additionally, the signal in each path experiences Doppler shift due to motion in the wireless environment. Such motion leads to frequency shifts in the signal's spectrum of individual reflected signals. These frequency shifts can be seen as time varying phase shifts. At the antenna of the mobile receiver many reflected signals arrive, all with different phase shifts. Thus, their relative phases change all the time and the amplitude of the resulting composite signal is affected. So, the Doppler effects determine the rate at which the amplitude of the resulting composite signal changes. The maximum speed of the terminal is assumed to be $3\ m/s$, which results in a maximum Doppler shift of $52\ Hz$. According to (4.13), the channel is considered constant during the experiments (i.e. during the time of 500 OFDM symbols).

To speed up the simulation time, the simulation sampling rate is equal to the HiperLAN/2 baseband sampling rate of 20 MHz instead of the transmission rate of 5 GHz. The wireless channel models the frequency offset and phase offset explicitly. The sampling clocks of the HiperLAN/2 transmitter and receiver are synchronized in our experiments, which means that there does not exist OFDM symbol drift (e.g. jitter). The MATLAB/SIMULINK HiperLAN/2 transmitter model described in [57] is used to generate the HiperLAN/2 bursts.

Table 4.10: *Parameters of typical HiperLAN/2 channels [15].*

Channel type	τ_{rms} $[ns]$	Environment	Line-of-sight
A	50	office	no
B	100	open space / office	no
C	150	large open space	no
D	140	large open space	yes
E	250	large open space	no

HiperLAN/2 performance simulations

The wireless channel model is used to verify the correctness of the implemented HiperLAN/2 receiver.

The Bit Error Rate (BER) curves in Figure 4.19 and 4.20 show the performance of the implemented HiperLAN/2 receiver under different wireless channel conditions. The results are shown for both the MONTIUM-based receiver (labelled "Montium") and the floating-point reference model (labelled "Reference"). The BER of the HiperLAN/2 receiver is determined at the output of the hard-decision de-mapper.

The HiperLAN/2 receiver was simulated using both an "A" and "E" channel with multipath and using an AWGN channel without multipath. The AWGN channel was simulated in order to verify whether the basic OFDM functionality of the HiperLAN/2 receiver is functioning correctly. During AWGN channel simulation, the equalization coefficients are initialized with real constants. Hence, the results of the AWGN curves in Figure 4.19 and 4.20 are not influenced by the channel estimation and equalizer functionality of the receiver. Multipath channel "A" and "E" are chosen for verification because these channel types represent two extreme cases in their channel characteristics. The multipath channel type "A" and "E" simulations verify the correctness of the channel equalization functionality in the receiver.

The curves of the MONTIUM receiver and the reference model in Figure 4.19 and 4.20 show hardly any difference in BER. The BER gap between reference model and MONTIUM implementation increases when the SNR becomes better. Possibly, for bad channel conditions (i.e. low SNR) the BER is mostly determined by the wireless channel. Whereas, for good channel conditions (i.e. high SNR) the limited 16-bit fixed-point precision of the hardware has a significant influence on the BER.

The effects of the analog RF front-end on the received signal were explicitly modelled. A frequency offset between the LOs of the transmitter and receiver was simulated. The BER curves in Figure 4.21 depict the simulation results of the HiperLAN/2 receiver in wireless channel type "A" conditions with variable frequency offset and SNR settings. In case of an ideal "A" channel, the SNR of the received signal during simulation was assumed infinite (i.e. no noise was added in the wireless channel, only multipath was considered). The results in Figure 4.21 show that the implemented HiperLAN/2 receiver can correct for frequency offset up to 150 kHz, which was already expected accord-

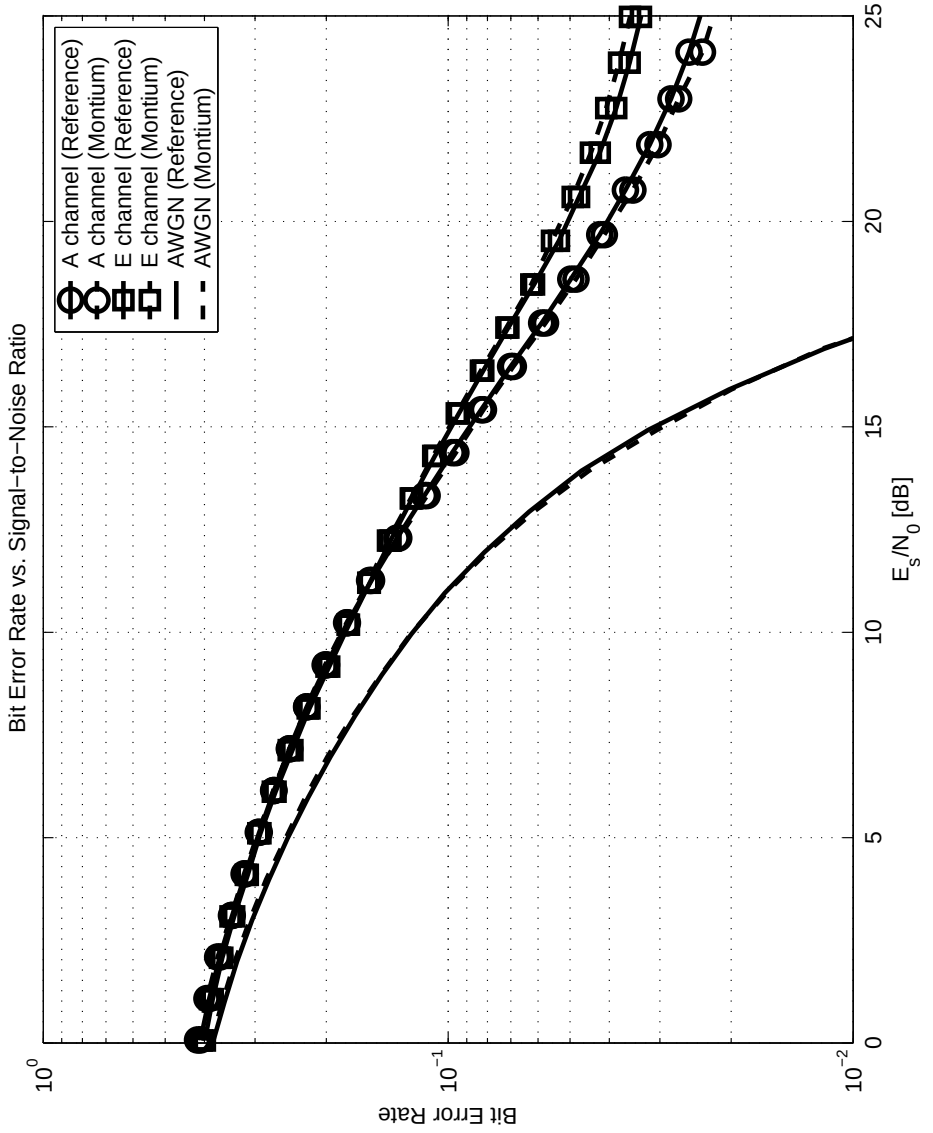

Figure 4.19: *The BER before error correction under different wireless channel conditions without frequency offset.*

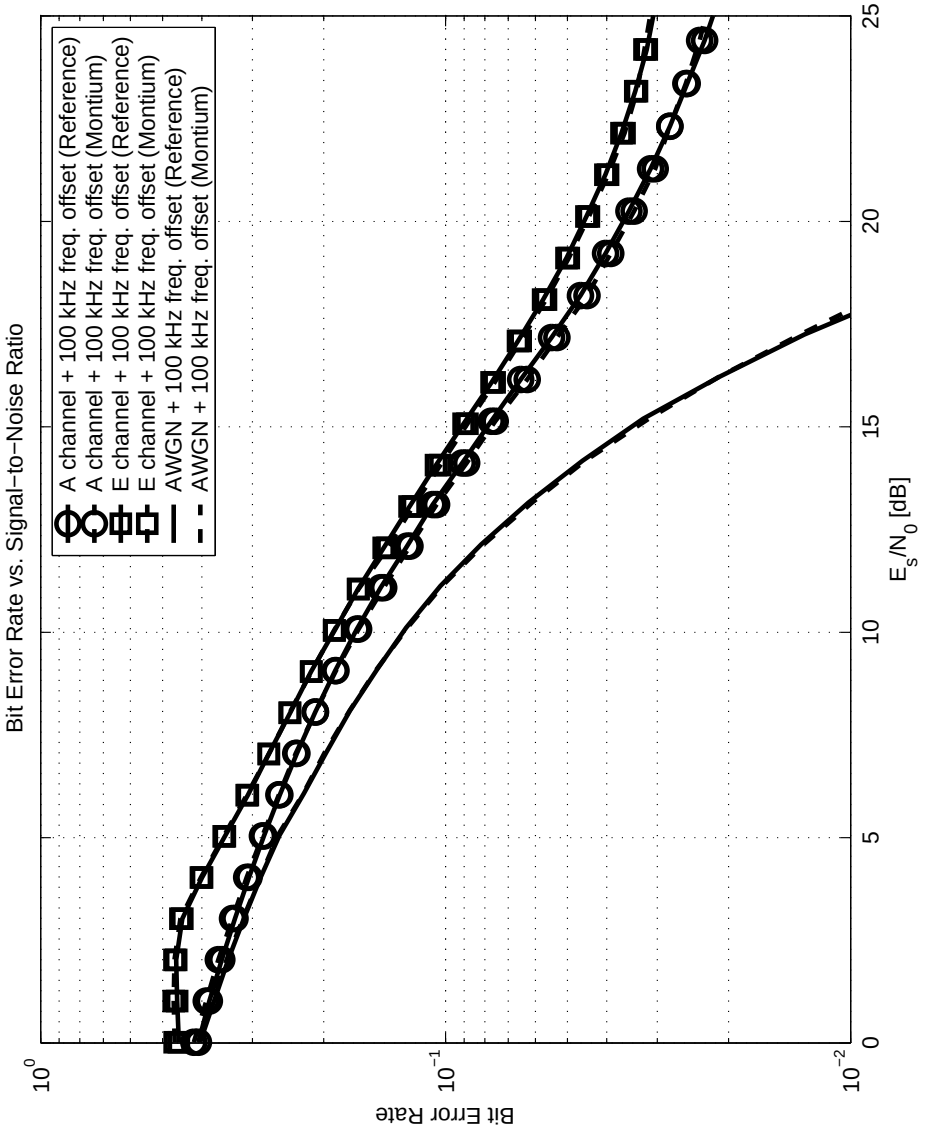

Figure 4.20: *The BER before error correction under different wireless channel conditions with frequency offset.*

ing to (4.10) [15]. Typically, the frequency offset correction in a HiperLAN/2 receiver can be improved by applying two steps. Firstly, coarse frequency offset correction is applied, which estimates the frequency offset roughly and aligns the OFDM spectrum. Secondly, fine frequency offset correction removes the remaining frequency offset from the received signal. The preamble *C* based frequency offset estimation is well suited for the fine frequency offset correction phase.

The phase offset between the LOs of the transmitter and receiver is another distortion induced by the analog RF front-end. Additionally, the frequency offset correction function is also able to cause common phase offset. The phase offset causes a phase shift in the constellation of the received signal. Under different channel conditions we have explicitly modelled the enforced phase offset. The BER curves in Figure 4.22 depict the simulation results of the HiperLAN/2 receiver in wireless channel type "A" conditions with variable phase offset and SNR settings. The SNR of the received signal during simulation was assumed infinite for the ideal "A" channel (i.e. only multipath was considered and no noise was added in the wireless channel). The simulation results show that the implemented HiperLAN/2 receiver can correct for phase offset. Furthermore, the BER curves show that the reference model and the MONTIUM implementation give almost the same results.

HiperLAN/2 performance

Only the baseband processing of the HiperLAN/2 receiver is implemented on the MONTIUM (in Section 4.3.3). FEC decoding, like Viterbi decoding, is not explicitly implemented in these experiments. However, since the characteristics of the FEC encoders that are used in the HiperLAN/2 communication system are known [36], the upperbounds for the BER after FEC decoding can be calculated [123, Chapter 7]. These estimations are upperbounds of the convolutional decoding algorithm (e.g. Viterbi decoder). Figure 4.23 and 4.24 show the upperbounds that are calculated for coding rate 9/16 and 3/4 based on the results of Figure 4.19 and 4.20, respectively. These FEC settings in combination with the QAM-16 modulation comply with mode V and VI of the HiperLAN/2 standard, respectively (Table 4.3).

The probability of a *bit error* after error decoding is given in [123, equation (7-16)]:

$$P_b < \sum_{k=d_{free}}^{\infty} c_k P_k, \tag{4.24}$$

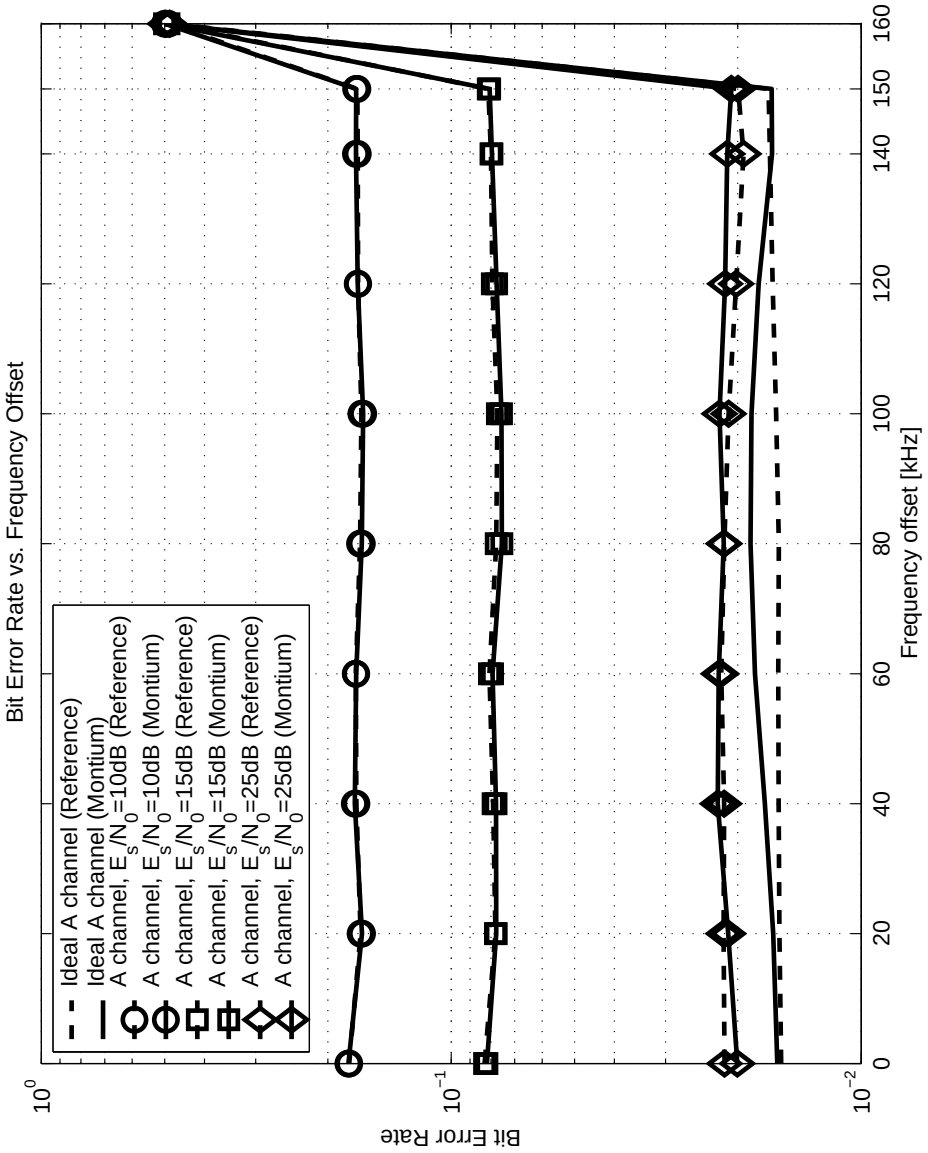

Figure 4.21: *The effect of frequency offset on the BER for different wireless channel settings.*

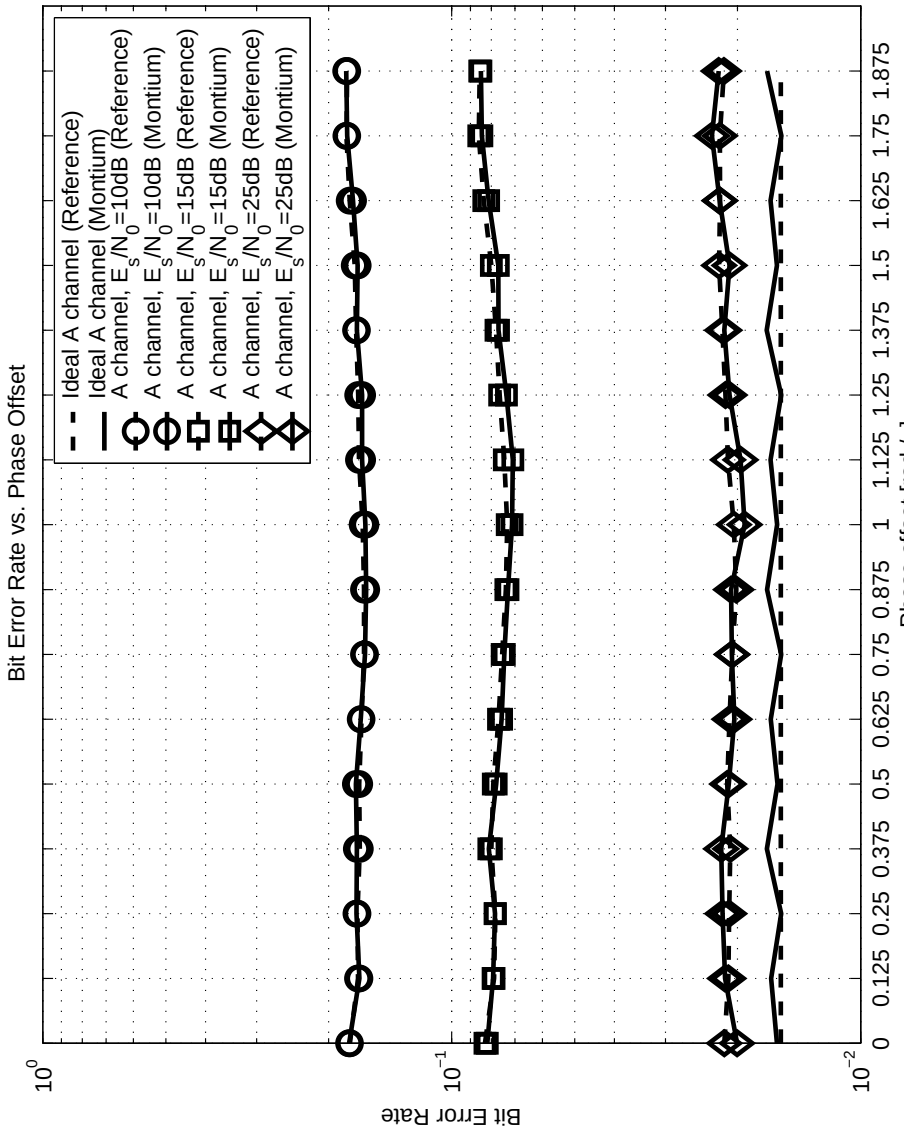

Figure 4.22: *The effect of phase offset on the BER for different wireless channel settings.*

where

- c_k denotes the number of bit errors associated with all paths that are Hamming distance[10] k from the all-zero path;
- P_k defines the probability of incorrectly selecting a path through the trellis with Hamming weight[11] k.

According to [123, equation (7-12)], P_k is defined by:

$$\begin{aligned} P_k &= \sum_{e=(k/2)+1}^{k} \binom{k}{e} p^e (1-p)^{k-e} && (4.25) \\ &+ \frac{1}{2}\binom{k}{k/2} p^{k/2} (1-p)^{k/2} && \text{, for } k \text{ even} \\ P_k &= \sum_{e=(k+1)/2}^{k} \binom{k}{e} p^e (1-p)^{k-e} && \text{, for } k \text{ odd} \end{aligned}$$

p denotes the probability of an incorrectly received bit at the receiver before FEC decoding in (4.25). The probability of incorrectly selecting a path that is Hamming distance k from the all-zero path is a function of the channel and the received SNR. An incorrectly received bit at the receiver before FEC decoding can result in the situation that the FEC decoder selects an incorrect path in the trellis, because the incorrectly received bit sequence is close to another output codeword sequence of the convolutional encoder. Hence, in order to estimate the BER after FEC decoding based on the results of the experiments, p has to be substituted with the BER results before FEC decoding (i.e. the BER curves from Figure 4.19 and 4.20).

Table 4.11 summarizes the associated bit errors of an incorrectly chosen path trough the trellis. The associated bits are given for the mother code used in the HiperLAN/2 communication system [57, 123]. The associated bit errors, c_k, were substituted in (4.24).

Discussion The HiperLAN/2 receiver needs a Bit Error Rate (BER) of $2.4 \cdot 10^{-4}$ in order to reach the minimum defined sensitivity of 10% Packet Error Rate (PER) [36]. The upper bounds on the BER after FEC decoding show that the minimum sensitivity can be met with the MONTIUM implementation. Note that these upper bounds are rough estimates. Moreover, the accuracy of the BER curves in Figure 4.23 and 4.24 for BERs lower than $1.5 \cdot 10^{-2}$ or $2.0 \cdot 10^{-2}$ are questionable because the estimates are based on $3/4 \times 95\,616$ or $9/16 \times 95\,616$ decoded bits for mode V and VI, respectively.

Unlike the HiperLAN/2 receiver in [96], the MONTIUM-based receiver is also simulated under multipath conditions. Without multipath conditions the implemented HiperLAN/2 receiver shows the required performance. However, we discovered that simple ZF equalization is not satisfactory in case of multipath conditions. More robust and complex channel estimation algorithms that are less sensitive to noise should be used instead.

[10] The Hamming distance gives the number of bits that are different between two bit sequences.
[11] The Hamming weight of a codeword is given by the number of non-zero elements in the codeword.

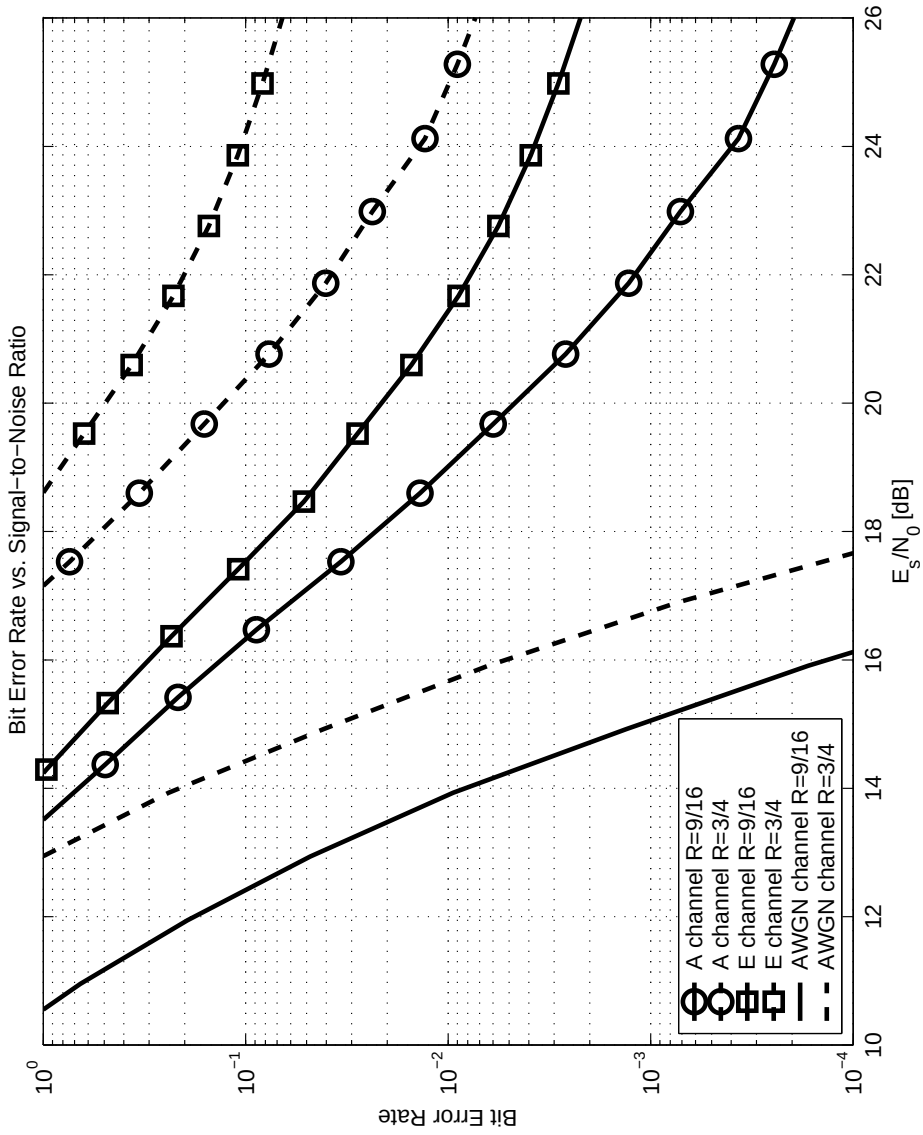

Figure 4.23: *The calculated upperbound of the BER after error correction without frequency offset.*

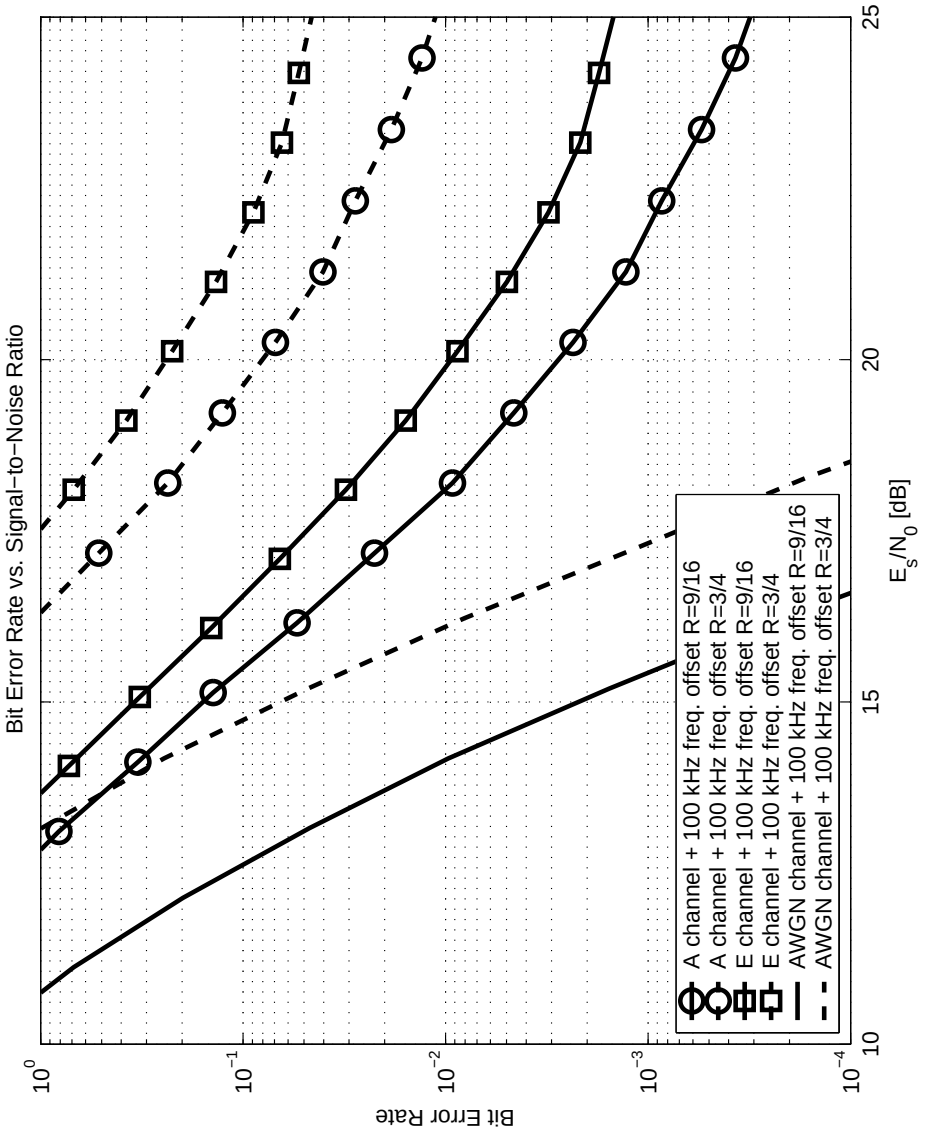

Figure 4.24: *The calculated upperbound of the BER after error correction with frequency offset.*

Table 4.11: *Associated bit errors, c_k, of an incorrect path at Hamming distance k for the HiperLAN/2 channel coder at $R = 1/2,\ 9/16$ and $3/4$ [57, 123].*

Coding rate, R	Free distance, d_f	c_k for k is $d_f \ldots$						
		$+0$	$+1$	$+2$	$+3$	$+4$	$+5$	$+6$
$1/2$	10	36	0	211	0	1404	0	11633
$9/16$	7	2.67	10.33	53.33	297.33	1504	7769	–
$3/4$	5	2	10.9	35.7	117.8	342.3	1172.2	–

4.4 Digital Audio Broadcasting

The Digital Audio Broadcasting (DAB) standard [38] was initiated – in the early 1980s – to replace the analog FM radio services and has already been adapted by many countries all over the world. The DAB system is capable of delivering data and audio services to end users. The source encoding of the audio services uses the MPEG-1 Layer 2 (MP2) Audio Coding scheme for reducing the audio bitrate. The DAB radio system provides audio services with almost CD quality. Four transmission modes are defined for DAB which enable DAB signal transmission over a wide range of operating frequencies under different radio channel conditions.

The techniques of the DAB receiver are outdated compared to state-of-the-art digital broadcasting standards such as DRM and DVB. The DAB working group is working on the DAB+ specification. One of the main revisions for the DAB standard is the addition with High Efficiency AAC (HE-AAC) [39]. Advanced Audio Coding (AAC) is currently the most efficient audio coding scheme and requires a bit rate of 64 $kbps$ to provide good audio quality on stereo stations. The MP2 audio coding scheme requires roughly 192 $kbps$ to achieve similar audio quality.

4.4.1 DAB transport mechanism

The DAB system is capable of delivering data and audio services. Therefore, the DAB signal has to multiplex several digital audio signals with data signals. These audio and data signals are denoted as service components, which can be grouped together to form the final service to the end user. The service components are transmitted in sub-channels. The entire DAB signal, called the DAB *ensemble*, can contain at most 64 sub-channels.

The DAB transport mechanism defines three channels to transmit information. These three channels together form the DAB transmission frame:

- *Synchronization channel*
  The synchronization channel contains training information for frame synchronization and demodulator initialization. The synchronization channel contains two OFDM symbols in every transmission mode. The first OFDM symbol is referred to

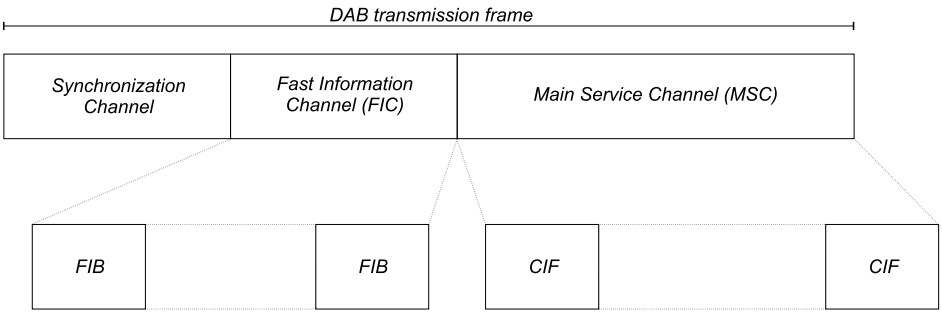

Figure 4.25: *Transmission mode independent description of the DAB transmission frame.*

as the *null* symbol, which is used for frame synchronization. The second OFDM symbol of the synchronization channel is the phase reference symbol. This phase reference symbol is used to initialize the differential demodulator. Unlike HiperLAN/2, the DAB signal does not carry any pilot information except for this phase reference symbol in the synchronization channel;

- *Fast Information Channel (FIC)*
  The FIC contains the configuration information that the receiver needs to demultiplex the various sub-channels. The FIC is a non-time-interleaved data channel with fixed equal error protection. The essential information in the FIC is the Multiplex Configuration Information (MCI), which contains information on the multiplex structure. The MCI also contains information about a reconfiguration of the multiplex;

- *Main Service Channel (MSC)*
  The MSC is used to carry the audio and data service components. The MSC is a time-interleaved data channel divided into a number of sub-channels, which are individually convolutionally coded; every sub-channel can be protected individually with equal or unequal error protection. The organization of the sub-channels and service components is called the multiplex configuration.

The organization as well as the length of a DAB transmission frame depends on the transmission mode as seen in Table 4.12. The Fast Information Block (FIB) and the Common Interleaved Frame (CIF) provide transmission mode independent data transport packages associated with the FIC and MSC, respectively. All FIBs contributing to one transmission frame are divided into a number of groups equal to the number of CIFs contributing to the same transmission frame. The information in one group of FIBs refers to one CIF (i.e. for transmission mode I, II, III and IV exist 4, 1, 1 and 2 FIB groups, respectively). The transmission mode independent description of the DAB transmission frame is depicted in Figure 4.25.

One CIF always contains $55\,296$ bits. The smallest addressable unit of the CIF is the Capacity Unit (CU), containing 64 bits. Hence, one CIF contains 864 CUs. Since the MSC

Table 4.12: *Transport characteristics of the DAB transmission frame [38].*

Transmission mode	Frame time, T_f $[ms]$	# FIBs per frame	# CIFs per frame
I	96	12	4
II	24	3	1
III	24	4	1
IV	48	6	2

is divided into sub-channels, each sub-channel occupies an integral number of consecutive CUs. The sub-channels in the multiplex are addressed in the MCI by their associated CU number ($0 \cdots 863$). Each of these sub-channels is individually convolutionally coded with its appropriate error protection.

The data in the MSC is divided into bursts of 24 ms. Each burst is referred to as a logical frame and is associated with a CIF. Because of additional error protection, the convolutionally coded data of all sub-channels within one logical frame are interleaved in time. The time interleaving spreads the convolutionally coded bits of the sub-channels over 16 logical frames (i.e. over 384 ms) to suppress burst errors in the DAB signal.

The DAB standard is based on the OFDM communication principle. Different transmission modes have been defined for different radio channel conditions. The basic OFDM characteristics of the DAB standard are summarized in Table 4.1 for all transmission modes. The occupied bandwidth of the DAB signal is 1.54 MHz. FFT sizes of $2\,048$, 512, 256 or $1\,024$ are applied for the transmission modes, respectively. Only $1\,536$, 384, 192 or 768 carriers have been modulated with data using Differential QPSK (D-QPSK) modulation. D-QPSK differs from QPSK because the information is not modulated in the absolute phase but is modulated in the phase difference between consecutive symbols (i.e. relative phase). Therefore, equalization in the frequency domain, as performed in the HiperLAN/2 receiver, is not required. The differential modulation scheme is more resilient to typical fading scenarios of DAB.

4.4.2 DAB receiver structure

The baseband processing of the DAB radio system is based on the OFDM communication principle. Figure 4.26 depicts the basic DAB receiver structure, derived from the generic OFDM framework. We only consider the receiver functionality in the digital domain of the receiver, starting at the output of the ADC. The sampling rate of the ADC is 2.048 MHz, as is recommended by the DAB standard [38].

The most important building blocks of the DAB receiver are described in order to implement the functionality in embedded processors.

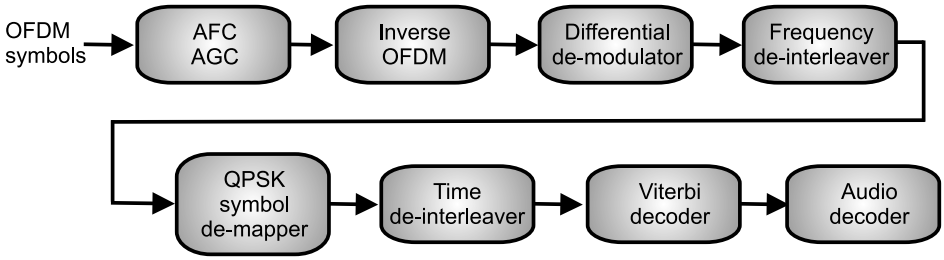

Figure 4.26: *DAB receiver block diagram.*

DAB signal synchronization

The first OFDM symbol of every DAB transmission frame is always the *null* symbol. Hence, during the time period $[0, T_{NULL}]$ the transmitted DAB signal is zero, $\tilde{s}(t) = 0$. This period of silence in the DAB signals is used to synchronize the DAB receiver with the start of the DAB transmission frame. Hence, a drop in the received power points at the start of a new DAB transmission frame.

Once the DAB receiver has been synchronized with the DAB transmission frame, the receiver can optionally be synchronized with the individual OFDM symbols. Therefore, the cyclic extension of the OFDM symbol can be used and finally be removed, resulting in samples with useful data only.

OFDM frequency offset correction

The phase reference symbol is the only OFDM symbol within the DAB transmission signal that can be considered as training sequence. The phase reference symbol provides particularly initialization information for the differential modulation of the next OFDM symbol. In addition, the phase reference symbol is also useful to estimate the frequency offset that is included with the DAB transmission signal. This phase reference symbol consists of reference pilots with good autocorrelation properties. Typical frequency offset estimation methods have been presented in e.g. [12, 74], which utilize the autocorrelation properties of the phase reference symbol.

The frequency offset estimation in OFDM systems is generally performed in a two step approach. Firstly, coarse frequency offset estimation is applied in order to roughly estimate the frequency offset in the received OFDM signal. Secondly, fine frequency offset estimation determines the frequency offset more exactly based on the coarse estimate.

Once the frequency offset is estimated for the received DAB signal, the received signal is compensated for frequency offset by multiplying with the inverse frequency offset in accordance with (4.11). The Automatic Frequency Control (AFC) function in Figure 4.26 implements the frequency offset correction.

Inverse OFDM

Depending on the DAB transmission mode, the useful data part of the OFDM symbol contains $2\,048$, 512, 256 or $1\,024$ samples in the time domain. Generally, the sub-carrier values are determined from the vector of time domain samples by applying the Fast Fourier Transform (FFT). As a consequence, the DAB receiver has to be capable of FFT processing with size $2\,048$, 512, 256 and $1\,024$ in order to be fully compliant with DAB transmission modes I, II, III and IV, respectively. Only, 1536, 384, 192 or 768 sub-carrier values are to be extracted from the output of the FFT operation, because not all sub-carriers are modulated.

Differential QPSK demodulation

Differential modulation is used for the QPSK modulated symbols on each sub-carrier; the sub-carriers of the DAB signal are $\pi/4$ D-QPSK modulated. The advantage of using differential modulation is that information is modulated in the phase difference between consecutive symbols instead of the absolute phase of the current symbol.

In the DAB transmitter, the differential modulation is defined by:

$$\tilde{z}_{l,k} = \tilde{z}_{l-1,k} \cdot \tilde{y}_{l,k}, \tag{4.26}$$

where

- l denotes the OFDM symbol number in the DAB transmission frame;
- k denotes the (modulated) sub-carrier number;
- $\tilde{y}$ is the QPSK modulated symbol;
- $\tilde{z}$ is the differential modulated symbol.

The phase reference symbol is the first OFDM symbol sent in the DAB transmission frame. So, the phase reference symbol is used to initialize the differential (de)modulator.

At the DAB receiver side, the differential modulated signal has to be demodulated to the original QPSK modulated signal. The differential demodulation operation is defined by:

$$\tilde{y}_{l,k} = \tilde{z}^*_{l-1,k} \cdot \tilde{z}_{l,k}, \tag{4.27}$$

where $\tilde{z}^*_{l-1,k}$ denotes the complex conjugate of $\tilde{z}_{l-1,k}$.

From (4.27) can be derived that channel state information is utilized implicitly during the differential demodulation phase. The channel state is concealed in the received baseband signals, $\tilde{z}_{l,k}$. In this manner the reconstructed baseband signal, $\tilde{y}_{l,k}$, which is resolved from the phase differences between two consecutive received symbols, $\tilde{z}_{l,k}$, is hardly influenced by channel distortions. Since the channel state has implicitly been considered, equalization of the sub-carrier values is not required.

Frequency de-interleaving

The QPSK symbols are not one-to-one mapped on the sub-carriers of one OFDM symbol. Frequency interleaving is applied to ensure that adjacent bits are not mapped to adjacent sub-carriers. This principle is used to combat frequency-selective fading effects. The DAB receiver has to apply the reverse operations that are performed by the DAB transmitter during frequency de-interleaving[12].

The QPSK symbols are re-ordered in the DAB transmitter according to the relation:

$$\tilde{y}_{l,k} = \tilde{q}_{l,n}, \tag{4.28}$$

where

- k denotes the sub-carrier index, with $-\frac{K}{2} \leq k < 0$ and $0 < k \leq \frac{K}{2}$;
- n denotes the QPSK symbol index at the output of the QPSK symbol mapper, with $n \in [0, K\rangle$.

The actual re-ordering is given by [38]:

$$k = \Pi(i) - \frac{N}{2}, \tag{4.29}$$

and

$$\Pi(i) = 13\Pi(i-1) + \frac{N}{4} - 1 \pmod{N}, \tag{4.30}$$

with

- N the FFT-size of the DAB transmission mode;
- $\Pi(0) = 0$;
- $i \in \langle 0, N\rangle$.

Since not all N sub-carriers are modulated, only K values out of the set of N values need to be mapped. Therefore, only the values $\frac{N-K}{2} \leq \Pi(i) < \frac{N}{2}$ and $\frac{N}{2} < \Pi(i) \leq \frac{N+K}{2}$ are considered in (4.29). In this way (4.29) produces an ordered set of K values. The first element in the ordered set gives the index of the sub-carrier, k, on which the first QPSK symbol is transmitted, and so on.

[12] The location of the frequency de-interleaver in the DAB receiver can be changed (i.e. the functionality can be moved). Eventually, the frequency and time de-interleaver can be merged. Disadvantage of merging the de-interleaver functionality in the receiver is the fact that de-interleaving rules might become more complex and address less regularity.

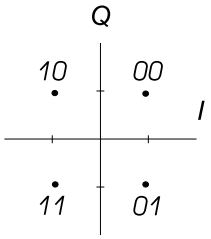

Figure 4.27: *DAB QPSK constellation.*

QPSK symbol de-mapping

The constellation scheme used in DAB is given in Figure 4.27 [38]. For every QPSK symbol two bits from the information stream are transmitted. Unlike the HiperLAN/2 system, these are not two consecutive bits. The bits, $p_{l,n}$, are mapped to the QPSK symbol, $q_{l,n}$, according to the relation:

$$q_{l,n} = \frac{1}{\sqrt{2}} \left[(1 - 2p_{l,n}) + j \, (1 - 2p_{l,n+K}) \right] \tag{4.31}$$

The received QPSK symbols are translated back to the information bit sequence in the QPSK de-mapping function of the receiver. Both *hard-decision* or *soft-decision* can be applied to do the de-mapping.

Time de-interleaving

Time interleaving is used on the DAB signals to combat burst errors. The time interleaving is only applied for all the encoded sub-channels of the MSC. The FIC is not subject to time interleaving.

The interleaving rules are defined according to Table 4.13. The rules show relative simple regular behaviour of the (de-)interleaver functionality. However, the bottleneck with this large interleaver over 16 logical frames is the memory capacity required to perform the interleaving operations of the entire DAB ensemble. The time interleaving rules are explicitly based on the bit index within the MSC bit sequence. The bit index of the MSC bit sequence defines in which logical frame the information is sent; the *bit-reversed* bit index $(\mathrm{mod}\ 16)$ defines the logical frame number to which the information is mapped. Note that the bit position (i.e. the bit index) within the logical frame does not change during time interleaving, only the logical frame in which the information is transmitted changes. Hence, a benefit of this approach is that partial time de-interleaving is possible in the DAB receiver in case not all sub-channels need to be received (i.e. the DAB ensemble is partly of interest by the end user).

In case the entire DAB ensemble needs to be received by the end user, 16 logical frames need to be fully buffered in the receiver. The size of one logical frame (or CIF) is invariably $55\,296$ coded bits. The uniform time interleaving rules, as given in Table 4.13,

cause all information bits to be distributed equally over 16 logical frames. Since the time interleaving is uniformly applied over 16 logical frames, on average $8.5 \times 55\,296$ bits need to be stored in the memory of the DAB de-interleaver. Figure 4.28 shows the progress of the de-interleaving operations for part of the logical frame.

The de-interleaving operations for only 16 bits are shown in Figure 4.28, because the operations are always identically performed on blocks of 16 coded bits. The data in the first row of Figure 4.28 gives the received data, which was interleaved at the transmitter side. The indices of the received interleaved data, $r_{x,y}$, denote the original position of the data in the logical frame at the transmitter before interleaving. The index y denotes the bit position in the logical frame (or CIF) and x depicts the index of the logical frame in which the data was originally transmitted before interleaving. Thus, the received interleaved data stream $\{r_{15,0}; r_{7,1}; r_{11,2}; \ldots\}$ denotes that the first received data sample, $r_{15,0}$, is originally (i.e. before interleaving) mapped to the first data item of logical frame 15, the second received data sample, $r_{7,1}$, maps to the second data item of logical frame 7, the third received data sample, $r_{11,2}$, maps to the third data item of logical frame 11, and so on.

The received data is written in the de-interleaver memory in a column-wise fashion. The dark-grey boxes show the write position of the received interleaved data in the de-interleaver memory. The de-interleaved data is read from the column that is labelled "F" in this example. Since the de-interleaver memory is cyclic, the column "F+1" outputs the next de-interleaved frame. Generally, $8.5 \times$ *coded bits of interest* need to be stored during the de-interleaving process[13].

The interleaver functionality is not considered as part of the baseband processing. The implementation of the de-interleaver in reconfigurable hardware will not be discussed in the remainder of this chapter. Chapter 6 is completely dedicated to channel decoding, which partly includes interleaving issues.

Channel decoding

Forward Error Correction (FEC) coding is used in the DAB system on the entire DAB transmission frame. The channel encoding process is based on punctured convolutional coding. The convolutional mother code, used in DAB, has rate $R = 1/4$ with constraint length $k = 7$. The generator polynomials of the mother code are $(133, 171, 145, 133)$ [38]. Depending on the required protection level of the data, different puncturing schemes are used, which allow Equal Error Protection (EEP) and Unequal Error Protection (UEP). 25 different puncturing schemes are defined in the DAB system. The 25 different puncturing schemes are combined in order to yield error protection profiles with different rates.

The data in the FIC are jointly encoded by the mother code. The encoding rate of the FIC after puncturing is approximately $1/3$. The service components (i.e. the subchannels) in the MSC are all individually encoded and punctured. 5 protection profiles have been defined for UEP mode, which is designed for audio. In the EEP mode, which is

[13] In case soft-decision is applied in the QPSK de-mapper, the total number of stored bits increases linearly with the number of soft-bits per coded bit.

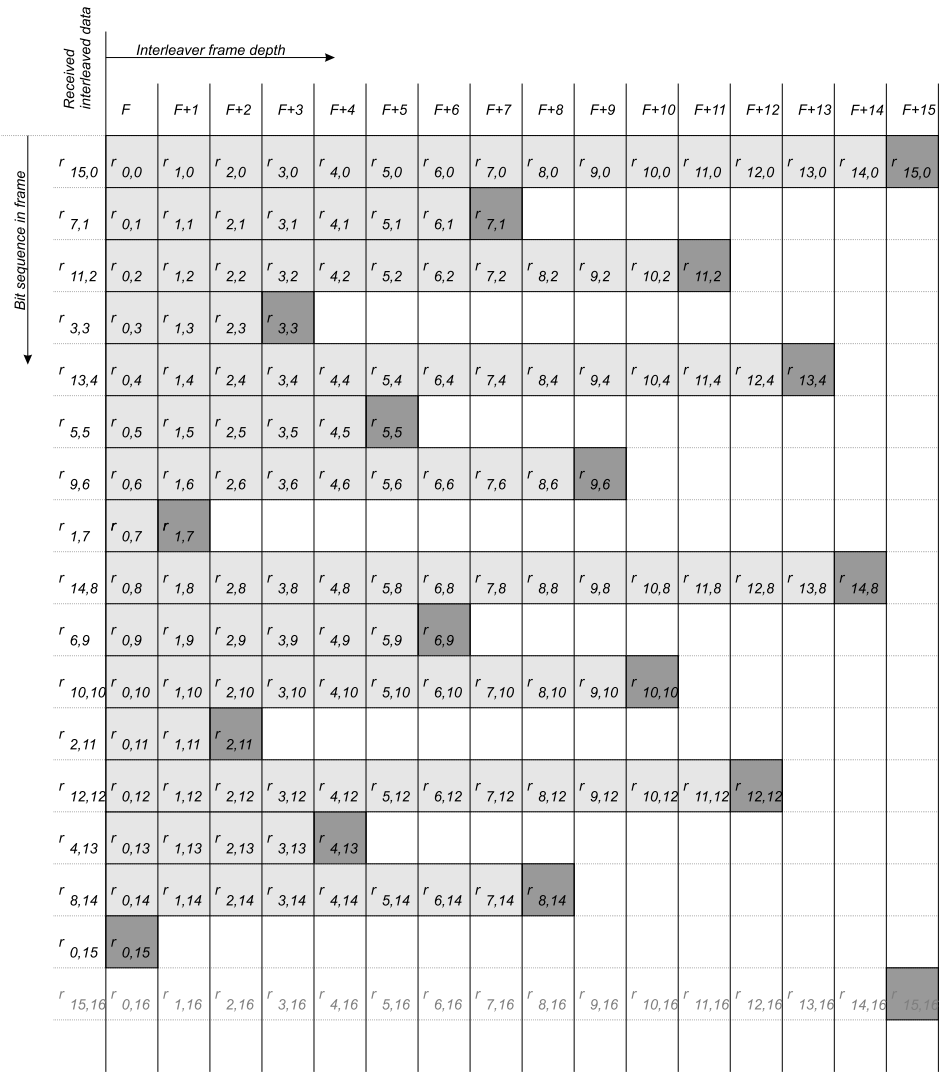

Figure 4.28: *Memory organization of the DAB de-interleaver for part of the logical frame (only 16 bits).*

intended for audio as well for data, 4 protection profiles have been defined. The coding rates of the different protection profiles range from $R = 1/4$ to $R = 4/5$. Information about the utilized error coding profiles for the respective sub-channels is contained within the MCI.

Channel decoding is typically performed by a Viterbi decoder [112]. The existence of a diverse range of puncturing schemes proves to be a challenge for the channel decoding

Table 4.13: *Time interleaving relationship of the DAB system [38, Table 42].*

Bit index of MSC bit sequence	New logical frame number relative to current frame
0	0
1	-8
2	-4
3	-12
4	-2
5	-10
6	-6
7	-14
8	-1
9	-9
10	-5
11	-13
12	-3
13	-11
14	-7
15	-15

complexity. The DAB channel decoder needs to be fairly flexible in order to support 25 puncturing schemes. Furthermore, the DAB channel decoder has to be capable of switching the puncturing scheme during the decoding process of one sub-channel, as the error protection profiles employ a combination of puncturing schemes. Channel decoding techniques and their implementations in reconfigurable hardware are further discussed in Chapter 6.

De-scrambling

Before the data is encoded in the DAB transmitter, the data is scrambled by a modulo-2 addition with a pseudo-random binary sequence. The randomization is done to guarantee that less consecutive *ones* or *zeros* bit sequences exist in the data bit stream. The same scrambler structure is used in the DAB transmitter and receiver to scramble the transmitted data and to de-scramble the received data.

4.4.3 DAB receiver implementation

Parts of the DAB receiver have been implemented on the coarse-grained MONTIUM architecture. The partitioning of the baseband processing functionality is comparable to the HiperLAN/2 receiver in Figure 4.13. Unlike the HiperLAN/2 receiver of Section 4.3, we combined the frequency offset correction and the inverse OFDM processing in one

MONTIUM processing tile. For the integration we took the mapping of the FFT as starting point. Based on the FFT configuration, we augmented an additional configuration for the frequency offset correction functionality.

Inverse OFDM

The inverse OFDM baseband processing is implemented in the MONTIUM by performing a Fast Fourier Transform of size $2\,048$, 512, 256 or $1\,024$. The mapping of the FFT kernel on the MONTIUM architecture is fundamentally identical to the FFT-64 function block of the HiperLAN/2 receiver. In general, N-point FFT kernels, with N a *power of 2* number, can be performed in $\left(\frac{N}{2}+2\right)\log_2\left(N\right)$ clock cycles on the MONTIUM architecture.

Hence, the inverse OFDM baseband processing requires $11\,286$, $2\,322$, $1\,040$ or $5\,140$ clock cycles for DAB transmission mode I, II, III or IV, respectively. The maximum size of the FFT kernel performed in the MONTIUM depends on the size of the local memories in the MONTIUM. The size of the local MONTIUM memories needs to be $\frac{N}{2}$ addresses deep in order to map the FFT-N kernel to the MONTIUM architecture.

Recall from (4.12) that the FFT efficiently implements the DFT:

$$\widetilde{C}\left[x\right] = \sum_{n=0}^{N-1} \tilde{r}\left[n\right] \cdot e^{-j2\pi\frac{xn}{N}} \tag{4.32}$$

The twiddle factors for the FFT operation, $e^{-j2\pi\frac{xn}{N}}$, are stored in the memories of the MONTIUM (refer to page 67). MEM9 contains the real component of the twiddle factors, while MEM10 contains the imaginary part of the twiddle factors (i.e. $\cos\left(2\pi\frac{xn}{N}\right)$ and $-\sin\left(2\pi\frac{xn}{N}\right)$ are stored, respectively). The twiddle factors in the MONTIUM memories are only stored for a half period, $[0,\pi\rangle$, regardless the size of the FFT[14].

OFDM frequency offset correction

The AFC kernel corrects for the frequency offset, f_Δ, of the incoming signal. This frequency offset is delivered to the function as phase shift per sample by the frequency offset estimation function. Unlike frequency offset correction in the HiperLAN/2 receiver, the implemented DAB frequency offset correction function efficiently reuses the configuration information of the FFT kernel. The AFC extension of the MONTIUM FFT implementation uses the twiddle factor information to generate the frequency offset correction coefficients.

Recall from (4.22) that for frequency offset correction, the input data has to be multiplied with:

$$e^{j\varphi_n} = \cos\left(\varphi\right) + j\sin\left(\varphi\right) \tag{4.33}$$

[14] The twiddle factors can be constructed from *cosine* and *sine* tables that are stored for only a quarter period, $[0,\frac{\pi}{2}\rangle$. However, reducing the size of the tables requires more control in generating the twiddle factors, reducing the regularity of operations in the FFT algorithm.

In the MONTIUM-based frequency offset correction implementation 4 Arithmetic Logic Units (ALUs), ALU1 through ALU4, are involved with complex multiplication. Every received sample is multiplied with the complex rotation factor. The complex multiplication instruction of the ALUs used in the FFT algorithm can be reused during frequency offset correction. The complex rotation factor is generated from the 2 memories, MEM9 and MEM10, which are used in Look-up Table (LUT) mode. These 2 memories contain the twiddle factor information used by the FFT algorithm (i.e. the $\cos(\varphi)$ and $-\sin(\varphi)$ tables for $\varphi \in [0, \pi\rangle$, respectively). The basic operation of the complex multiplication is performed according to:

$$\begin{array}{rcl} \tilde{z} & = & \tilde{x} * \tilde{y} \\ & \Updownarrow & \\ \Re(\tilde{z}) & = & \Re(\tilde{x}) * \Re(\tilde{y}) - \Im(\tilde{x}) * \Im(\tilde{y}) \\ \Im(\tilde{z}) & = & \Im(\tilde{x}) * \Re(\tilde{y}) + \Re(\tilde{x}) * \Im(\tilde{y}) \end{array} \tag{4.34}$$

The memory address for the LUTs is determined by ALU5, which calculates the total accumulated phase over the received samples of the OFDM symbol. The phase offset per sample due to frequency offset, which is determined by the frequency offset estimator, is added to the already accumulated phase for every sample. The LUTs cover only a half period, $[0, \pi\rangle$, due to the twiddle factors in the FFT implementation. As a consequence, rules have to be implemented to reconstruct the complete period of the *cosine* and *sine* values.

ALU5 operates in fixed-point mode to generate the addresses for the memories MEM9 and MEM10. The ALU takes the normalized phase shift per sample, $\varphi_n = \varphi/\pi$, as input. In this way one complete period of the phase shift is mapped to the range $[-1, 1\rangle$, which perfectly fits within the MONTIUM fixed-point number representation. The MSBs of the ALU output are used as memory address in the LUTs. The number of MSBs taken into account depends on the size of the local MONTIUM memories. The following rules apply for creating the frequency offset correction coefficient out of the *cosine* and *sine* LUTs:

$$\begin{array}{lll} \text{if} & \varphi \geq 0 & \text{then use MEM9, MEM10} \\ \text{if} & \varphi < 0 & \text{then use -MEM9, -MEM10} \end{array}$$

In Chapter 3 the MONTIUM architecture was introduced. One property of the MONTIUM is that the ALUs can provide status information about calculations inside the ALU. The MONTIUM can be configured to provide particular status information of an operation performed in a particular ALU (e.g. saturation, zero, sign, etc.). This status information is represented as a single status bit (SB). The SB can be used in the MONTIUM sequencer to control the sequencer program (e.g. jump on SB condition).

The mapping of the complex multiplication on the MONTIUM always performs the calculation of the *real* multiplication results, $\Re(\tilde{z})$, on ALU1 and ALU2. The *imaginary* part of the complex multiplications, $\Im(\tilde{z})$, is mapped on ALU3 and ALU4. Consequently,

the instructions of ALU1 and ALU3 only have to be changed based on the SB information. The instructions of ALU2 and ALU4 remain constant for every quadrant of the rotation factor. The main loop of the frequency offset correction for DAB mapped on the MONTIUM is shown in Figure 4.29.

Because the accumulated phase is calculated in ALU5, the SB information of ALU5 is used to determine the quadrant in the complex plane of the rotation factor. So, based on the SB of ALU5, the ALU instructions for ALU1 through ALU4 are selected in order to perform the complex multiplication with the right *cosine* and *sine* values. The MONTIUM sequencer controls the signal processing during frequency offset correction by selecting the appropriate ALU instructions based on the SB. In fact, the basic complex multiplication rules in (4.34) are modified to perform the correction with the right rotation factor:

$$
\begin{aligned}
&\text{for } \varphi \geq 0: \\
&\qquad \Re(\tilde{z}) = \Re(\tilde{x}) * \Re(\tilde{y}) + \Im(\tilde{x}) * \Im(\tilde{y}) \\
&\qquad \Im(\tilde{z}) = \Im(\tilde{x}) * \Re(\tilde{y}) - \Re(\tilde{x}) * \Im(\tilde{y}) \\
&\text{for } \varphi < 0: \\
&\qquad \Re(\tilde{z}) = -\Re(\tilde{x}) * \Re(\tilde{y}) - \Im(\tilde{x}) * \Im(\tilde{y}) \\
&\qquad \Im(\tilde{z}) = -\Im(\tilde{x}) * \Re(\tilde{y}) + \Re(\tilde{x}) * \Im(\tilde{y})
\end{aligned}
$$

The MONTIUM sequencer implements a state machine, which indirectly controls, among other things, the ALU instructions (refer to Chapter 3). In general, mappings of signal processing kernels on the MONTIUM extensively exploit regularity in the algorithm. Those regular structures in the algorithms generally result in loop constructions in the MONTIUM sequencer program.

The frequency offset correction of the DAB signal is always performed on an OFDM symbol basis. This mean that for DAB transmission mode I, II, III or IV always $2\,048$, 512, 256 or $1\,024$ time domain samples are applied for frequency offset correction, respectively. Since the basic operation of this frequency offset correction is the complex multiplication, the algorithm can be easily implemented using loops of sizes $2\,048$, 512, 256 or $1\,024$ in the MONTIUM sequencer program.

Figure 4.30 depicts an example of a control loop that can be implemented on the MONTIUM sequencer. The pseudo code shows how the appropriate ALU instructions are selected based on the SB condition. However, this sequencer program does not change the ALU instructions for every clock cycle. The MONTIUM sequencer program is limited to one sequencer instruction per clock cycle. By introducing jump statements based on the SB condition, this means that extra sequencer instructions are issued in the programmed loop. As a consequence the sequencer loops become one clock cycle longer for every additional jump instruction. Hence, the ALU instruction remains constant for at least two clock cycles in the example sequencer program of Figure 4.30.

The loop counter value in the sequencer program is first initialized by setting `N=510`. After initializing the loop counter value, the next code line of the sequencer program

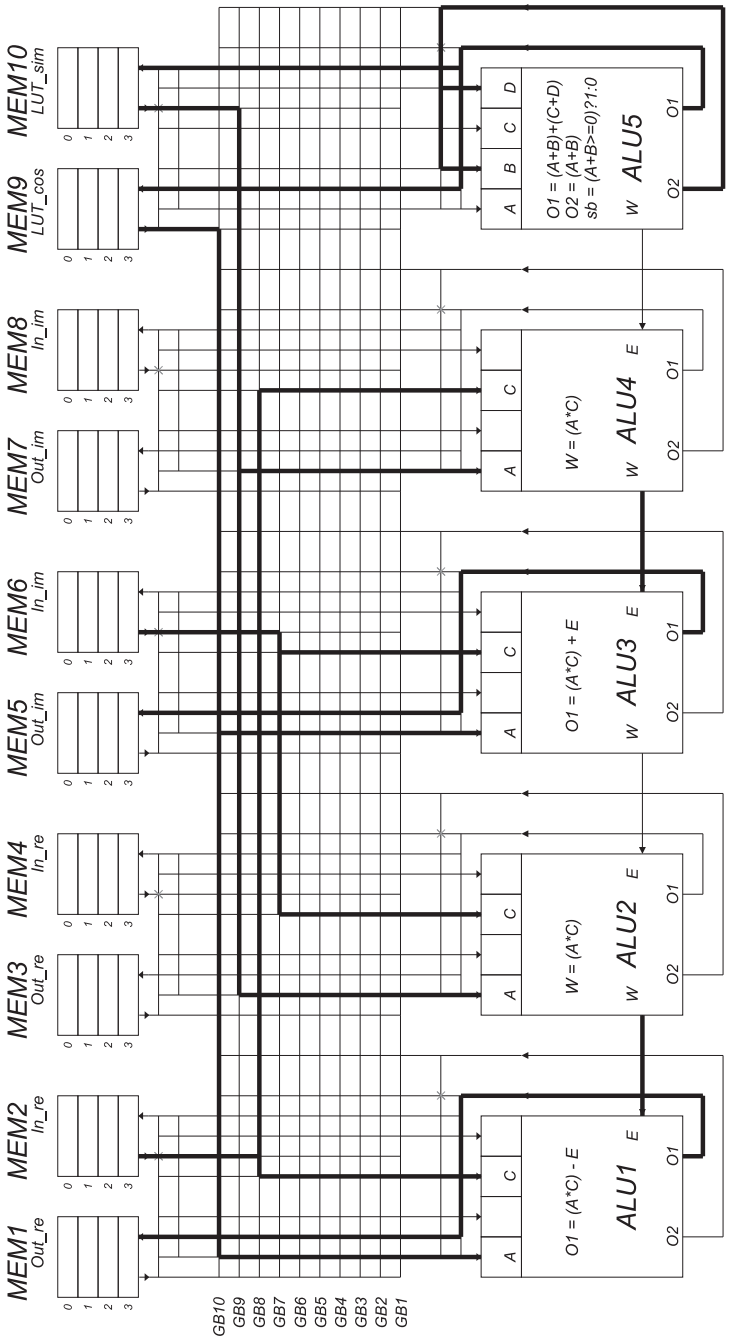

Figure 4.29: *Main loop of DAB frequency offset correction part mapped on the* MONTIUM.

code alias	*ALU instruction*	*Sequencer instruction*
`start`	⋮	⋮
⋮	⋮	`SET (N,510)`
`alu_0`	`ALU_0`	`JNC (SB, loop_1)`
`loop_0`	`ALU_0`	`LOOP (alu_0, N)`
`end_0`	`ALU_0`	`JUMP (next)`
`alu_1`	`ALU_1`	`JCC (SB, loop_0)`
`loop_1`	`ALU_1`	`LOOP (alu_1, N)`
`end_1`	`ALU_1`	`JUMP (next)`
`next`	⋮	⋮
⋮	⋮	⋮
`finish`	⋮	⋮

Figure 4.30: *Pseudo code for control loops in the* MONTIUM *sequencer program based on SB conditions.*

is executed: ALU instruction `ALU_0` is issued and depending on the SB condition, the sequencer program will execute code line `loop_0` or `loop_1`. Code line `loop_0` or `loop_1` issues ALU instruction `ALU_0` or `ALU_1`, respectively. Every time the sequencer `LOOP` instruction is executed, the loop counter value, `N`, is decremented by 1. As long as the loop counter value, `N`, is not equal to zero, the sequencer program jumps back to the start of the programmed loop (i.e. code line `alu_0` or `alu_1`). If the loop counter value, `N`, equals zero, then the next sequencer code line will be executed. So, code line `end_0` or `end_1` is executed when the loop `loop_0` or `loop_1` has finished, respectively. The `end_0` and `end_1` sequencer code lines issue the ALU instructions `ALU_0` and `ALU_1` for a last time, while the sequencer jumps out of the control loop and proceeds the remaining program, starting at code line `next`.

Totally, $(510 + 1) \times 2 + 1$ ALU instructions are issued in the given example of the sequencer program. The loop in the sequencer program is always controlled by the SB condition. The sequencer program is fully stable and will not show any deadlocks, because both conditional loops, `loop_0` and `loop_1`, use the same loop counter.

DAB receiver implementation results

We implemented the frequency offset correction kernel and the inverse OFDM kernel of the DAB receiver on the coarse-grained MONTIUM Tile Processor (TP). The rest of the DAB receiver functionality was modelled in MATLAB. We integrated the frequency offset correction functionality with the FFT kernel. The integrated functionality was im-

plemented for DAB transmission mode IV. Thus, we took the FFT-1024 MONTIUM configuration as a starting point.

Configuration The size of the configuration file for ordinary FFT-1024 functionality is 1432 bytes. With the integration of the frequency offset function we reused as many instructions as possible that were already mapped on the MONTIUM for FFT processing. However, extra instructions needed to be added to the configuration file. Furthermore, the sequencer program of the MONTIUM was modified, as the time domain samples first need to be corrected for frequency offset before they can be used in the FFT algorithm.

The original FFT-1024 sequencer program has been extended with 28 lines of additional sequencer code, resulting in a new sequencer program of 201 lines of code. The extra lines implement the control loop as described in the example pseudo code (Figure 4.30). Furthermore, extra ALU instructions were added, as well as some extra decoder instructions. In total the new configuration file of the integrated frequency offset correction + FFT kernel is only 1676 bytes large[15].

Frequency scaling All DSP operations in the DAB receiver are performed on OFDM symbol basis. Thus, during the OFDM symbol time the integrated frequency offset correction and FFT have to be performed. The MONTIUM mapped FFT algorithm requires 11 286, 2 322, 1 040 or 5 140 clock cycles for DAB transmission mode I, II, III or IV, respectively. For frequency offset correction an additional number of 2 052, 516, 260 or 1 028 clock cycles are required, respectively. So, in total 13 338, 2 838, 1 300 or 6 168 clock cycles have to be processed within the OFDM symbol time of respectively 1 246, 312, 156 or 623 μs. This requires that the MONTIUM runs at a clock frequency of about 10.7, 9.1, 8.3 or 9.9 MHz if the communication overhead is not considered. More DSP operations can be performed on one particular OFDM symbol by the same MONTIUM when the operating clock frequency of the MONTIUM is increased.

However, the limited time slot (e.g. OFDM symbol period) is not the only limiting factor of partitioning the DAB receiver DSP functionality on MONTIUM tiles. The availability of free resources within the reconfigurable MONTIUM architecture influences the partitioning as well. Additional integration of the frequency offset correction, FFT and differential demodulation functionality in the same MONTIUM is not feasible, as there are not sufficient memory resources available in the MONTIUM.

The integration of the FFT and differential demodulation becomes problematic because the symbols (after FFT processing) of the last received OFDM symbol are used to perform the differential demodulation of the current received OFDM symbol. For the differential demodulation function the QAM symbols of the entire OFDM symbol need to be stored. However, during FFT processing all available memory in the MONTIUM is allocated for storing intermediate results and twiddle factors. Hence, storing the old QAM symbols in the same MONTIUM tile during FFT processing is impracticable. By

[15] The incremental configuration file for frequency offset correction was manually integrated with the FFT-1024 configuration and manually mapped on the MONTIUM architecture. Currently, MONTIUM compiler tools are being developed that can generate partial/incremental configurations.

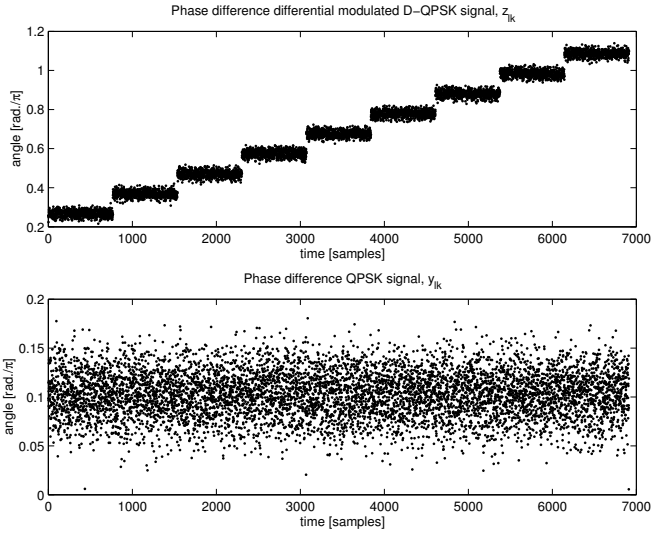

Figure 4.31: *DAB baseband signal received with reference model versus baseband signal received with FFT performed on* MONTIUM *TP.*

doubling the capacity of the local memories and partitioning the memory in (virtual) memory blocks the differential demodulation and FFT processing can, nevertheless, be combined.

The differential demodulation function can be integrated in a second MONTIUM tile, together with the QPSK symbol de-mapping and the frequency de-interleaving functions. These functions perform all of their DSP operations on the same data. We have not mapped this functionality on the MONTIUM.

4.4.4 DAB implementation verification

The implemented DAB receiver is tested by using hardware / software co-simulations as illustrated in Figure 4.17. A reference model of the DAB receiver in MATLAB (using 64-bit floating-point computations) was used to verify the MONTIUM implementation. The reference model is based on a MATLAB model that was developed in the SMART CHIPS FOR SMART SURROUNDINGS (4S) project [6]. Simulations were performed with both the reference model and the MONTIUM implementation of the DAB receiver.

With simulations we verified whether the integrated approach of combining the frequency offset correction and the FFT functions in one MONTIUM tile results in correct functional behaviour. We simulated the reception of an entire DAB frame. For DAB transmission mode IV 76 OFDM symbols are transmitted per frame. The DAB transmission frame contains 6 FIBs and 2 CIFs. The FIBs as well as the CIFs contain random bit sequences for verification during simulation.

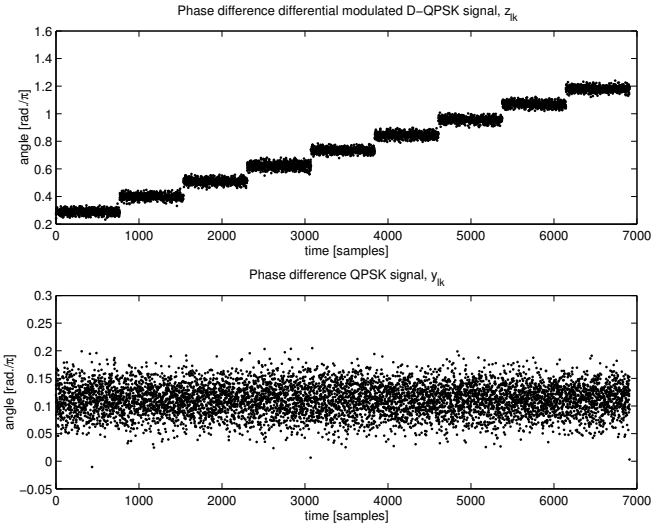

Figure 4.32: *DAB baseband signal received with reference model versus baseband signal received with FFT and frequency offset correction performed on* MONTIUM *TP.*

During experiments we explicitly added frequency offset to the received DAB signal. For completeness the included frequency offset was estimated in software (i.e. MATLAB) by the DAB receiver. This estimation was used by the MONTIUM to correct the received DAB signal for frequency offset. No additional noise was added to the received DAB signal, because we only verified the functional behaviour of the frequency offset correction based on reuse of the FFT twiddle factors.

Three situations of the DAB receiver have been simulated:

1. The entire DAB receiver is simulated by the floating-point reference model;

2. All baseband functions of the DAB receiver are performed by the floating-point reference model, except for the inverse Orthogonal Frequency Division Multiplexing (OFDM) function. The inverse OFDM function is performed by the FFT algorithm that is mapped on the MONTIUM TP;

3. All baseband functions of the DAB receiver are performed by the floating-point reference model, except for the frequency offset correction and the inverse OFDM function. The frequency offset correction and the FFT algorithm are mapped on one MONTIUM TP, in which the twiddle factors of the FFT are reused by the frequency offset correction function.

In order to estimated the influence of the frequency offset correction and the FFT algorithm performed on the MONTIUM TP, the phase information of the received differential modulated signal, z_{lk}, has been compared for the different situations. Furthermore, the

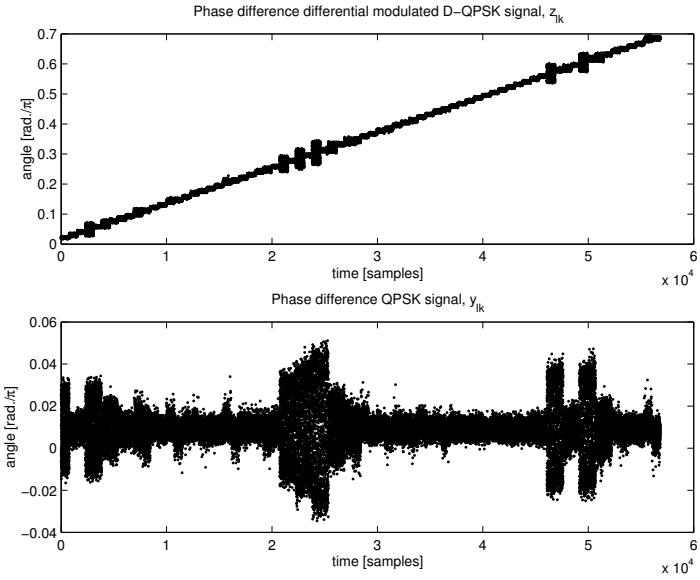

Figure 4.33: *DAB baseband signal received with FFT performed on* MONTIUM *versus baseband signal received with FFT and frequency offset correction performed on* MONTIUM *TP.*

phase information of the baseband signal after the differential demodulation function, which is actually an QPSK signal, y_{lk}, is derived and compared for the different situations. The simulation results are shown in Figure 4.31, 4.32 and 4.33. All three figures show the time on the *x-axis*, indicating the received sample number.

The upper figure in Figure 4.31 shows the relative phase difference between the differential modulated DAB signal, z_{lk}, received with the floating-point reference model and the MONTIUM-based DAB receiver in which only the FFT has been performed on the MONTIUM TP. The lower figure shows the relative phase difference of the QPSK signal, y_{lk}, received with the floating-point reference model and the MONTIUM-based DAB receiver in which only the FFT has been performed on the MONTIUM TP. The reception of 9 OFDM symbols is shown in these figures.

Figure 4.32 compares the floating-point reference model with the integrated approach (i.e. FFT and frequency offset correction are mapped on one MONTIUM TP). The reception of 9 OFDM symbols is shown in this figure.

In Figure 4.33 the situation in which only the FFT is mapped on the MONTIUM TP is compared with the integrated approach in which both the FFT and the frequency offset correction are mapped on the MONTIUM TP. In the integrated approach the twiddle factors are reused by the frequency offset correction function.

The lower figures of Figure 4.31, 4.32 and 4.33 are shown because the actual data is contained in this QPSK baseband signal. This QPSK baseband signal is extracted from the

differential modulated baseband signal according to (4.27). It shows that the integrated frequency offset correction does not negatively influence the data contained in the QPSK signal.

The phase difference between the QPSK baseband signal which is received using the MONTIUM-based DAB receiver in which only the FFT has been performed on the MONTIUM TP and the QPSK baseband signal which is received using the MONTIUM-based DAB receiver in which the FFT processing and frequency offset correction have been integrated in one MONTIUM TP is on average 0.01π. Figure 4.33 depicts this phase difference during reception of an entire DAB transmission frame. The phase difference of 0.01π does not cause demodulation errors, because the information after differential demodulation of the DAB signal is encapsulated in QPSK signals. Hence, the QPSK information is still set in the proper quadrant of the complex plane.

The upper figures of Figure 4.31, 4.32 and 4.33 show that a constant phase offset is introduced for the different OFDM symbols. However, this constant phase offset has no significant influence on the QPSK baseband signal since the data is differentially modulated. The frequency offset correction and FFT processing can be efficiently integrated in the same MONTIUM TP by reusing the FFT twiddle factors, yielding satisfying results.

4.5 Digital Radio Mondiale

Digital Radio Mondiale (DRM), approved as ETSI standard in 2001 [37], has been proposed to replace the analog AM radio services. The DRM system delivers near-FM quality sound over short wave (SW), medium wave (MW) and long wave (LW) radio channels. An advantage of the digital radio system is that it can coexist together with the analog AM radio service in a hybrid radio system context, referred to as In-Band On-Channel (IBOC) system, and DRM uses existing AM broadcast frequency bands. The DRM signal is designed to fit within the existing AM broadcast band plan, based on signals of 9 kHz or 10 kHz bandwidth (Europa or USA, respectively). Furthermore, four robustness modes (mode A, B, C and D) are defined to comply with different SW / MW / LW radio propagation conditions [58].

The DRM system delivers audio and data services to the end-user. In contrast to DAB, MPEG-4 High Efficiency AAC (HE-AAC) has been applied in DRM, which provides good sound quality at a low bit rate. Nowadays, DRM is considered to be an interesting candidate for replacing all analog radio services including FM radio services. Therefore, the DRM standard is extended to use Very High Frequency (VHF) radio channels too. Furthermore, additional channels are defined in the extended DRM standard (DRM+). Those additional channels provide larger data rates.

4.5.1 DRM transport mechanism

The DRM system is capable of transmitting other digital data besides audio. The DRM transport mechanism defines multiple channels, which are multiplexed on one radio channel. The information of these channels is sent within the DRM transmission super frame:

- *Fast Access Channel (FAC)*
  The FAC provides service selection information for fast scanning. The channel contains information about physical layer parameters, to enable the receiver to decode the data in the Service Description Channel (SDC) and Main Service Channel (MSC). The transmitter can choose between different configurations for the SDC and MSC to deal with different conditions of the radio transmission channel and error protection requirements of the data. The FAC provides information on e.g. the channel width, spectrum occupancy and interleaving. The FAC is transmitted in every logical frame;

- *Service Description Channel (SDC)*
  The SDC gives information on how to decode the MSC. The SDC informs the DRM receiver for alternative sources of the same data and provides detailed information about the multiplex configuration of the MSC. The SDC is transmitted once every super frame;

- *Main Service Channel (MSC)*
  The MSC contains the data for all the services contained in the DRM multiplex. The multiplex may contain between one and four services, which can be either audio or data. The audio services are encoded by different MPEG-4 audio decoding schemes (e.g. HE-AAC), depending on the transmission capacity and content. The MSC is transmitted once every super frame.

The data of all three channels are encoded by a multi-level coding (MLC) scheme. The principle of MLC is the joint optimization of coding and modulation to reach the best transmission performance. The MLC scheme ensures that more error prone positions in the constellation of the modulated DRM signal get a higher protection level. The different levels of protection are reached with different punctured convolutional codes.

All MSC QAM symbols are interleaved after channel encoding in the DRM transmitter. The interleaving is applied to the QAM symbols of the MSC after MLC with the possibility to choose low or high interleaving depth (short or long interleaving) according to the predicted propagation conditions. The QAM symbols are interleaved over five DRM transmission frames during long interleaving. Short interleaving only interchanges the sub-carrier allocations within one DRM transmission frame.

Besides the three channels, three different pilot types are transmitted in the DRM signal. Those pilots provide means to derive channel state information in the receiver:

- *Time pilots* are used for synchronization purposes (i.e. determine the start of a DRM transmission frame);
- *Frequency pilots* are used to detect the presence of a DRM signal. They can also be used to estimate the frequency offset;
- *Gain pilots* are mainly used for coherent demodulation. The pilots are scattered throughout the overall time and frequency domain. They are used by the receiver to estimate the channel response.

In DRM four different robustness modes (mode A, B, C and D) are defined that are a trade-off between robustness and data bandwidth. An increase in robustness against transmission errors due to bad channel conditions will be at the cost of the available bandwidth for data. The characteristics of the DRM receiver for the different robustness modes are shown in Table 4.1. Besides the different robustness modes, which are defined to cope with different radio channel propagation conditions, each mode has six different spectrum occupancies[16].

The elementary structure of the DRM transmission signal is organized in transmission frames. Each transmission frame covers 400 ms. Three transmission frames together form a DRM transmission super frame. The basic DRM transmission frame structure is depicted in Figure 4.34, which shows how the data of the three channels is distributed. The basic frame structure is equal for each robustness mode, however, the number of available modulated sub-carriers in the DRM transmission signal changes with the applied robustness mode (Table 4.1). All available information (FAC, SDC, MSC and pilots) is distributed over all sub-carriers of the OFDM symbols within the DRM transmission super frame[17]:

- The *frequency pilots* are always transmitted on three fixed sub-carriers within every OFDM symbol;
- The *time pilots* are always transmitted on particular fixed sub-carriers in the first OFDM symbol of every DRM transmission frame;
- The *gain pilots* are distributed over the entire super frame on different sub-carriers in every OFDM symbol in a pseudo-regular fashion;
- The *SDC* is always transmitted on particular fixed sub-carriers in the first two OFDM symbols of the DRM transmission super frame;

[16] The spectrum occupancies are based on the traditional AM broadcast band plan, to be non-interfering with the present AM radio stations. Besides traditional channel widths (9 kHz or 10 kHz for Europa or USA, respectively), DRM supports double or half frequency bands. The half bandwidth occupancy allows simultaneous use of the analog AM service in the same band (IBOC).

[17] The exact allocation of sub-carriers depends on the applied robustness mode.

- The *FAC* is spread over the entire super frame in a pseudo-regular fashion on different sub-carriers in every OFDM symbol, except the first two. Regardless of the spectrum occupancy, the FAC is always transmitted in the higher frequency part of the DRM signal spectrum;

- The *MSC* is transmitted on the sub-carriers that are not allocated for FAC, SDC or pilots.

The sub-carriers of the three channels (FAC, SDC and MSC) are modulated with different QAM constellations. The FAC is mandatory modulated using a QPSK modulation scheme. The SDC is either QPSK or QAM-16 modulated. For the MSC QAM-16 or QAM-64 modulation schemes are applied. In contrast with the HiperLAN/2 or DAB transport mechanism, where all data is transmitted in a consecutive manner, all different types of data in the DRM system are transmitted in a scattered fashion through the entire signal spectrum. Consequently, consecutive sub-carriers in the OFDM symbol of the DRM signal adopt diverge functions (i.e. FAC, SDC, MSC or pilot symbol) and so the modulation of consecutive sub-carriers alternates continuously.

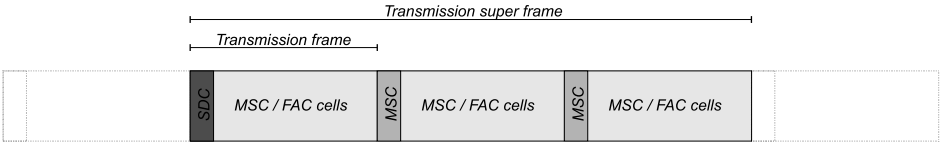

Figure 4.34: *Schematic overview of the basic DRM transmission frame structure and the allocation of the data symbols.*

4.5.2 DRM receiver structure

The baseband processing of the DRM receiver is based on the generic OFDM receiver framework from Figure 4.4. Figure 4.35 shows the basic structure of the DRM receiver. Only the baseband receiver functionality in the digital domain of the receiver, starting at the output of the ADC, is considered. The sampling rate of the ADC is 12 kHz for half and single band spectrum occupancies. For double band spectrum occupancy the sampling rate of the ADC is increased to 24 kHz, as is recommended by the DRM standard [37].

The most important baseband processing building blocks of the DRM receiver are described to investigate the complexity of the functionality to be implemented in embedded processors [85, 113].

DRM signal synchronization

Synchronization of the received DRM signal is supported by the time pilots, which are transmitted in the first OFDM symbol of every DRM transmission frame. Moreover, the SDC information facilitates the synchronization with the start of the DRM super frame.

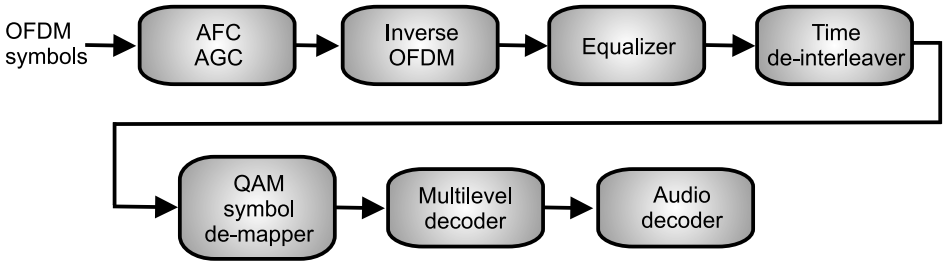

Figure 4.35: *DRM receiver block diagram.*

OFDM frequency offset correction

The inserted frequency pilots in the DRM transmission signal facilitate the estimation of frequency offset in the received DRM signal. Since the frequency pilots are always assigned to fixed sub-carriers in the DRM OFDM symbols, the frequency shift is easily determined with respect to the reference sub-carriers (i.e. the frequency pilots). The frequency offset can alternatively be estimated by using the extended cyclic prefix information of the OFDM symbols.

The received DRM signal is compensated for frequency offset through multiplying the received time domain samples with the inverse frequency offset in accordance with (4.11). The Automatic Frequency Control (AFC) function in Figure 4.35 implements the frequency offset correction.

Inverse OFDM

Depending on the DRM robustness mode and the assigned spectrum occupancy, the useful data part of an OFDM symbol contains 288, 256, 176 or 112 samples in the time domain for half and single spectrum bandwidth. In case of double spectrum bandwidth the number of samples in the useful data part are doubled. Consequently, the DRM receiver has to be capable of performing Fast Fourier Transforms (FFTs) with size 288, 256, 176, 112, 576, 512, 352 and 224.

Special characteristics of the DRM baseband processing are that *non-power of 2* FFTs are applied in 3 of the 4 robustness modes. In order to preserve the orthogonality of the sub-carriers in the DRM transmission signal it is not possible to apply *power of 2* FFTs with zero padding.

OFDM symbol equalization

The received sub-carriers suffer from distortions as a result of frequency-selective fading. Generally, the sub-carriers are compensated for frequency-selective fading effects by multiplying with the inverse channel transfer information. The equalization coefficients for each sub-carrier are determined by the channel estimation function using the included gain pilots in the DRM transmission signal. These gain pilots are spread equally in time

and frequency and are used as reference signals. The amount of gain pilots in the DRM transmission signal, which depends on the robustness mode, determines the accuracy of the channel estimation. Because the characteristics of the radio channel change continuously, the channel estimates need to be updated continuously. The channel profiles as defined for DRM state a maximal Doppler spread of about 8 Hz, thus the coherence time of the DRM channels is at least 125 ms [37, Annex B]. Hence, the channel estimation needs to be applied for every OFDM symbol.

Channel de-multiplexer

The information in the three DRM channels (FAC, SDC and MSC) and the different pilots (gain, time and frequency) are all multiplexed within one DRM transmission signal. The allocation of the different channels and pilots differs every moment in time. So, from OFDM symbol to OFDM symbol the significance of the sub-carriers changes.

The function of the channel de-multiplexer is to separate the multiplexed information at the receiver side. After de-multiplexing the information of all channels and pilots is collected and grouped according to the appropriate function (i.e. FAC, SDC, MSC or pilots).

In comparison with the HiperLAN/2 or DAB receiver, the de-multiplexing of the received DRM signal stream into the desired separated information streams is complex. The complexity is caused by the fact that information of the separate channels and pilots is spread all over the sub-carriers within the entire super frame. Whereas in e.g. DAB and HiperLAN/2 the information, which belongs to one certain channel, is sent in a grouped fashion in multiple consecutive OFDM symbols.

QAM symbol de-interleaving

Interleaving of the QAM symbols is only applied to the symbols containing MSC information [37]. During interleaving in the DRM transmitter, the QAM symbols are shuffled in time with the choice of short or long interleaving. Short interleaving shuffles the MSC QAM symbols within one DRM transmission frame. Hence, the MSC data of one entire frame, called a multiplex frame, need to be buffered in the DRM receiver. For long interleaving, the MSC QAM symbols are interleaved over five DRM transmission frames. Hence, five multiplex frames need to be buffered in the DRM receiver.

The basic interleaver parameters are adapted to the size of a multiplex frame, which corresponds to N_{MUX} QAM symbols. So, one DRM transmission frame contains N_{MUX} QAM symbols with MSC data.

The QAM symbols are interleaved in the DRM transmitter according to the following rules. The same rules are applied in the DRM receiver for de-interleaving [37]:

$$\tilde{z}_{n,i} = \tilde{y}_{n-\Gamma(i),\Pi(i)}, \tag{4.35}$$

where

- $\tilde{z}$ defines the interleaved QAM symbols;
- $\tilde{y}$ defines the QAM symbols before interleaving;
- n denotes the multiplex frame index;
- i denotes the index of the QAM symbol, with $i \in \langle 0, N_{MUX} - 1 \rangle$.

The actual re-ordering is given by [37]:

$$\Gamma(i) = i \pmod{D}, \tag{4.36}$$

and

$$\Pi(i) = \left(5\Pi(i-1) + \left(\frac{s}{4} - 1\right)\right) \pmod{s}, \tag{4.37}$$

with

- $D = 1$ for short interleaving, $D = 5$ for long interleaving;
- $\Pi(0) = 0$;
- $s = 2^{\lceil \log_2(N_{MUX}) \rceil}$;
- If $\Pi(i) \geq N_{MUX}$ then $\Pi(i) = \left(5\Pi(i) + \left(\frac{s}{4} - 1\right)\right) \pmod{s}$.

QAM symbol de-mapping

The sub-carriers of the received DRM signal need to be de-mapped to bit sequences using the appropriate QAM constellations. M-level Quadrature Amplitude Modulation, with $M = 1, 2$ or 3 yielding QPSK, QAM-16 and QAM-64, has been applied in the DRM system. The mapping strategy for each QAM symbol depends on the assigned channel (FAC, SDC, MSC) and the robustness mode. QAM-64 provides high spectral efficiency, but is error prone. The combination of multi-level modulation with error coding (i.e. multi-level coding) ensures that more error prone positions in the constellation of the modulated DRM signal get a higher protection level. For more robustness the DRM standard defines three divergent constellations for QAM-64. For both QPSK and QAM-16 only one constellation is defined in the DRM standard. The DRM receiver has to be able to perform the QAM symbol de-mapping for 5 different constellations in total.

Channel decoding

FEC coding is applied in the DRM system on all information in the DRM transmission super frame by means of MLC. Different MLC protection schemes are applied to protect for bit errors. The individual bits in every level of the QAM symbols are convolutionally encoded and punctured, if necessary. The individual convolutionally encoded bit streams

in every level of the QAM symbols are combined in the DRM transmitter and finally mapped to QAM symbols using the appropriate constellation scheme.

The channel decoder has to be able to reconstruct the individual bit streams of all levels in the QAM symbols (i.e. $1, 2$ or 3 levels for QPSK, QAM-16 or QAM-64, respectively). The reconstruction of the bit streams is supported by information on the convolutional code and the puncturing strategy. The convolutional mother code that is used in DRM is of rate $R = 1/4$ with constraint length $k = 7$ and is equal to the code used in DAB. The generator polynomials of the mother code are $(133, 171, 145, 133)$ [37].

A multi-level decoder, as applied in the DRM receiver, is typically implemented as iterative multistage decoder (iMSD) [14]. The channel decoder is not considered as part of the baseband processing in this chapter. Channel decoding techniques and their implementations in reconfigurable hardware are discussed in Chapter 6.

De-scrambling

The purpose of de-scrambling or *energy dispersal* is to avoid the transmission of regular signal patterns. Before encoding the data of the different channels (FAC, SDC, MSC) the data is scrambled in the DRM transmitter by a modulo-2 addition with a pseudo-random sequence. The same scrambler structure as defined for DAB has been applied in the DRM transmitter and receiver.

4.5.3 DRM receiver implementation

Parts of the baseband processing of the DRM receiver are implemented on the coarse-grained MONTIUM architecture [85, 113]. The partitioning of the baseband functionality is inspired by the HiperLAN/2 receiver in Figure 4.13. The synchronization of the OFDM symbols (i.e. prefix detection and removal) and the frequency offset correction functions are integrated in one MONTIUM Tile Processor (TP) [113]. The *(non-)power of 2* FFTs are mapped on one MONTIUM TP. Furthermore, the possibility of implementing the channel de-multiplexer in reconfigurable hardware is investigated [85].

OFDM symbol synchronization and frequency offset correction

The OFDM symbol synchronization and the frequency offset correction are integrated in the same MONTIUM TP because the DSP operations for both functions are closely related. Especially, the operations for prefix removal, generally referred to as guard time removal, and frequency offset estimation apply common correlation operations [58].

In essence the prefix removal function estimates the start of the useful data part of the OFDM symbol. The start of the useful part is determined by applying a cross-correlation on the cyclic prefix information (i.e. the first N_g samples and the last N_g samples of the OFDM symbol). The location where the correlation value takes the absolute maximum value determines the start of the cyclic prefix and, hence, of the OFDM symbol.

Once the useful data part of the OFDM symbol is determined, the samples are corrected for frequency offset. The cyclic prefix information can be helpful to estimate the

frequency offset of one OFDM symbol. By applying a cross-correlation on the cyclic prefix information, the rotation of the maximum correlation value determines the frequency offset of the entire OFDM symbol.

In [113] the cross-correlation Digital Signal Processing (DSP) kernel is mapped on the MONTIUM TP. The cross-correlation DSP kernel returns as a result the (absolute) value of the correlation as well as the index when the correlation value is at (local) maximum value. Using this information the General Purpose Processor (GPP) determines the average rotation per sample caused by frequency offset. Finally, the samples of the useful data part are corrected for frequency offset. Investigations for determining the frequency offset based on the correlation results are given in [113] as well. The investigation concludes that it is also possible to accurately determine the rotation of the correlation result, and thus the rotation due to frequency offset, in the MONTIUM TP.

Inverse OFDM

Although *power of 2* FFTs are implemented more efficiently than *non-power of 2* variants, *non-power of 2* FFTs have been implemented on the MONTIUM TP. The *non-power of 2* FFTs are required in the DRM system because they generate an orthogonal set of sub-carriers. The *non-power of 2* FFT mapping on the MONTIUM TP is based on the Prime Factor Algorithm (PFA) [46, 47]. The PFA splits the original DFT in sets of smaller DFTs that are relatively prime. The set of smaller DFTs consists of *power of 2* and *non-power of 2* DFTs. The *power of 2* DFTs are mapped on the MONTIUM TP as radix-2 FFTs (e.g. in [94, 119]).

DRM channel de-multiplexer

The de-mapping of the QAM symbols depends on the position in both time and frequency domain. The goal of the de-multiplexing process is to determine the function of each QAM symbol. Implementation of the de-multiplexer is a control intensive task and, therefore, it becomes questionable whether it can be implemented on the MONTIUM TP.

Using a LUT approach, we de-multiplex the received information into several information streams (i.e. FAC, SDC, MSC or pilots). The positions of the sub-carriers containing FAC and time pilots are mostly irregular. Furthermore, three frequency pilots are available in every OFDM symbol. All this position / function information can be stored in a LUT. The positions of the sub-carriers that contain gain pilots are arranged according to a regular pattern and can be mathematically calculated. So, this information is not stored in the LUT containing the sub-carrier position / function information.

The de-multiplexer procedure filters the different sub-carriers gradually according to their function. Firstly, the sub-carrier is checked to be a gain pilot. Secondly, in case the prior check has failed, the de-multiplexer investigates whether the sub-carrier information is available in the position / function LUT. We deal with time or frequency pilots or FAC information if the information about the sub-carriers was contained in the LUT. Finally, in case the prior check has failed, we deal with SDC or MSC information. Whether the sub-carrier contains SDC or MSC information depends on the OFDM symbol

number. The first two OFDM symbols within the DRM transmission super frame contain SDC information. The remaining OFDM symbols in the DRM transmission super frame contain sub-carriers with MSC information. It was concluded that the control intensive de-multiplexing task can be more efficiently implemented on a GPP [85].

4.6 Summary

This chapter covered Orthogonal Frequency Division Multiplexing (OFDM)-based communication systems. Different standards (HiperLAN/2, DAB and DRM) are discussed that are derived from the generic OFDM framework. The investigation showed many differences and similarities between the different OFDM-based communication standards. This investigation showed problems and opportunities for implementing OFDM receivers in reconfigurable embedded systems.

4.6.1 Conclusions

Generally, the baseband processing of OFDM-based communication systems is guided by the principle of translating the information from the time to the frequency domain and vice versa. In principle, the structure of the OFDM receiver is similar for every OFDM standard and dominated by the DFT / FFT kernel. Every OFDM receiver comprises a frequency offset correction unit in order to restore the orthogonality of the received OFDM sub-carriers. This functionality is implemented by regular patterns of Multiply-Accumulate (MAC) operations.

Major differences turn up in the manner of sending information on the different sub-carriers in the OFDM symbols:

- Various modulation techniques are used to send information on the sub-carriers of the OFDM signal: differential modulation or coherent (multi-level) modulation;
- All information, contained in the OFDM signal, is generally sent in container structures that are typically referred to as MAC transmission frames. The MAC transmission frames consist of various control streams and data streams.

The complexity of de-multiplexing the various information streams of the MAC transmission frame in the OFDM receiver strongly depends on the manner of sending information on the sub-carriers of the OFDM symbol. Especially, when the function of the allocated sub-carriers alternates for every sub-carrier in the OFDM symbol, the demodulation process becomes additionally complex. The additional complexity is caused by the fact that the applied QAM constellation alternates for every consecutive OFDM sub-carrier.

The CHAMELEON System-on-Chip (SoC) template of Chapter 3 is used to map the baseband processing DSP functions of the HiperLAN/2 receiver. The baseband processing functions of the HiperLAN/2 receiver are partitioned over multiple MONTIUM TPs. The control functions of the High Performance Radio LAN type 2 (HiperLAN/2) receiver are directed by a GPP that is available in the heterogeneous reconfigurable SoC template.

The functionality with regular patterns of MAC operations is efficiently mapped on the MONTIUM architecture (e.g. the frequency offset correction, FFT and channel equalizer). The frequency offset correction is mapped on the MONTIUM TP using the local memories in LUT mode. The LUTs are initialized with *cosine* and *sine* tables. We also integrated the frequency offset correction with the FFT mapped on the same MONTIUM TP. The twiddle factors, used for FFT, have been reused by the frequency offset correction in the integrated approach.

The capabilities of the MONTIUM with respect to de-multiplexing the various transport streams within one MAC transmission frame, such as used in DRM, are questionable because many control operations are involved.

One of the objectives is to identify opportunities for exploiting adaptivity in multi-standard multi-mode wireless communication receivers. In this chapter the following adaptivity features were identified in the HiperLAN/2 receiver:

- *Standards level*
  Different communication standards can be implemented using the same reconfigurable hardware. Table 4.9 shows the size of the configuration files in order to configure the processing elements of the SoC to HiperLAN/2 receiver. Configuring the MONTIUM processing elements for the baseband processing functions takes less than 10 μs, assuming that the communication bandwidth of the Network-on-Chip (NoC) is sufficient;

- *Algorithm-parameter level*
  In the HiperLAN/2 standard different modulation schemes (BPSK, QPSK, QAM-16 and QAM-64) can be applied. This parameter can be changed by switching the constellation table in the de-mapper. This operation does not require any reconfiguration of the MONTIUM TP, but the contents of the Look-up Table (LUT) has to be adapted according to the required constellation.

  Moreover, many OFDM standards have multiple modes defined in which the number of used sub-carriers is changed. Hence, within one standard the size of the applied FFTs has to be adapted according to the applied propagation mode. The size of the FFT can be adapted by partial reconfiguration when it is implemented on the MONTIUM TP [115].

4.6.2 Recommendations

Investigation of the integration of frequency offset correction and FFT functionality in the same MONTIUM TP showed the feasibility of reusing the LUT information for both functions (i.e. *cosine* and *sine* tables or twiddle factors, respectively). The integrated approach reused the twiddle factors, used for FFT, to construct the coefficients for frequency offset correction.

Basically, the coefficients for frequency offset correction are generated with the support of LUT tables that are initialized with *cosine* and *sine* information. The coefficients are generated from the LUTs by using the phase of the coefficient as memory address. However, the LUTs are usually not initialized with one period, $[0, 2\pi\rangle$, of the *cosine* and *sine*, but with half or quarter period. The coefficient generation, therefore, depends on the quadrant in the complex plane in which the coefficient is situated. The right sign of the *cosine* and *sine* information at the output of the LUTs is determined based on the quadrant information.

We observed that control operations, based on the quadrant information, are involved during generation of the frequency offset correction coefficients, especially when the LUTs are not loaded with the *cosine* and *sine* information for one entire period. We implemented the control mechanism for selecting the right sign of the *cosine* and *sine* information with conditional loops in the sequencer of the MONTIUM. The information about the quadrant of the coefficients needs to be checked continuously. However, implementing the continuous control mechanism in the sequencer of the MONTIUM TP is not feasible, since other sequencer instructions need to be executed as well[18].

We recommend to add extra control logic hardware in the MONTIUM TP (e.g. multiplexers that select their operands based on SBs). With the extra control logic it is possible to bypass the MONTIUM sequencer and implement conditional operations in hardware. The advantage is that sequencer instructions are executed by the sequencer, while conditional operations are applied continuously as well. Furthermore, the control logic enables parallelism with conditional loops, which improves the performance of control intensive applications. In Chapter 6 we observe that the conditional operations (e.g. conditional multiplexer functionality like Add Compare Select (ACS) operations) are heavily applied in channel decoding algorithms.

We mainly focused on the mapping of baseband processing on reconfigurable hardware, but we noticed that the channel decoding functionality contains resource demanding functions as well. Especially the (time) de-interleaving functions in the receiver require large memories to buffer data. Therefore, special memory tiles need to be present in the heterogeneous tiled SoC. The address generators for the memory tiles should be easily programmable with the appropriate interleaving rules. In Chapter 6 the channel decoder functionality of wireless communication receivers will be further discussed.

[18] The MONTIUM sequencer can only execute one sequencer instruction per clock cycle.

Chapter 5

WCDMA Communication Systems

This chapter deals with Wideband CDMA (WCDMA) communication systems. First the general framework of the Code Division Multiple Access (CDMA) communication system is introduced, based on the Universal Mobile Telecommunications System (UMTS) standard. Several implementations of signal processing kernels are discussed. The Rake receiver, an important Digital Signal Processing (DSP) kernel in WCDMA receivers, is described in detail and mapped on the coarse-grained reconfigurable MONTIUM *architecture.*

The WCDMA receiver is used to illustrate the feasibility of designing a multi-standard receiver based on a heterogeneous reconfigurable System-on-Chip (SoC) platform. General purpose, fine-grained reconfigurable and coarse-grained reconfigurable processing elements of the heterogeneous reconfigurable SoC are used for implementing the UMTS receiver functionality. The baseband signal processing is mainly performed on MONTIUM *processing elements. The General Purpose Processor (GPP) contributes in control, whereas Field Programmable Gate Array (FPGA) elements in the SoC are used for scrambling code generation.*

Major parts of this chapter have been published in [P9, P10, P13, P18, P20].

5.1 Introduction

In this chapter we illustrate our approach of implementing adaptive multi-standard wireless communication receivers in heterogeneous reconfigurable System-on-Chip (SoC) architectures. The template of the heterogeneous SoC, as shown in Figure 3.3, is applied to illustrate the mapping of a wireless communication receiver supporting the Universal Mobile Telecommunications System (UMTS) standard.

Section 5.2 discusses the principles of Wideband CDMA (WCDMA) communication systems. The basic characteristics of the UMTS standard are described in Section 5.3. The main Digital Signal Processing (DSP) kernel in the WCDMA receiver, i.e. the Rake receiver, is mapped on the coarse-grained reconfigurable processing elements of the SoC, i.e. the MONTIUM Tile Processor (TP). The mapping of the Rake receiver on the MONTIUM is described in Section 5.4. The mapping of the Rake receiver on the MONTIUM has been verified by means of Bit Error Rate (BER) versus Signal-to-Noise Ratio (SNR) performance simulations in Section 5.5.

The approach of implementing adaptive multi-standard wireless communication receivers capable of handling WCDMA communication systems is summarized at the end of this chapter in Section 5.6.

5.2 Wideband CDMA

The Universal Mobile Telecommunications System (UMTS) standard, defined by ETSI [4], is an example of a Third Generation (3G) mobile communication system. The communication system has an air interface that is based on Direct Sequence CDMA (DS-CDMA) [59]. The bit rate of UMTS at the physical level depends on the modulation type and the spreading factor (SF) which will be explained later.

The idea of Code Division Multiple Access (CDMA) is that every transmitted bit is coded with a spreading code of a higher rate, which means that every transmitted bit is multiplied with a spreading code sequence. The individual samples of the spreading code sequence are called *chips*. In this way the information is transmitted at a higher rate (i.e. the chip rate), so the spectrum is spread. In spread spectrum wireless communication systems, the transmission bandwidth is much higher than the information data rate. The spreading factor (SF) determines the ratio between the chip rate and the information data rate.

In order to distinguish between different users / sources in the communication system, many orthogonal code sequences are used that are unique for every user. The orthogonal codes exhibit low cross-correlation between different codes, while having good autocorrelation properties. Hence, choosing the right spreading codes gives CDMA its multiple access capabilities. Figure 5.1 depicts the basic operation principle of CDMA.

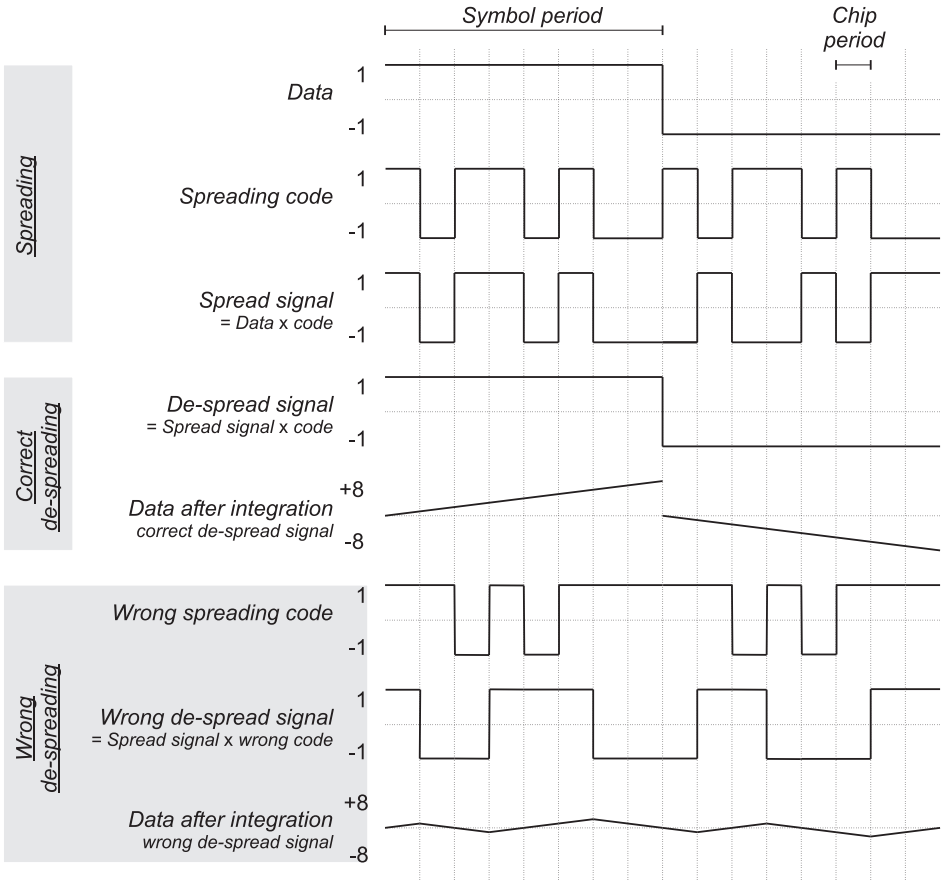

Figure 5.1: *The CDMA operation principle.*

5.3 Universal Mobile Telecommunications System

5.3.1 UMTS transport mechanism

The physical layer of the UMTS communication system offers data transport services to higher layers through the use of transport channels via the Medium Access Control (MAC) sub-layer [3, 27]

The data in the UMTS communication system is transmitted through many different transport channels:

- *Dedicated Channel (DCH)*
  The DCH is a downlink (DL) or uplink (UL) transport channel and conveys data to / from the user;

- *Broadcast Channel (BCH)*
  The BCH is a DL transport channel that is used to broadcast system and cell specific information;

- *Forward Access Channel (FACH)*
  The FACH is a DL transport channel, which contains secondary system and cell specific information;

- *Paging Channel (PCH)*
  The PCH is a DL transport channel that contains secondary information to support efficient sleep-mode procedures;

- *Random Access Channel (RACH)*
  The RACH is an UL transport channel. Users can use this channel to initiate packet transfers on the DCH;

- *Common Packet Channel (CPCH)*
  The CPCH is an UL transport channel and is used to transmit bursty data traffic;

- *Downlink Shared Channel (DSCH)*
  The DSCH is a DL transport channel shared by several users. The DSCH is associated with one or several downlink DCHs;

- *High Speed Downlink Shared Channel (HS-DSCH)*
  The HS-DSCH is a DL transport channel shared by several users. The HS-DSCH is associated with one downlink Dedicated Physical Channel (DPCH).

All transport channels have dedicated functions, but eventually they are all mapped to physical channels. Physical channels are defined by a specific carrier frequency, scrambling code and spreading code, which is also referred to as the channelization code. Physical channels are divided into radio frames with a length of $38\,400$ chips (i.e. $10\ ms$).

Each radio frame contains 15 slots. Every physical channel exhibits about the same structure, except for the allocation of the data bits inside the slots. Table 5.1 depicts the general characteristics of the UMTS communication system.

The major physical channel is the Dedicated Physical Channel (DPCH), which is available in DL and UL direction. Figure 5.2 shows the UMTS frame structure of the DPCH in the DL direction. The DPCH is divided into frames, which are again divided into 15 slots. Each slot contains a Dedicated Physical Control Channel (DPCCH) and Dedicated Physical Data Channel (DPDCH) section. Different transport channels are multiplexed on the DPDCH. The DPCCH section informs the receiver about e.g. parameters of the different transport channels that are multiplexed on the DPDCH using Transmit Power Control (TPC) and Transport Format Combination Indicator (TFCI) bits. Furthermore, pilot signals are sent on the DPCCH section. The DPCCH and DPDCH sections are time multiplexed on the physical channel in the DL direction. Essentially, the frame structure is equal for the UL direction. However, the sections are not time multiplexed, but In-phase/Quadrature (I/Q) multiplexed[1] in the UL direction.

Table 5.1: *Downlink UMTS properties in the FDD mode.*

chip rate	3.84 $Mcps$
chips per frame	38 400
chips per slot	2 560
slots per frame	15
frame period	10 ms
slot period	666.67 μs
scrambling code length	38 400 chips
scrambling code period	10 ms
spreading factor (SF)	4 – 512
symbol rate	7.5 – 960 $ksps$
modulation	QPSK, QAM-16

All data in the physical layer is QPSK modulated, except for the High Speed Physical Downlink Shared Channel (HS-PDSCH) that is used to carry the HS-DSCH transport channel. An HS-PDSCH may use either QPSK or QAM-16 modulation.

[1] In-phase/Quadrature (I/Q) multiplexing means that separate data streams are independently BPSK modulated on the In-phase (I) and Quadrature (Q) component of the complex-number signal, resulting in a QPSK modulated signal. I/Q multiplexing has been applied in the UL to prevent pulsed transmission of the DPCH, which may cause Electromagnetic Compatibility (EMC) problems.

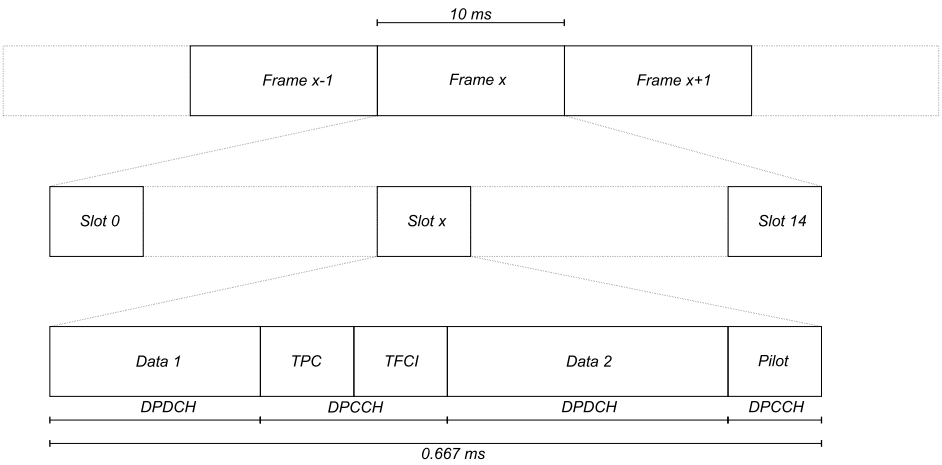

Figure 5.2: *The UMTS frame structure for the downlink Dedicated Physical Channel (DPCH).*

5.3.2 UMTS receiver structure

We investigate the possibilities for implementing the Digital Signal Processing (DSP) functionality of a UMTS receiver on heterogeneous reconfigurable hardware. The heterogeneous System-on-Chip (SoC) template from Figure 3.3 is considered to be the target architecture on which multi-standard adaptive receivers are mapped. We only focus on the downlink of the UMTS receiver at the mobile terminal in the Frequency Division Duplex (FDD) mode. Figure 5.3 shows the DSP functionality of the downlink UMTS receiver.

The CDMA principle requires that multiple multiplications with the spreading code have to be performed in the UMTS receiver in parallel. Figure 5.4 shows the basic idea of the Wideband CDMA (WCDMA) receiver, which is based on a Rake receiver, in more detail. One can distinguish five receive phases in the WCDMA receiver:

- Pulse shaping;
- De-scrambling;
- De-spreading;
- Maximal Ratio Combining;
- De-mapping.

In Figure 5.3 also control-oriented functions are shown. These functions retrieve control information from the received signal and provide this information to the basic WCDMA

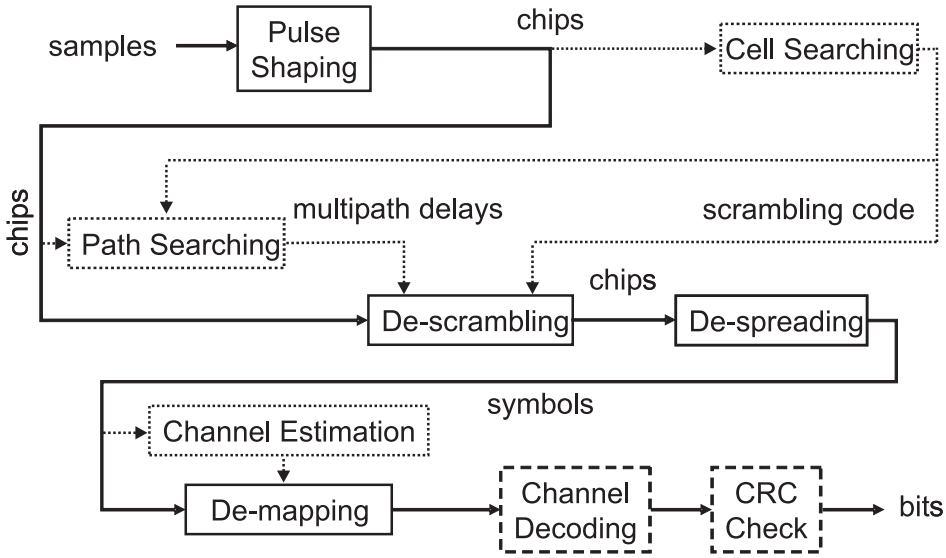

Figure 5.3: *The DSP functionality of the downlink UMTS receiver.*

functions:

- Cell searching;
- Path searching;
- Channel estimation.

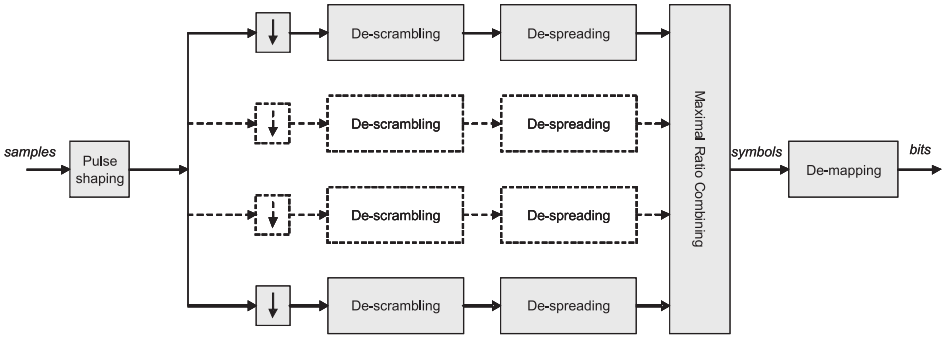

Figure 5.4: *Baseband processing in the Rake-based WCDMA receiver.*

Pulse shaping

Pulse shaping techniques are used in communication systems to simultaneously reduce the intersymbol interference effects and the spectral width of a modulated digital signal. In UMTS a root-raised cosine roll-off filter has been applied [2].

De-scrambling and de-spreading

In UMTS two different spreading mechanisms are applied: de-scrambling and de-spreading [5]. The de-scrambling operation is performed in order to identify the cell, in which the mobile receiver is situated. The UMTS communication system assigns a scrambling code to every cell in the UMTS communication infrastructure. One scrambling code contains $38\,400$ chips. The length of the scrambling code is equal to the frame length in UMTS (i.e. 10 ms).

The de-spreading operation is performed in order to identify the user to which the received information belongs. One or more spreading codes are assigned to every user in the UMTS communication system. The length of the spreading code depends on the SF and is denoted by SF. The SF indicates how many chips are used to transmit one information symbol.

During de-scrambling and de-spreading the received chips are multiplied with the appropriate scrambling code and spreading code, respectively. After multiplication the results are accumulated, which results in one soft-value for the received symbol. This basic operation principle of CDMA has already been shown in Figure 5.1.

Maximal Ratio Combining

Since multipath fading is a common phenomenon in wireless communication systems, the receiver has to combat for the effects of multipath fading. Generally, the signals from the strongest multipaths are received individually in the UMTS communication system. This means that the receiver searches for the strongest received paths and estimates the path-delays. Whenever the delay of an individual path is known, the receiver performs the de-scrambling and de-spreading operations on the delayed signal. The operations of de-scrambling and de-spreading are also referred to as Rake finger. In the Maximal Ratio Combining (MRC) function the received soft-values of the individual Rake fingers are individually weighted and combined to provide optimal Signal-to-Noise Ratio (SNR). The weighting factors of the individual Rake fingers are determined by the channel estimator. The Rake fingers in combination with the MRC are called the Rake receiver[2].

[2] Alternative implementations of the WCDMA receiver are channel equalizers. The equalizer-based WCDMA receivers perform their de-scrambling and de-spreading operations only for one signal path. The de-spread symbols are equalized according to the Channel Impulse Response (CIR), which has been determined by the channel estimation function. Equalizer-based WCDMA receivers yield better performance than Rake receivers under severe Multiple Access Interference (MAI) conditions but are more computationally intensive [60, 107].

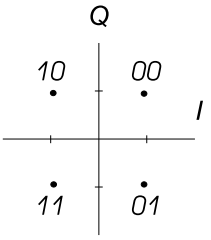

Figure 5.5: *UMTS QPSK constellation.*

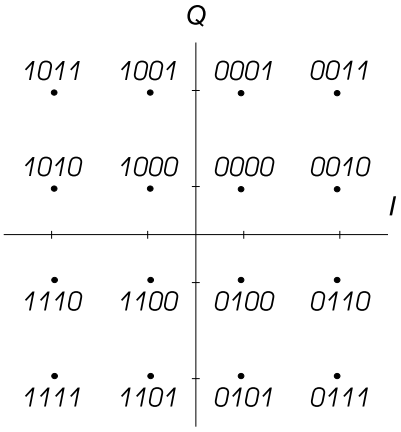

Figure 5.6: *UMTS QAM-16 constellation.*

De-mapping

During de-mapping the received symbols are translated into received bits. Depending on the communication mode, QPSK and QAM-16 modulation are applied in UMTS [4, 5]. Figure 5.5 and 5.6 show the applied constellation schemes for QPSK and QAM-16, respectively.

Cell searching

The cell searcher in the UMTS receiver retrieves the assigned scrambling code from the received signal. A scrambling code is assigned to every individual cell in the UMTS communication infrastructure. The retrieved scrambling code is used during the de-scrambling phase in the WCDMA receiver.

Path searching

Multiple instances of the same signal are received at the UMTS receiver due to multipath effects. Those instances have different delays due to different traveling path lengths of the signal caused by e.g. reflections. The path searcher determines the delays of the most powerful paths. The delays of the paths are used by the Rake fingers to synchronize and correlate with the received signal. Typically, the path-delays are estimated in multiple chip periods.

Channel estimation

Multipath effects do not only lead to the reception of multiple instances of the same signal with different delays, but the instances can all have different phase shifts and amplitudes. In the MRC the received soft-values of the individual Rake fingers are individually weighted and combined to provide optimal SNR. The channel estimator determines the complex-number weighting factors of the individual Rake fingers.

5.4 Rake receiver implementation

Figure 5.4 shows the baseband processing performed in the WCDMA receiver using a Rake receiver. The WCDMA receiver has been implemented in a heterogeneous reconfigurable hardware platform. The heterogeneous SoC template from Figure 3.3 illustrates the target architecture on which multi-standard adaptive receivers are mapped. The majority of the computationally intensive baseband functions has been implemented on coarse-grained reconfigurable hardware, whereas fine-grained reconfigurable hardware and General Purpose Processors (GPPs) are supposed to be used for additional control functionality.

Since most DSP operations in the WCDMA receiver consist of Multiply-Accumulate (MAC) operations on words, the WCDMA Rake receiver has been implemented on coarse-grained reconfigurable hardware, i.e. the MONTIUM Tile Processor (TP). The scrambling code in the receiver can be generated with simple digital logic, consisting of shift-registers and Exclusive OR (XOR) gates [5]. These are typical operations that can be performed in fine-grained reconfigurable hardware, e.g. Field Programmable Gate Array (FPGA). Although we described the necessity of control-oriented functionality in the WCDMA receiver, we will not concentrate on the control functionals, but only on the baseband data processing. We assume that the control-oriented functions provide the right information to the baseband data processing part of the WCDMA receiver.

The receive filter is commonly implemented with a pulse-shape filter. The pulse-shape filter, which can be implemented as a Finite Impulse Response (FIR) filter, can be implemented on one MONTIUM tile. The output streams of the pulse-shape filter are the input for the Rake receiver, which has been implemented on a second MONTIUM tile. Figure 5.7 shows the functional blocks in the WCDMA receiver that are implemented on the tiles of the heterogeneous reconfigurable SoC. The processing elements in Fig-

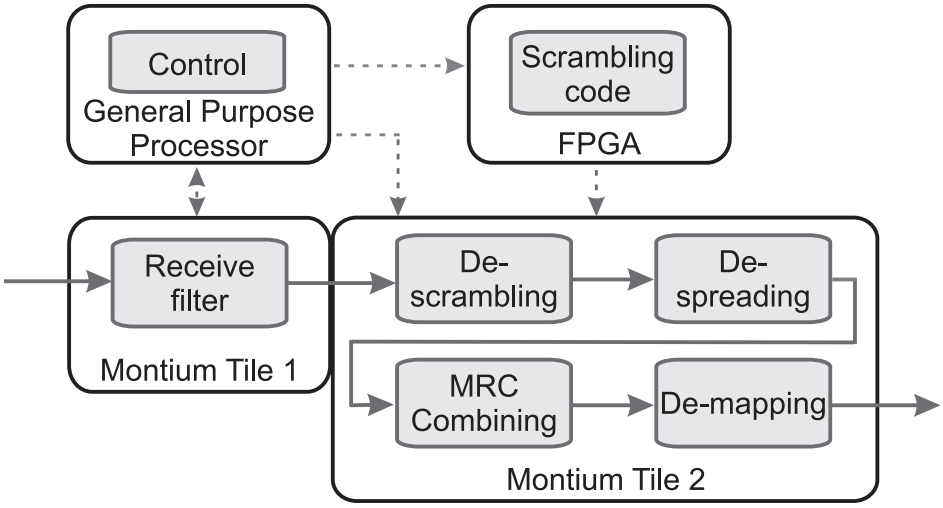

Figure 5.7: *WCDMA receiver mapped on the heterogeneous reconfigurable SoC.*

ure 5.7 (i.e. GPP, FPGA and MONTIUM tiles) can be mapped on the target architecture of Figure 3.3.

The control functionality in the WCDMA receiver is an important part of the WCDMA communication receiver in order to function properly. However, the basic operation principles of the CDMA communication system are implemented by the baseband functionality of the Rake receiver. The control functions, however, also have similar functionality as the baseband functions of the Rake receiver:

- *Cell searching*
  The cell search procedure consists of 3 steps that are all performed on the Synchronisation Channel (SCH). The procedure is well described in the UMTS specifications [3, 5]. The cell search procedure is based on correlation with synchronization codes and the operations are similar to those in the Rake finger;

- *Path searching*
  The path searcher performs its operations at the oversampled chip rate. The autocorrelation of the received oversampled signal is determined. The peaks in this autocorrelation determine the delays of the individual paths;

- *Channel estimation*
  The channel estimator performs its operations at symbol rate. Channel estimation is performed on the de-scrambled, de-spread chips of the individual Rake fingers. Estimates of the channel are determined based on pilot symbols that are transmitted in the Common Pilot Channel (CPICH) or in the DPCCH.

The control functions in the WCDMA receiver are assisted by Rake finger operations. The channel estimation is based on the output of the Rake receiver, which contains the pilot symbols of the CPICH or the DPCCH. Hence, the channel estimation is performed after de-scrambling and de-spreading of the CPICH or the DPCCH. Accordingly, the pilot symbols are used to estimate the channel coefficients for the MRC function. So, the Rake finger operations can be identified as the basic operations in the channel estimator.

Many approaches for channel estimation and path searching have been studied. Discussing and studying the details of these control-oriented functions is beyond the scope of our research. [23, 28, 60, 107] introduce many details about the control functions in the WCDMA communication system. We consider the Rake receiver as a basic part of the WCDMA receiver, since the Rake receiver is also involved in the control-oriented functions of the WCDMA receiver.

5.4.1 Timing properties

The characteristics of the implemented receiver depend on the properties of the UMTS communication system. The most important baseband properties of the downlink UMTS communication system in FDD mode are summarized in Table 5.1. The properties are important for the downlink receiver. During implementation of the receiver one has to consider the sample rate in the different receiver phases:

- The *pulse shaping* filter performs the filtering operations on oversampled chips;
- The *de-scrambling* operations are performed on chips;
- The *de-spreading* operations are performed on chips as well;
- The signals from the different Rake fingers are combined during *Maximal Ratio Combining*. The combining is performed at symbol rate;
- The symbols are de-mapped to bits. The *de-mapping* is performed at the symbol rate as well.

5.4.2 Block versus streaming communication

The timing properties of the UMTS communication system concern both the data processing and the control part of the receiver. However, it seems natural that the data processing is performed according to the *streaming* communication principle, while the control-oriented functions are partly done according to the *block* communication. The implemented WCDMA receiver has been implemented according to the streaming communication principle because for the block communication too many resources, i.e. memory for storing the scrambling code, are required.

According to Table 5.1 the scrambling code sequence consists of $38\,400$ samples. Hence, if the WCDMA receiver operates in block mode then $38\,400$ samples have to be stored in the local memory of the MONTIUM. Even when data would be processed on a

slot-basis instead of a frame-basis, then still $2\,560$ data samples would have been stored in local memory. Therefore, we may conclude that the block communication principle in the WCDMA receiver is not efficient, since blocks are too large.

5.4.3 Communication requirements

The implemented WCDMA receiver operates according to the streaming communication principle. The receiver can process four individual paths of the received signal. Consequently, the receiver requires four complex-number data streams for the four implemented fingers. All implemented fingers require the same scrambling code. The scrambling code can be generated using e.g. an FPGA tile. The implemented receiver takes the complex-number scrambling code stream as an input. Before de-spreading starts, the appropriate spreading code is stored in the local memory of the MONTIUM TP. The spreading code is stored in local memory because the code has a maximum length of 512 samples. Thus, a relatively small amount of data has to be stored in contrast to the scrambling code. Furthermore, the spreading code is assigned to a particular user in the UMTS communication system and, therefore, the spreading code will not change frequently. After de-spreading and before de-mapping, the received symbols of the individual signal paths are combined. During this combining phase, each symbol is scaled with a coefficient. These coefficients are provided by the channel estimator. So, the implemented receiver takes this stream of coefficients as an input. The implemented receiver needs many input streams in order to function properly. The receiver outputs a stream of symbols which contain two bits of received data. The characteristics of all the input and output streams are given in Table 5.2 for a receiver with four fingers.

Table 5.2: *Signal stream characteristics of the implemented WCDMA receiver.*

Signal stream	Direction	Data rate $[MSPS]$
data finger 1	complex input	3.84
data finger 2	complex input	3.84
data finger 3	complex input	3.84
data finger 4	complex input	3.84
scrambling code	complex input	3.84
MRC coefficient finger 1	complex input	$3.84/SF$
MRC coefficient finger 2	complex input	$3.84/SF$
MRC coefficient finger 3	complex input	$3.84/SF$
MRC coefficient finger 4	complex input	$3.84/SF$
de-mapped bits	real output	$3.84/SF$

The MONTIUM targets the 16-bit DSP algorithm domain, and so requires 16-bit data words. Using the MONTIUM as the target architecture, we can translate the characteris-

tics of Table 5.2 into requirements for the Network-on-Chip (NoC). Notice that the output of the receiver is a stream of symbols, which correspond to a pair of two bits for QPSK modulation. The characteristics in Table 5.2 assume that the receiver outputs symbols of 16-bits wide, which eventually contain a pair of two bits[3]. All inputs of the implemented receiver are complex, which means that for each input stream two 16-bit words are required. The maximum bandwidth requirements for the NoC are summarized in Table 5.3, under the assumption of minimum spreading (i.e. $SF = 4$).

Table 5.3: *Maximum bandwidth requirements for the NoC with* $SF = 4$.

Signal stream	Bandwidth $[MBps]$
data finger 1	15.36
data finger 2	15.36
data finger 3	15.36
data finger 4	15.36
scrambling code	15.36
MRC coefficient finger 1	3.84
MRC coefficient finger 2	3.84
MRC coefficient finger 3	3.84
MRC coefficient finger 4	3.84
de-mapped bits	1.92

Figure 3.4 shows that the Communication and Configuration Unit (CCU) is directly connected to the global buses inside the MONTIUM TP. The CCU implements the interface for off-tile communication and so it guarantees that the correct signal streams are available for the MONTIUM TP. Figure 5.8 depicts typical signal activity on the global buses inside the MONTIUM TP during Rake processing. The different signal streams, which are streamed from outside the MONTIUM TP by the CCU[4], are indicated with characters ('A' $\cdots$ 'J') in Figure 5.8. The figure shows that for each finger two signal streams are needed, because the *real* and *imaginary* part of the signal are different streams. The MONTIUM is able to process two Rake fingers in parallel. The data samples of two Rake fingers can be both de-scrambled and de-spread in two clock cycles. The typical signal activity reveals the regular organization of the implemented Rake receiver, which is shown in Figure 5.9 as well. First one chip of finger 1 and one of finger 2 are de-scrambled and de-spread, in the next 2 clock cycles one chip of finger 3 and one of finger 4 are de-scrambled and de-spread. Hence, time multiplexing of the de-scrambling and

[3] An optimization of the QPSK de-mapper results in packing together 16 de-mapped bits of 8 QPSK symbols.

[4] The Rake receiver has been implemented on the MONTIUM TP without having a detailed implementation of the CCU. Therefore, we implemented the Rake receiver using a hypothetical CCU. The CCU [20], as given in Chapter 3, has been designed in a later stage by using the results of the MONTIUM-based Rake implementation as design guideline.

de-spreading operations has been applied for more than two Rake fingers. This typical sequence of signal processing repeats till a complete symbol (consisting of SF samples) is de-scrambled and de-spread. The next five clock cycles are used for combining the results of the four fingers and de-mapping the symbols to a bit stream. The different phases of WCDMA baseband processing operations that are indicated in Figure 5.9 are also depicted in Figure 5.8.

At most six signal streams need to be present simultaneously on the global buses in the combining phase. However, during the de-scrambling and de-spreading phase four signal streams are used for signal processing. Typically, the average communication bandwidth for processing 4 Rake fingers with $SF = 4$ is $94.08\ MBps$.

The input signals of three out of four Rake fingers need to be buffered because they have a different delay. The local memories inside the MONTIUM are used for buffering. Six memories are used for buffering, since the signal streams for the fingers are complex. The spreading code is stored in an additional memory. The same real-valued spreading code has been applied for the I and Q components of the DPCH [5]. So, in total seven memories in the MONTIUM are used for Rake processing.

5.4.4 Dynamic reconfiguration

The complete Rake receiver is implemented on the MONTIUM TP. This receiver includes the de-scrambling and de-spreading of four individual fingers, combining the results of the four fingers, and de-mapping (as shown in Figure 5.4). The configuration size of the complete Rake receiver in the MONTIUM TP is only 858 bytes. Since two bytes per clock cycle can be stored in the configuration memory of the MONTIUM TP, one tile can be configured for Rake receiving in 429 clock cycles. With a configuration clock frequency of 100 MHz this means that a Rake receiver with four fingers can be configured in 4.29 μs[5].

In case the spreading code changes, and so the SF, the new spreading code only has to be loaded in the local memory of the MONTIUM. Storing a particular spreading code into the local memory takes SF clock cycles. Furthermore, one constant in the sequencer program (i.e. the loop counter) has to be changed.

The input streams of the different fingers are buffered in local memories inside the MONTIUM TP. When the delay of one of the channel paths changes, the buffering strategy of the local memories has to be changed (i.e. the AGU instructions are reconfigured). The buffering strategy of the memories is configured with 24 bytes. These 24 bytes can be reconfigured in 12 clock cycles. Consequently, the Rake receiver can update its complete path-delay profile in 120 ns[5].

As can be seen from the signal activity in Figure 5.8, the signal processing of four Rake fingers is very regular. The idea behind the modular, regular structure of the Rake-4 receiver is that it can be easily adapted to another configuration with for instance less fingers. The modular organization of the basic operations in the MONTIUM TP is also shown in Figure 5.9. Suppose we want to change the receiver to a Rake-2 configuration,

[5] In the rest of this chapter we assume that the clock frequency of (re)configuration is 100 MHz.

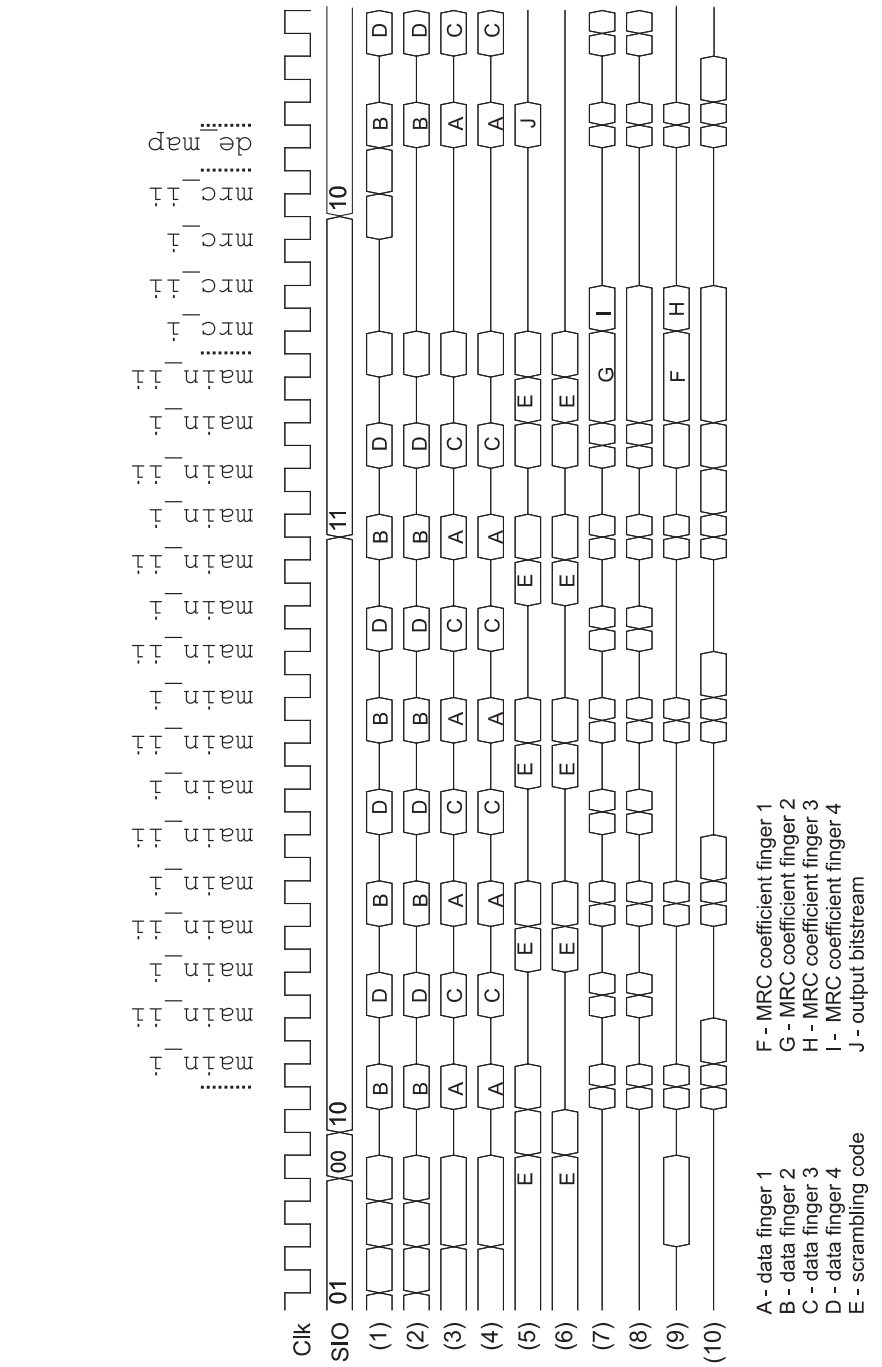

Figure 5.8: *Signal activity inside the* MONTIUM *on the global buses (1) · · · (10) during 4-finger Rake processing.*

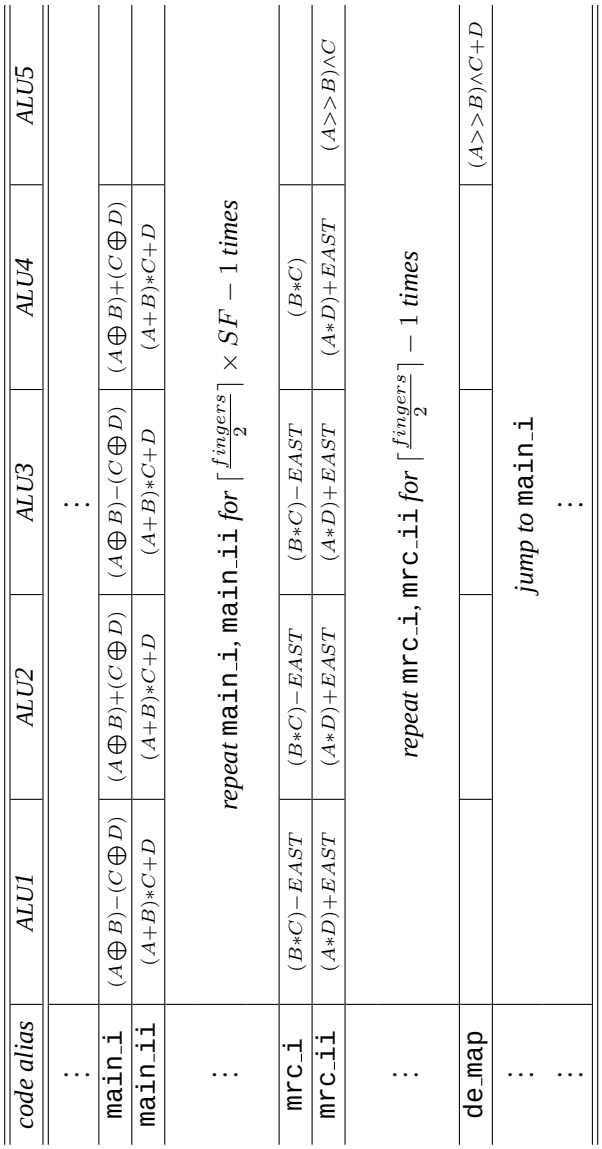

Figure 5.9: *WCDMA baseband processing operations in the* MONTIUM.

this means that finger 3 and finger 4 are no longer needed. The CCU will therefore stall the streaming of stream 'C' and 'D' onto global buses (1) $\cdots$ (4) (refer to Figure 5.8). So, the de-scrambling and de-spreading phase of finger 3 and finger 4 (i.e. data streams 'C' and 'D') can be bypassed. This can be done by changing / adding jump conditions in the sequencer program. Since only two fingers are processed instead of four, the combining phase can also be reduced.

In order to reconfigure the sequencer program, only four jump conditions have to be changed, which corresponds to 8 bytes of configuration data. Furthermore, 4 register instructions and 4 ALU instructions need to be changed, which results in 16 extra bytes that need to be reconfigured. In total, for reconfiguring the number of fingers from four to two, only 24 bytes have to be reconfigured in the configuration memory of the MONTIUM TP. The Rake receiver can be reconfigured in 120 ns, which corresponds to 12 clock cycles[5].

5.4.5 Dynamic power consumption

Voltage and frequency scaling are important measures to control the dynamic power consumption of embedded systems, as given in Chapter 3. Because of the modular, regular structure of the Rake receiver, frequency scaling can be easily applied. From Figure 5.8 can be seen that the clock frequency of the MONTIUM during Rake processing of 4 fingers is about 4 times the chip rate. However, when the Rake receiver is reconfigured to 2 finger processing, the clock frequency of the MONTIUM can be reduced to about 2 times the chip rate.

Using power estimation tooling, the dynamic power consumption of a typical MAC operation in the MONTIUM TP was estimated in [54] to be about 0.5 mW/MHz, realized in 0.13 μm CMOS technology. In Appendix A the average power consumption for Rake processing has been estimated. The dynamic power consumption during 4-finger Rake processing has been estimated to be 0.47 mW/MHz. Detailed information on the power estimation experiments of the MONTIUM-based Rake receiver is given in Appendix A. Table 5.4 summarizes the average dynamic power consumption for the implemented Rake receiver, when two or four fingers are processed.

Table 5.4: *Average dynamic power consumption of the* MONTIUM *Rake receiver.*

	Clock frequency MONTIUM [MHz]	Dynamic power consumption MONTIUM [mW]
2 fingers	10	4.7
4 fingers	20	9.4

An efficient Application Specific Integrated Circuit (ASIC) implementation of a WCDMA Rake receiver was described in [82]. The Post Buffering Rake receiver was

implemented in 0.13 μm CMOS technology. The receiver was partly based on the flexible Rake receiver, *FlexRake*, that has been presented in [50]. According to [82], the total power dissipation of the ASIC implementation is about 1.5 mW with 4 active fingers and about 1.2 mW with 2 active fingers[6]. The total area of the implemented Post Buffering Rake Receiver is determined to be about 0.2 mm^2. When we compare the power consumption of the ASIC implementation with the MONTIUM implementation, we can conclude that the power consumption of the MONTIUM is about 3 to 7 times larger. As expected, the ASIC implementation is more energy efficient than an implementation in reconfigurable hardware, however, the ASIC implementation is fixed and the functionality of the ASIC cannot be changed.

Based on the ideas from [50, 82], another flexible Rake receiver implementation has been presented in [89]. The architecture of that flexible Rake receiver is a specific instantiation from the general AVISPA architecture template [67]. The AVISPA architecture is configurable at design time and programmable after fabrication. According to [89], the area for one reconfigurable Rake receiver is 0.2 mm^2 in 0.12 μm CMOS technology. The power consumption of the generated Rake receiver is estimated to be 6.64 mW. However, these area and power figures are inconsistent with numbers given in [67, 88], which present instances of the same architecture template.

In [66] the Rake finger has been implemented on a TIGERSHARC TS101S DSP [9, 10]. Although it is impossible to make a direct comparison between the implemented Rake finger on the MONTIUM and on the TIGERSHARC, we compare the basic characteristics of both implementations that process DPDCH data. The TIGERSHARC implementation synchronizes the fingers in the MRC unit, while the MONTIUM implementation synchronizes the fingers prior to de-scrambling / de-spreading. The Rake functions of both implementations that can be fairly compared are: *de-scrambling*, *despreading* and *channel compensation*[7]. Table 5.5 summarizes the number of clock cycles that are consumed while receiving one UMTS slot in the DL direction for the FDD mode. The figures in Table 5.5 are given for the situation $SF = 256$.

Table 5.5: MONTIUM *versus* TIGERSHARC *Rake function implementation.*

Function	MONTIUM # clock cycles	TIGERSHARC # clock cycles
De-scrambling & de-spreading	2560	1210
Channel compensation	10	634

[6] The total power dissipation is given for the Post Buffering Rake receiver with $SF = 4$ and with clock gating enabled.

[7] The channel estimation functions have not been considered in the implementation comparison, but only the correction of the samples with the channel coefficients is considered.

The average internal power consumption of the TIGERSHARC TS101S [9] is approximately 5 mW/MHz [8]. Hence, the energy consumption of the TIGERSHARC TS101S is on average 5 nJ per clock cycle. The number of clock cycles required for baseband processing in Table 5.5 shows about equal order of magnitude for both processor architectures (i.e. MONTIUM and TIGERSHARC). Given the figures from Table 5.5, the MONTIUM Rake receiver is at least 10 times more energy efficient than the TIGERSHARC counterpart.

5.5 Rake receiver verification

The functional behaviour of the implemented Rake receiver on the MONTIUM TP has been verified by means of hardware / software co-simulations. Figure 4.17 shows the co-simulation environment with MATLAB [71] and MODELSIM [73], which has been used for functional simulations. Using this co-simulation environment, we can simulate the baseband functionality of the UMTS communication system in software (i.e. using MATLAB and SASUMTSSIM [87]) and partly in hardware (i.e. using MODELSIM). SASUMTSSIM has been developed to evaluate the performance of the de-spreading and demodulation functions of the UMTS downlink [87].

5.5.1 UMTS performance simulations

The receiver that is implemented in the SASUMTSSIM simulator can de-scramble, despread and demodulate multiple downlink physical channels. The physical channels are processed using a Rake receiver for each channel. The number of fingers of the Rake receiver can be specified in the simulator. Cell search, path search and channel estimation functions are implemented in the SASUMTSSIM simulator in order to realistically model a UMTS receiver. In the co-simulation framework of Figure 4.17 we used the baseband functions and algorithms provided by SASUMTSSIM, in order to perform UMTS performance simulations with the MONTIUM-based Rake receiver implementation.

The SASUMTSSIM simulator generates all UMTS signals and stimuli to evaluate the functionality of the MONTIUM-based Rake receiver. The SASUMTSSIM simulator does not only perform baseband functions in the UMTS receiver, but it also incorporates the baseband transmitter functionality of a UMTS communication system. Furthermore, channel models are implemented in the SASUMTSSIM simulator. The Rake receiver, implemented on the MONTIUM TP according to Figure 5.7, needs as inputs:

- *Channel estimates* for all individual Rake fingers. The channel estimates are provided by the channel estimation algorithms in SASUMTSSIM;
- *Scrambling code*. The scrambling code is provided by SASUMTSSIM. The cell search functionality of SASUMTSSIM provides the proper scrambling code;
- *Delay information* for all individual Rake fingers. The path-delay profile information is provided by the path search functionality of SASUMTSSIM;

- *Data streams* for all individual Rake fingers. The data streams are derived from the received UMTS baseband signal. The appropriate chips for the individual Rake finger are selected using the delay information, which is provided by the path search functionality of SASUMTSSIM. The UMTS baseband signal has been generated by the transmitter functionality of SASUMTSSIM.

The performance and functional behaviour of the MONTIUM-based Rake receiver have been evaluated under different propagation conditions. The propagation conditions for performance measurements in a multipath fading environment are defined in [2, Annex B] for the UMTS communication system. The UMTS scenarios from [2, Annex B] that are relevant for MONTIUM-based Rake receiver performance simulations are summarized in Table 5.6. These multipath fading propagation conditions are used to evaluate the MONTIUM-based Rake receiver with 2 and 4 fingers.

Table 5.6: *UMTS propagation conditions for multipath fading environments.*

Case 1 (3 km/h)		*Case 3* (120 km/h)		*Case 4* (250 km/h)	
Relative Delay [ns]	Average Power [dB]	Relative Delay [ns]	Average Power [dB]	Relative Delay [ns]	Average Power [dB]
0	0	0	0	0	0
976	-10	260	-3	260	-3
		521	-6	521	-6
		781	-9	781	-9

For every propagation condition case simulations were performed with both the reference model (i.e. SASUMTSSIM) and the MONTIUM implementation of the Rake receiver. In each simulation one UMTS frame was received. The individual paths in the multipath channel were modelled with Rayleigh fading. In each case the DPCH in the DL direction has been received with a SF of 4. Hence, totally $38\,400$ chips are received in each simulation. This yields $9\,600$ de-spread symbols with a SF of 4. Since the DPCH has been QPSK modulated, a total number of $19\,200$ bits have been received during simulation. The DPDCH part of the DPCH is the only part that contains useful data bits, therefore the performance of the Rake receiver (i.e. BER versus SNR) has been determined with only $18\,720$ data bits. The remaining bits in the DPCH contain information concerning the DPCCH. Since the BER has been determined based on only $18\,720$ bits, BER values smaller than $5 \cdot 10^{-3}$ are hardly significant (assuming that at least 100 errors have been simulated). The SNR is expressed in E_c/N_0, i.e. the ratio of the chip energy (E_c) and noise spectral density in the baseband spectrum (N_0). The SNR in the experiments has been determined by the SASUMTSSIM simulator.

The simulations are all performed under the assumption that the channel estimation, cell search and path search functions are ideal, i.e. they provide the right coefficients without any uncertainty. In this way, the performance of the typical baseband functions in the UMTS Rake receiver is not disturbed by incorrect behaviour of any control func-

tionality in the UMTS receiver. Hence, the comparison of the baseband functions of the MONTIUM-based Rake receiver and the SASUMTSSIM Rake receiver are fair.

UMTS Case 4

Figure 5.10 shows the BER versus SNR performance for the Rake-4 receiver under *Case 4* multipath fading propagation conditions. The results of multiple simulation runs are given in Figure 5.10 for both the SASUMTSSIM and MONTIUM-based Rake receiver[8]. The BER curves in Figure 5.10 show the averaged result after 5 times simulating the reception of one UMTS frame. The curves for both the SASUMTSSIM and MONTIUM-based Rake receiver are given (labelled "Reference" and "Montium", respectively). The relative delay and average power of the multipaths, as defined in Table 5.6, are used as control input for the Rake receiver which implies ideal channel estimation and ideal path searching. The speed of the receiver is assumed to be 250 km/h.

The simulation results show that the performance of the MONTIUM-based Rake receiver and the SASUMTSSIM Rake receiver are hardly different, only for bad channel conditions (i.e. low SNR) a performance gap arises. This gap is caused by the fact of saturation in the ALUs. Properly scaling the input data of the MONTIUM, reduces the effect of saturation. The effect of proper input scaling is depicted in Figure 5.11 by the additional simulation results (labelled "Additional input scaling, Montium"). The additional simulation results are obtained by more rigorously scaling the input data of the MONTIUM in bad channel conditions.

UMTS Case 3

In UMTS *Case 3*, the same multipath channel conditions with respect to power and delay profile are applied as in UMTS *Case 4*. The only difference with *Case 4* is that the velocity of the mobile receiver has been set to 120 km/h for *Case 3*. Again, both the SASUMTSSIM and MONTIUM-based Rake-4 receiver have been simulated. The results of 5 simulation runs and the resulting averaged BER have been depicted in Figure 5.12 (labelled "Reference" and "Montium", for the SASUMTSSIM and MONTIUM-based Rake-4 receiver respectively).

The performance of the Rake-4 receiver under *Case 3* channel conditions shows hardly any difference with *Case 4* channel conditions. Furthermore, the performance of the SASUMTSSIM and MONTIUM-based Rake receiver show similar results[9]. The SASUMTSSIM receiver performs slightly better in bad channel (i.e. low SNR) conditions. Again, the performance gap in bad channel conditions is due to insufficient input scaling. Additional simulations show that properly scaling the input data of the MONTIUM improves the BER performance of the MONTIUM-based Rake receiver (labelled "Additional input scaling, Montium" in Figure 5.13).

[8] $*$ in the BER curves indicates that simulation results for the MONTIUM and SASUMTSSIM are identical.
[9] $*$ in the BER curves indicates that simulation results for the MONTIUM and SASUMTSSIM are identical.

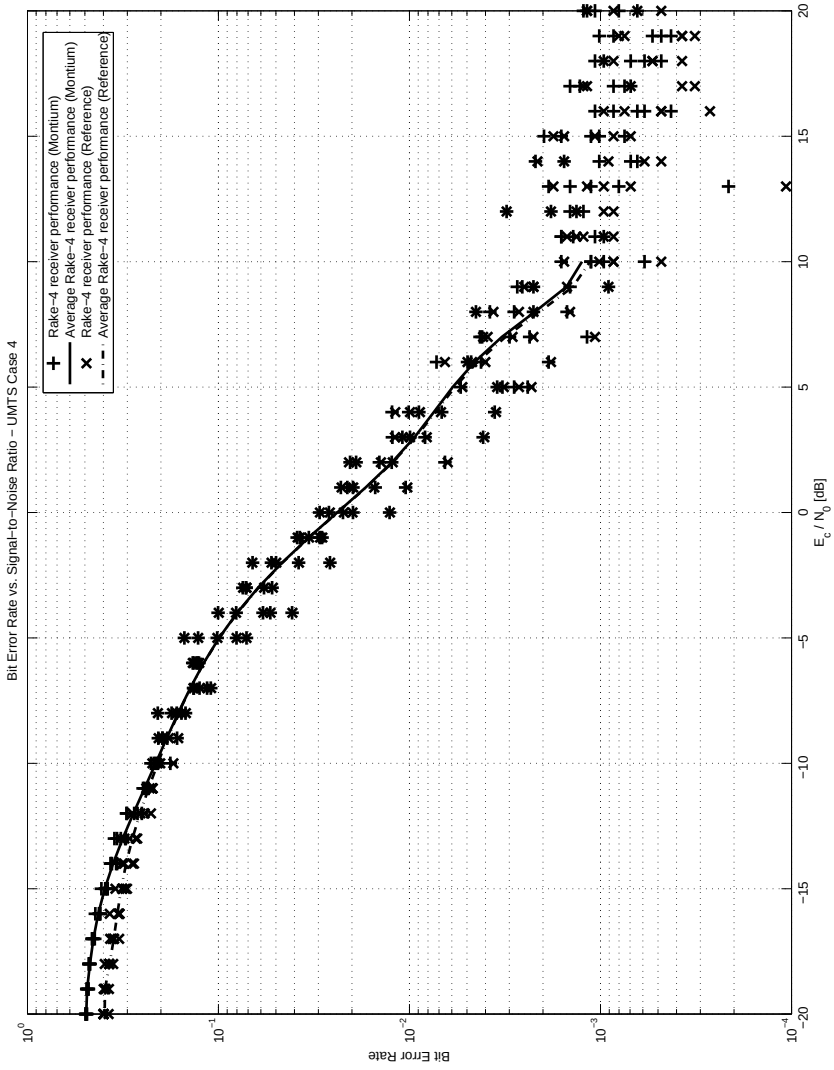

Figure 5.10: *The BER before error correction of the Rake-4 receiver under Case 4 propagation conditions with ideal channel estimation.*

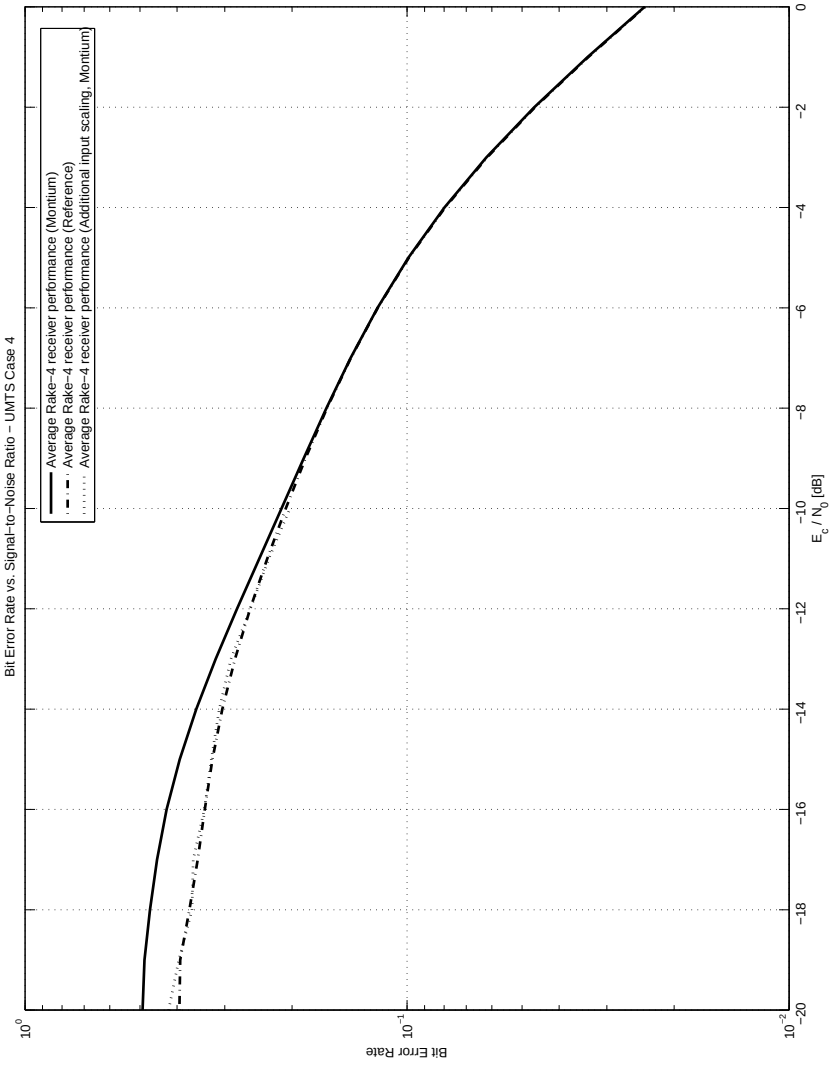

Figure 5.11: *The influence of additional input scaling on the average BER before error correction under Case 4 propagation conditions (average BER obtained from Figure 5.10).*

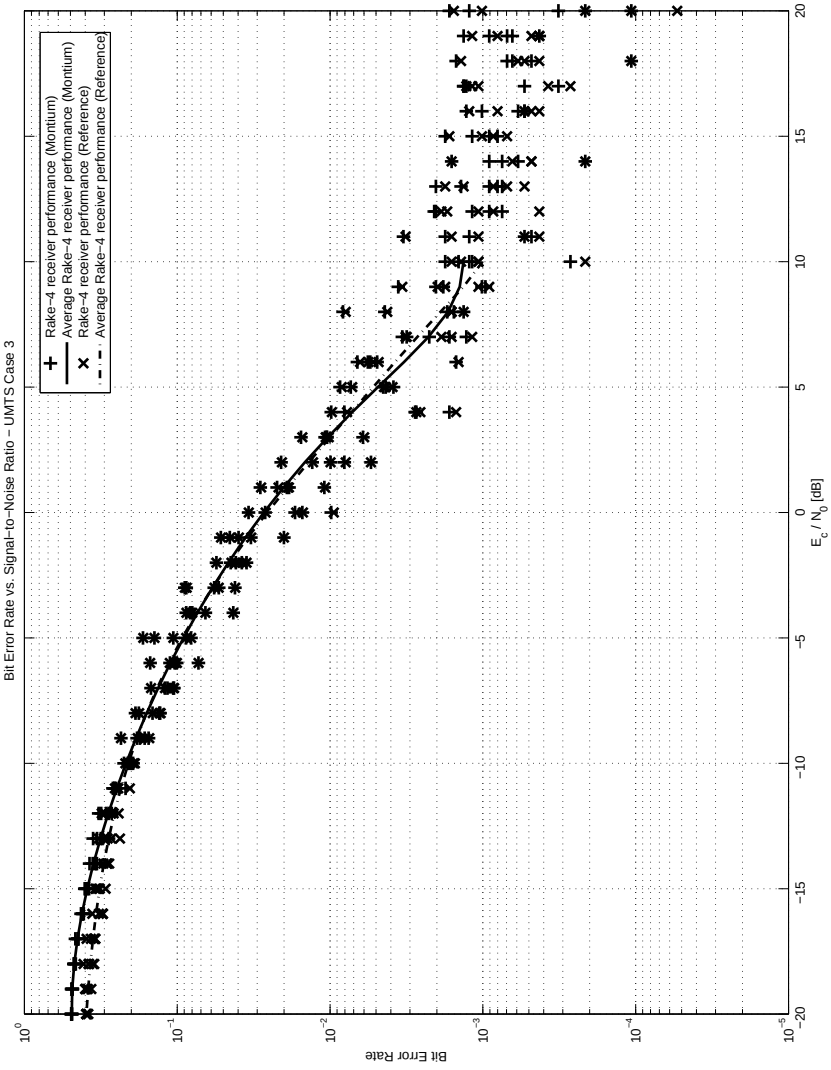

Figure 5.12: *The BER before error correction of the Rake-4 receiver under Case 3 propagation conditions with ideal channel estimation.*

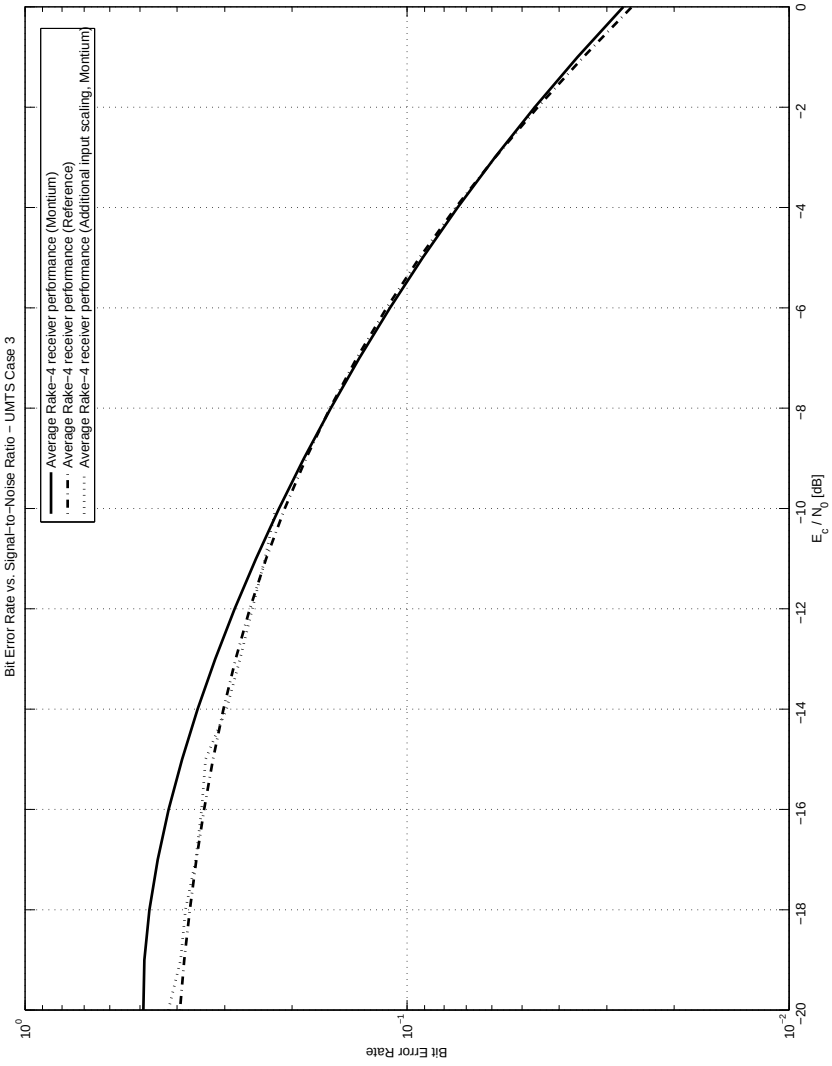

Figure 5.13: *The influence of additional input scaling on the average BER before error correction under Case 3 propagation conditions (average BER obtained from Figure 5.12).*

UMTS Case 1

Under UMTS *Case 1* multipath fading conditions 2 signal paths have been defined according to Table 5.6. Those 2 paths are received by the two fingers of a Rake-2 receiver. In UMTS *Case 1* the velocity of the mobile receiver is set to 3 km/h.

The performance of the MONTIUM-based Rake receiver with 2 fingers has been evaluated and compared with the SASUMTSSIM receiver. Figure 5.14 shows the results of 5 simulation runs as well as the average obtained BER for both receivers[10]. For low SNR channel conditions the performance of the SASUMTSSIM Rake receiver with 2 fingers is slightly better than the MONTIUM counterpart. Both receivers show similar performance figures under increasing SNR conditions.

The importance of properly scaling the fixed-point input data in the MONTIUM-based Rake receiver can also be derived from the BER results in Figure 5.14. In bad channel conditions, the SASUMTSSIM receiver performs slightly better than the MONTIUM-based Rake receiver. This behaviour is due to saturation effects in the MONTIUM ALUs. Saturation of the data in the ALUs can be combated by more rigorously scaling the input data of the MONTIUM. We performed some simulations with more rigorous scaling of the input data in bad channel conditions. The results of these additional tests are included in Figure 5.15 (labelled "Additional input scaling, Montium").

[10] $*$ in the BER curves indicates that simulation results for the MONTIUM and SASUMTSSIM are identical.

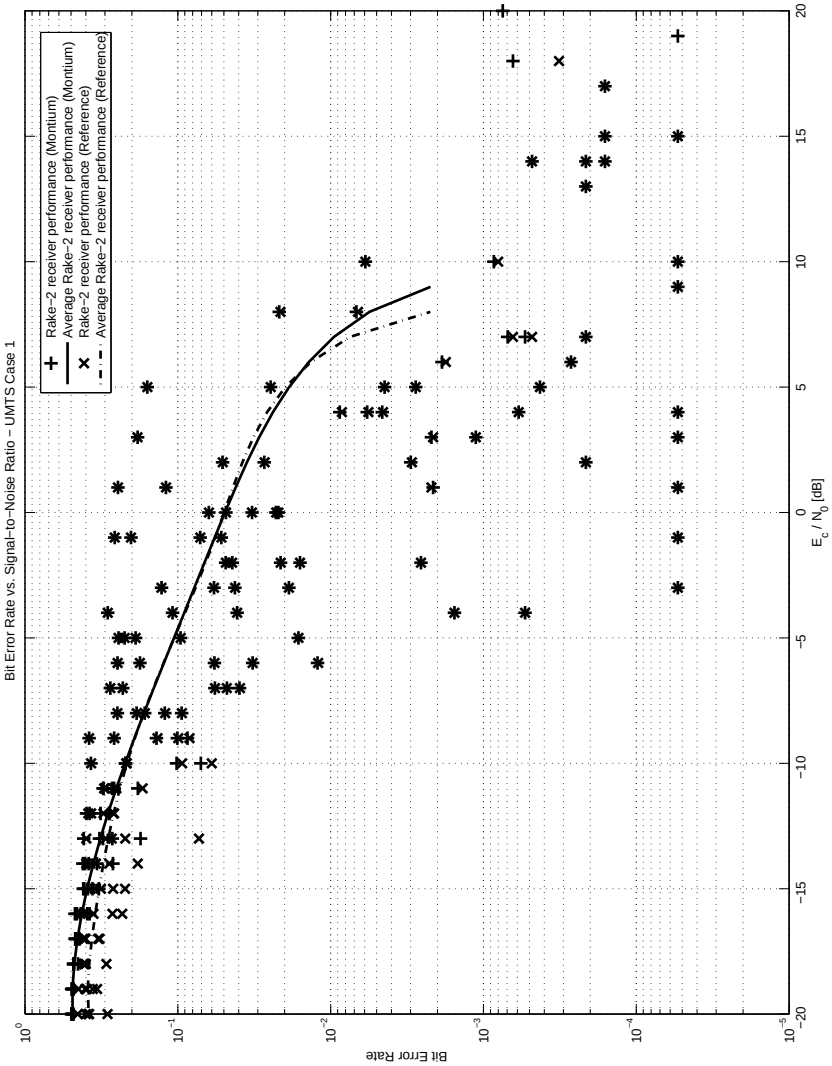

Figure 5.14: *The BER before error correction of the Rake-2 receiver under Case 1 propagation conditions with ideal channel estimation.*

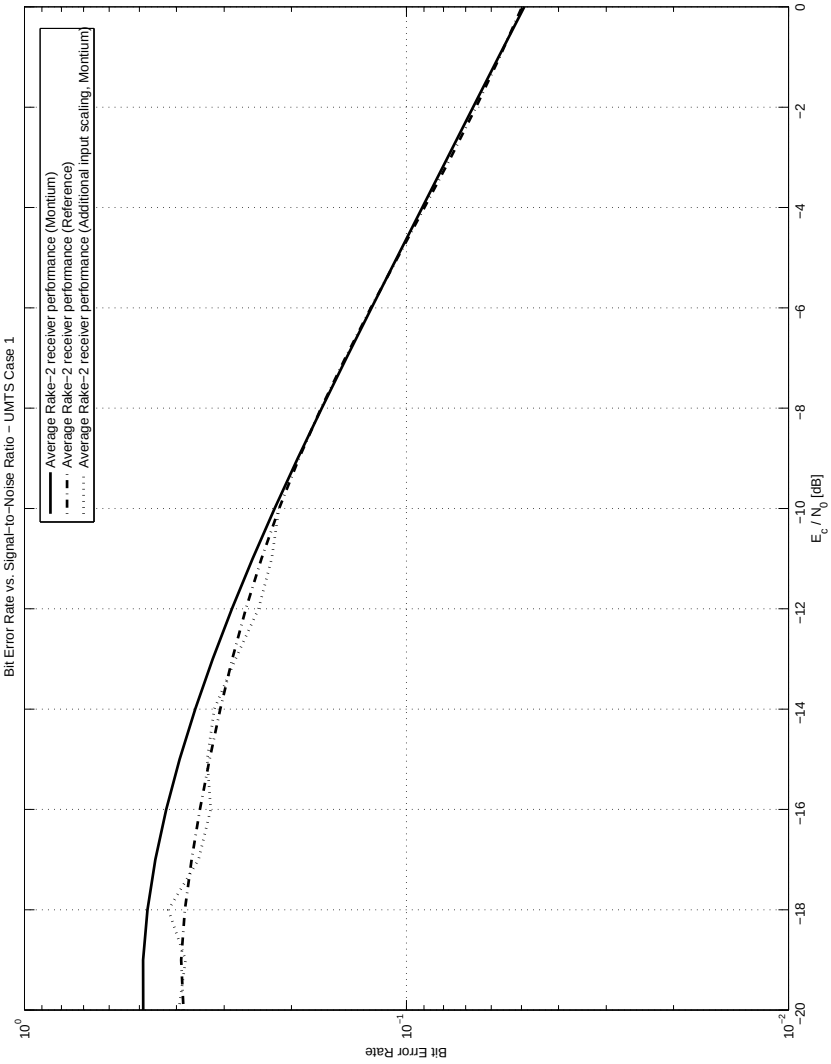

Figure 5.15: *The influence of additional input scaling on the average BER before error correction under Case 1 propagation conditions (average BER obtained from Figure 5.14).*

5.6 Summary

This chapter covers Wideband CDMA (WCDMA)-based communication systems. The Universal Mobile Telecommunications System (UMTS) standard is used as an example of WCDMA communication systems. Requirements for implementing the Digital Signal Processing (DSP) functionality in hardware have been investigated. The investigation showed problems and opportunities for implementing WCDMA receivers in reconfigurable hardware, like the MONTIUM.

The template of the heterogeneous SoC, as shown in Figure 3.3, is applied to illustrate the mapping of a wireless communication receiver supporting the UMTS standard. General purpose, fine-grained reconfigurable and coarse-grained reconfigurable processing elements of the heterogeneous reconfigurable SoC are used for implementing the UMTS receiver functionality.

5.6.1 Conclusions

The Rake functionality is efficiently mapped on the MONTIUM. The Rake functionality comprises Multiply-Accumulate (MAC) operations for de-spreading. Furthermore, multiplications are performed in the Maximal Ratio Combining (MRC) phase of the Rake receiver. The MRC phase combines the de-spread signals from different reflected signal paths. Moreover, the de-spread signals are compensated for channel effects in the MRC phase, like the equalizer operations in the Orthogonal Frequency Division Multiplexing (OFDM) receiver (discussed in Chapter 4).

The algorithms performed in the baseband processing part of the WCDMA receiver comprise relatively simple operations. However, the implementation complexity of the algorithms is caused by the high data rates and high chip rates of the WCDMA signals. Furthermore, the channel estimation algorithms are complicated, but these are not analyzed in this chapter. The channel estimation algorithms, however, are partly dependent on the same baseband algorithms that were presented in this chapter. The control functions in the WCDMA receiver, e.g. cell searching, path searching and channel estimation, interpret the data that is processed by the common baseband algorithms. The interpretations of the control functions result in estimates of e.g. the wireless channel. Hence, the common baseband algorithms provide a WCDMA receiver with important functionality, as these are used in several ways.

The performance of the Rake receiver is verified against a floating-point reference implementation of a Rake receiver. The performance of the Rake receiver is expressed in Bit Error Rate (BER) versus Signal-to-Noise Ratio (SNR) graphs. Simulations for typical UMTS scenarios show hardly any difference in performance between the reference and MONTIUM-based Rake receiver implementation. Experiments show that proper input scaling of data influences the performance of the MONTIUM-based Rake receiver considerably.

The WCDMA receiver implementation on the MONTIUM shows a typical example of *streaming* Digital Signal Processing (DSP). The WCDMA receiver was implemented according to the streaming communication principle because for block communication

too many resources, i.e. memory for storing the scrambling code, are required during the baseband processing.

The implemented Rake receiver, which comprises the main baseband processing of a WCDMA receiver, is designed according to a scalable, flexible scheme. The design is flexible and scalable in the sense that the number of Rake fingers and the path-delay profile can be configured dynamically. Consequently, the characteristics of the WCDMA receiver (i.e. the Rake receiver) can be adapted to its operating conditions at run-time.

Mapping the Rake receiver on the MONTIUM gives the flexibility to adapt the receiver quickly to typical operating conditions. The configuration size of the Rake receiver with four fingers, mapped on the MONTIUM, is only 858 bytes. The MONTIUM is configured in 4.29 μs when a configuration clock of 100 MHz is applied, as every clock cycle 16-bits are written in the configuration memory. Adaptation of the Rake receiver to typical environmental conditions, e.g. changing path-delay profile, is done by partial reconfiguration. The characteristics of the Rake receiver are semi-instantly changed through partial reconfiguration of the MONTIUM. Partial reconfiguration occurs in a few clock cycles, i.e. in the order of nanoseconds.

The streaming Rake implementation on the MONTIUM TP requires a flexible communication interface which connects the MONTIUM TP with e.g. a Network-on-Chip (NoC). Because of scalability, the communication interface of the MONTIUM, referred to as Communication and Configuration Unit (CCU) (depicted in Chapter 3), needs to handle a variable number of data streams. The throughput of these data streams needs to be guaranteed. Furthermore, the CCU needs the ability to adapt the number of high bandwidth streams at run-time. Hence, the total communication bandwidth of the CCU should be able to change dynamically.

One of the objectives is to identify opportunities for exploiting adaptivity in multi-standard multi-mode wireless communication receivers. In this chapter the following adaptivity features were identified in the UMTS receiver:

- *Standards level*
  Different communication standards can be implemented using the same reconfigurable hardware. Configuring the reconfigurable processing elements into a WCDMA receiver requires less than 10 μs configuration time;

- *Algorithm-selection level*
  Within the WCDMA receiver the functionality of the Rake receiver can be switched to equalization and vice versa. Reconfiguring the processing element, which was performing the equalizer functionality, to Rake receiver functionality costs about 4 μs configuration time;

- *Algorithm-parameter level*
  The number of fingers is one of the parameters that can be changed in the Rake receiver. The number of fingers in the Rake receiver of Section 5.4 can vary between 1 and 4. Adapting the number of fingers means that the MONTIUM TP which implements the Rake receiver has to be partially reconfigured with 24 bytes of data.

This partial reconfiguration costs $120\ ns$ reconfiguration time. Reducing the number of fingers results in lower clock frequencies of the MONTIUM TP, which results in reduced energy consumption.

Another parameter that can be changed is the delay profile of the fingers in the Rake receiver. The reconfiguration time for changing the delay of the fingers is $120\ ns$, because 24 bytes have to be reconfigured.

The third parameter that can be adapted in the Rake receiver is the spreading factor. Changing the spreading factor is performed by partial reconfiguration and loading of the new spreading code.

The pulse-shape filter can be implemented with a FIR filter, which can adapt the number of taps. Changing the number of taps also implies a partial reconfiguration. Consequently, the clock frequency of the processing element can be adapted to the new filter structure, which results in a change in energy consumption.

5.6.2 Discussion

The importance of properly scaling the fixed-point input data in the MONTIUM-based Rake receiver is derived through experiments. In bad channel conditions, the floating-point reference receiver performs slightly better than the MONTIUM-based Rake receiver. This behaviour is due to saturation effects in the MONTIUM Arithmetic Logic Units (ALUs). Saturation of the data in the ALUs can be combated by scaling the input data of the MONTIUM. Correctly scaling the fixed-point input data results in the situation that the floating-point reference receiver and the MONTIUM-based Rake receiver show equal BER versus SNR performance.

The outputs of the experiments confirm the importance of a thorough understanding of both hardware and algorithm. Mapping algorithms on the MONTIUM with knowledge of the DSP algorithm results in an efficient implementation, with similar performance compared to a floating-point implementation.

Comparing Rake receiver implementations mapped on different hardware architectures in a fair way is problematic. The only way to compare a Rake receiver on different hardware architectures is to map the same, identical receiver on different architectures.

General conclusions on the dynamic power consumption of the Rake receiver, mapped on different hardware architectures, can be drawn based on comparing common functions of the different implementations. The dynamic power consumption of Rake functionality implemented in an Application Specific Integrated Circuit (ASIC) is about 3 to 7 times lower compared to the MONTIUM alternative. As expected, the ASIC implementation is more energy efficient than an implementation in reconfigurable hardware, however, the ASIC implementation is fixed and the functionality of the ASIC cannot be changed. The MONTIUM-based Rake receiver, on the other hand, is at least 10 times more energy efficient than a Digital Signal Processor (DSP) implementation. The dynamic power consumption of the alternative Rake receiver implementations is derived for 0.13 μm CMOS technology for all hardware architectures.

Chapter 6

Channel Decoding

This chapter deals with channel decoding for communication systems. The Viterbi and Turbo channel decoding algorithms are studied because they typically appear in a multi-standard wireless communication receiver.

The Viterbi decoder, an important Digital Signal Processing (DSP) kernel in Orthogonal Frequency Division Multiplexing (OFDM) and Wideband CDMA (WCDMA) receivers, is described in detail and mapped on the coarse-grained reconfigurable MONTIUM *architecture. Furthermore, the Turbo decoder is analyzed. Therefore, the Max-Log-MAP (MLM) decoder, an important DSP kernel in the Turbo decoder, is described and mapped to the same coarse-grained reconfigurable* MONTIUM *architecture.*

The Viterbi and Turbo decoder are used to illustrate the feasibility of designing a multi-standard receiver based on a heterogeneous reconfigurable System-on-Chip (SoC) platform. General purpose, fine-grained reconfigurable and coarse-grained reconfigurable processing elements of the heterogeneous reconfigurable SoC are used for implementing the multi-standard receiver functions. The channel decoder functions are mainly performed on coarse-grained processing elements.

Bottlenecks in the integration of de-interleavers in the multi-standard receiver are discussed. The investigation of these issues resulted in the proposal of special memory architectures, which are to be integrated in the heterogeneous reconfigurable SoC.

Major parts of this chapter have been published in [P12, P14, P15, P20].

6.1 Introduction

In the previous chapters, the baseband processing functions of Orthogonal Frequency Division Multiplexing (OFDM) receivers (in Chapter 4) and Wideband CDMA (WCDMA) receivers (in Chapter 5) have been mapped on the heterogeneous reconfigurable System-on-Chip (SoC) architecture which contains coarse-grained MONTIUM processing tiles. To implement adaptive multi-standard wireless communication receivers in heterogeneous reconfigurable SoC architectures, these receivers need to support different channel decoding strategies. Therefore, the template of the heterogeneous SoC, as shown in Figure 3.3, is applied to illustrate the mapping of different channel decoding algorithms. Section 6.2 discusses the basic principles of channel decoding.

The basic characteristics of the Viterbi algorithm are described in Section 6.3. Furthermore, the Viterbi algorithm is mapped on the MONTIUM Tile Processor (TP) in Section 6.3.4. This has been verified by means of Bit Error Rate (BER) performance measurements in Section 6.3.5.

Another channel decoding approach, Turbo decoding, is discussed in Section 6.4. The basic decoder of the iterative Turbo decoding process is mapped on the MONTIUM TP in Section 6.4.4. Experiments on this mapping have been performed in Section 6.4.5.

Interleaving and puncturing are generally performed as part of channel coding in wireless communication systems. In Section 6.5 issues arising from interleaving and puncturing in wireless communication standards are discussed.

The approach of implementing adaptive multi-standard wireless communication receivers capable of handling different channel decoding algorithms is summarized at the end of this chapter in Section 6.6.

Section 6.7 summarizes with conclusions on mapping channel decoding algorithms on heterogeneous reconfigurable SoC architectures.

6.2 Channel decoding

Shannon has proven in [99] that it is possible to reliably send information over a communication channel with a transmission rate, which is limited by the Shannon capacity (or *Shannon limit*). The Shannon limit is the absolute limit, where no improvement on the Bit Error Rate (BER) can be made without increasing the energy per bit. Shannon described this existence theorem, however, he did not give a solution to reliably send information (i.e. Shannon gave an *existence proof*). Many error correcting code schemes have been proposed until now. Among others, convolutional codes are widely used in communication systems as error correcting codes. These error correcting codes enable reliable communication of information over a noisy, distorted communication channel by adding redundant information [68].

A convolutional code is generated by passing the data through a finite state shift register. The contents of the shift register (i.e. the state of the shift register) determines the output code. So, the encoding of information can be represented with a state machine. As an example, Figure 6.1 shows a convolutional encoder and its state machine. The *rate*

of this encoder is $R = 1/2$ as one bit is encoded in two bits. Three different bit-values, i.e. the current value and two old values, are used in this encoder to create the code. Hence, the *constraint length* of the encoder is $k = 3$. The encoder in Figure 6.1 is called Non Systematic Convolutional (NSC) because the input data bits are not directly connected to the output code bits.

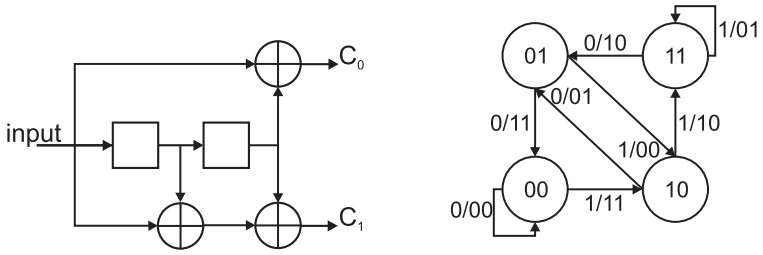

Figure 6.1: *Convolutional encoder (left) and its state machine (right).*

A common representation of the encoder's state is given by a trellis diagram. The trellis diagram simply shows the progression of the state of the encoder for different symbol times. Figure 6.2 depicts part of the trellis for the NSC encoder of Figure 6.1. The trellis shows for each time instant all possible states of the encoder, and all possible state transitions.

The dotted lines in Figure 6.2 denote the state transitions in case a *1* was present at the input of the NSC encoder. The solid lines denote the state transitions in case a *0* was present at the input of the NSC encoder. Suppose the current state of the encoder is *00* and a *0* is present at the input, then the new state of the encoder is *00* and the generated codeword at the output of the encoder is *00*. However, in case a *1* is presented at the input of the encoder, the new state of the encoder is *10* and the generated codeword is *11*. In this way, all the source information is encoded.

Convolutional code decoding algorithms are used to estimate the encoded input information, using a method that results in the minimum possible number of errors. In [112] Viterbi originally described his maximum-likelihood sequence estimation algorithm, commonly known as the Viterbi algorithm. The job of the channel decoder is to estimate the path through the trellis that was followed by the encoder.

Turbo codes, a new family of convolutional codes were proposed in [16, 17]. Figure 6.3 depicts the basic building blocks of the Turbo encoder. The Turbo encoder consists of two Recursive Systematic Convolutional (RSC) encoders and an interleaver (`I` in Figure 6.3). These codes are built using concatenation of two RSC codes and their performance is close to the *Shannon limit*. The recursive codes have a feedback loop in the convolutional encoder, which causes the state of the encoder to be dependent on the state as well as on the input. One encoder receives the input data bits directly, while the second encoder receives the input data bits in a shuffled way. The shuffling is performed by the interleaver. The interleaver operates on input blocks of a fixed length which is specified

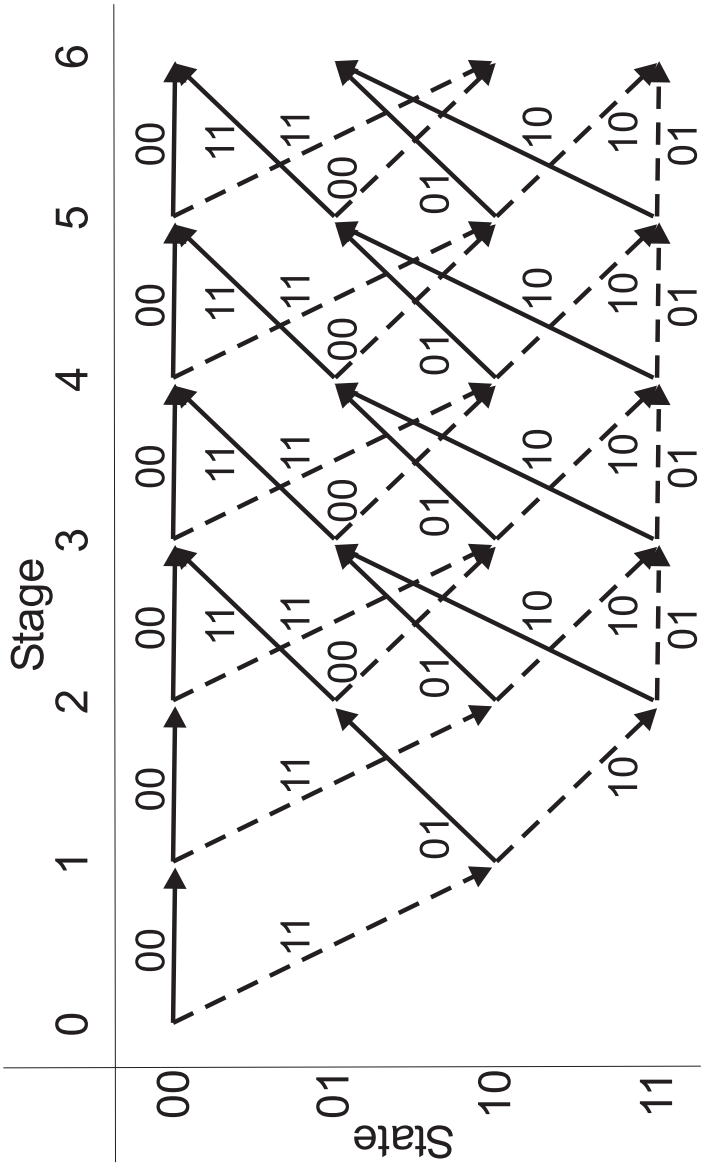

Figure 6.2: *Trellis diagram with 4 states.*

within a specific standard. The interleaver block size is specified between 40 and 5114 bits [1] for the Universal Mobile Telecommunications System (UMTS) standard.

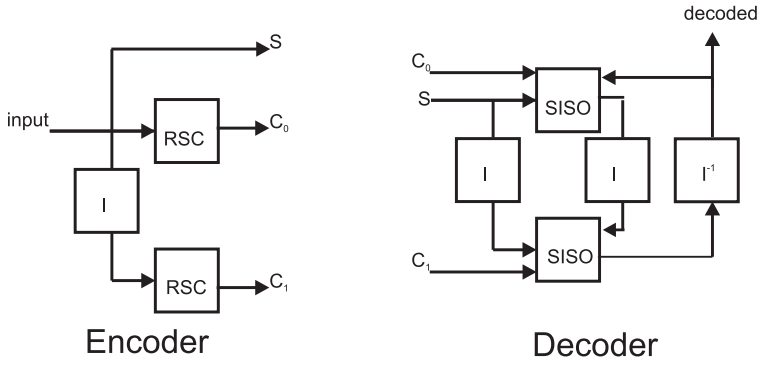

Figure 6.3: *Turbo encoder (left) and decoder (right).*

6.3 Viterbi decoder

Convolutional code decoding algorithms are used to estimate the encoded input information, using a method that results in the minimum possible number of errors. In [112] Viterbi originally described his maximum-likelihood sequence estimation algorithm, commonly known as the Viterbi algorithm. The job of the decoder is to estimate the path through the trellis that was followed by the encoder. Hence, at the receiver side (i.e. the decoder) the same trellis has to be constructed as the one that was used at the transmitter side (i.e. the encoder).

Note that in Figure 6.2, transitions from state *00* to state *00* and *10* are possible as well as from state *01* to state *00* and *10*. These four transitions form a so-called *Viterbi butterfly*. A generalized description of the Viterbi butterflies is given in Figure 6.4. The m in Figure 6.4 denotes the time instant in the trellis (*stage* in Figure 6.2) and S_x gives the state of the encoder (*state* in Figure 6.2). The Viterbi algorithm performs all operations on these butterflies.

The Viterbi algorithm can be divided in three parts: the *branch metric unit*, which performs the branch metric calculation, the *Add Compare Select unit*, which performs the path metric updating, and the *survivor memory unit*, which updates the survivor sequences. These parts are discussed in the following sections. Figure 6.5 explains the three parts of the Viterbi algorithm using an example.

6.3.1 Branch metric calculation

Figure 6.4 shows the generalized Viterbi butterflies, which exist in a trellis. In each butterfly four branches exist. The branches connecting to state $S_{i/2}$ always correspond

to a *0* as decoded output. The branches connecting to state $S_{(i+N)/2}$ always correspond to a *1* as decoded output.

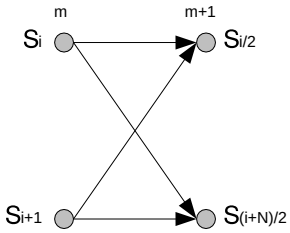

Figure 6.4: *Generalized Viterbi butterfly.*

All four branches in the butterfly have a particular codeword assigned. During *branch metric calculation* the Euclidean distance of the received codeword with the assigned codeword is determined. The *branch metric unit* determines all branch metrics in the trellis. Branch metric calculation is performed on Viterbi butterflies. Hence, the Euclidean distance corresponds to the branch metric. The branch metric keeps the score of a certain hypothesis.

The Euclidean distance is calculated between the received codeword and the codeword, which is assigned to the particular branch. The codeword length of for example a *rate* $R = 1/4$ decoder is 4, resulting in the Euclidean distance:

$$\gamma = |y_0 - c_0|^2 + |y_1 - c_1|^2 + |y_2 - c_2|^2 + |y_3 - c_3|^2, \tag{6.1}$$

where y_x depicts the x^{th} bit of the received codeword and c_x depicts the x^{th} bit of the assigned codeword to the branch.

6.3.2 Path metric updating

Two branches are connected to every output state of the butterfly (refer to Figure 6.4). After the branch metrics are calculated, the path metric at the output states of the Viterbi butterfly can be calculated. The path metrics at the output state are calculated by adding the path metric at the input state and the corresponding branch metric. Hence, the path metrics of two paths, both terminating in the same output state, are determined. The path with the minimum path metric is selected as the *survivor* and the corresponding path metric is assigned to the output state. The example in Figure 6.5 shows that the branch metrics of the branches have been calculated in the *left picture*. In the *middle picture* of Figure 6.5, the path metrics are determined using the path metrics of the input states and the calculated branch metrics. In the middle picture it is shown that for both output states the survivor origins from the lower input state. The path metrics of the survivors for both output states are *3+1=4* and *3+2=5*, respectively.

The *path metric updating* operation requires typically a *min()* operation, selecting the minimum of two values. Moreover, the originating state (S_i or S_{i+1} in Figure 6.4)

has to be determined, since this information is used during *survivor sequence updating*. That operation is known as Add Compare Select (ACS); the path metrics and the branch metrics are added, the results are compared and the survivor is selected.

6.3.3 Survivor sequence updating

When the path metrics for the complete or truncated trellis are calculated, the decoded output bit stream can be generated. Two approaches are often used to record survivor branches: *Traceback* and *Register Exchange* [41, 90]. Basically, the difference between the two approaches is the manner in which information about the survivors is stored during the intermediate steps.

Traceback

The Traceback (TB) approach stores the survivor branch of each state. When the survivor branch of each state is known, the survivor path through the trellis can be reconstructed. Concatenation of decoded output bits in every stage in reversed order of time generates the decoded output sequence of the survivor path through the trellis.

Register Exchange

The Register Exchange (RE) approach stores the entire decoded output sequence for each state during path metric updating. Therefore, in each stage of the trellis the decoded output sequence is known and there is no need to traceback. Moreover, the decoded output sequence can be generated at a high speed. The approach is not power efficient due to frequently copying the decoded output sequence from one stage to the next stage.

The *right picture* in Figure 6.5 shows the RE approach. The decision bit (*0* or *1*) at the output states of the Viterbi butterfly is appended to the bit sequence of the survivor (*10111*).

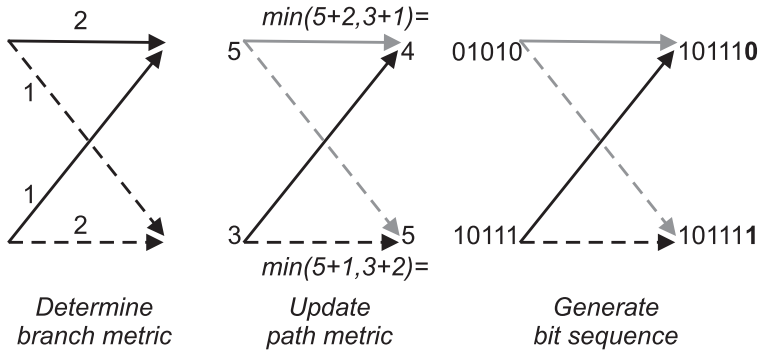

Figure 6.5: *The three parts of the Viterbi algorithm:* branch metric calculation *(left picture)*, path metric updating *(middle picture) and* survivor sequence updating *(right picture).*

```
@start:
   // perform Viterbi algorithm for next 10 stages
   for stage = 1:10
      calculate_branch_metrics();
      update_path_metrics();
      append_decision_bit_to_bit_sequence();
   end;

   // search the state with minimum path metric and
   // look-up from RE contents the survivor sequence
   search_minimum_path_metric();
   generate_minimum_survivor_sequence();

goto @start;
```

Figure 6.6: *Pseudo-code of the implemented Viterbi decoder mapped on the* MONTIUM.

6.3.4 Implementation

We implemented the Viterbi algorithm on the coarse-grained reconfigurable MONTIUM. The mapping of the Viterbi algorithm on the MONTIUM architecture is highly influenced by the specifications of the reconfigurable hardware:

- survivor sequence updating is performed using the RE approach, because bit sequences are stored more efficiently than individual bits in the coarse-grained MONTIUM;
- separate memories for input and output states of the Viterbi butterfly are defined because the memories in the MONTIUM are implemented with single port SRAMs [54];
- in-place computation [90] is not performed, since consecutive *read* and *write* operations disorganize the regular behaviour of the memory operations. The non-regularity results in additional Address Generation Unit (AGU) instructions and memory decoder instructions.

The bit sequence estimation is performed in a regular pattern. Figure 6.6 shows a piece of pseudo-code of the Viterbi implementation, mapped on the MONTIUM.

The implemented Viterbi decoder is universal in the sense that different *rates*, R, and different *constraint lengths*, k, can be handled. Commonly used rates of convolutional codes in communication systems are $R = 1/4$ (e.g. DAB and DRM), $R = 1/3$ (e.g. UMTS) and $R = 1/2$ (e.g. UMTS and HiperLAN/2). The *constraint lengths* in common communication systems are $k = 7$ (e.g. DAB, DRM and HiperLAN/2) and $k = 9$ (e.g. UMTS) [38, 37, 1, 36]. Although our implementation can handle different *rates* and *constraint lengths*, all examples of the Viterbi decoder in the remainder of this chapter are illustrated by a convolutional code that is used in the Digital Audio Broadcasting (DAB) system [38].

Branch metric unit

The *branch metric* unit calculates the Euclidean distances between the received codeword and the codewords, which are assigned to the transitions in the trellis. For a *rate* $R = 1/4$ code, the branch metrics are calculated according to (6.1). The five Arithmetic Logic Units (ALUs) in the MONTIUM are able to calculate the branch metric, based on five coded bits, in one clock cycle, because each ALU calculates the distance of one code bit.

The *rate*, R, of the convolutional code has its impact on the branch metric calculation. The *rate* of the Viterbi decoder can easily be adapted by changing the instructions of the *branch metric* unit in the MONTIUM. Because of the flexibility inside the MONTIUM, one can easily choose to compute a branch metric based on 2, 3, 4 or 5 code bits in parallel. After the branch metrics are calculated, the values are stored in the local registers of the ALUs which can be used for the *Add Compare Select* task.

For each stage of the trellis, new branch metrics have to be calculated. In the DAB system eight different branch metrics have to be computed, because only eight codewords are assigned to state transitions in the trellis for DAB[1]. The different codewords, which exist in the DAB system, are stored in local memory of the MONTIUM. One branch metric, belonging to a codeword of 4-bits wide, is calculated in one clock cycle by 4 ALUs. Hence, per stage of the trellis one needs 8 clock cycles for branch metric calculation. The calculated branch metrics are stored in local registers of the ALUs. These branch metrics are directly used by the ACS unit.

Add Compare Select unit

In the *Add Compare Select (ACS)* unit, the path metrics at the output states of the Viterbi butterfly are updated. The ACS operation is performed on the Viterbi butterflies. The number of Viterbi butterflies in a trellis depends on the *constraint length*, k, of the convolutional code. The number of Viterbi butterflies is always equal to $\frac{1}{2} \cdot 2^{k-1}$. Hence, the Viterbi decoder in a DAB system with *constraint length* $k = 7$ has to handle 32 butterflies per stage of the trellis. So, the path metrics of 64 states have to be updated for that system.

In each butterfly the path metrics of the four possible paths in the butterfly are calculated. For each output state of the butterfly the path with the smallest path metric is selected as the survivor. The values of the path metric at the output states are stored in local memory and these values are used as input for the Viterbi butterflies in the next stage of the trellis.

The *add-compare* operation that has to be performed in the Viterbi butterflies can easily be mapped on the MONTIUM Tile Processor (TP). This requires only an addition and a *min()* operation, selecting the minimum of two values.

The path metrics are not only updated in the ACS unit, but the decoded bits at the output states are also generated. For every upper state, $S_{i/2}$, of the Viterbi butterfly a *0* is generated as output bit and for every lower state, $S_{(i+N)/2}$, of the Viterbi butterfly a *1*

[1] In general $2^{\frac{1}{R}}$ codewords are assigned to state transitions in the trellis, but in the DAB encoder two codebits are generated by the same generator polynomial [38].

is generated (refer to Figure 6.4). Moreover, the Viterbi decoder also has to know what the surviving path in the butterfly is. In other words, the originating state of the surviving path through the Viterbi butterfly has to be determined. This is a task of the *select* part in the ACS unit.

In the MONTIUM architecture, described in [54], the *compare-select* operation can be performed by using conditional calls in the sequencer program of the MONTIUM. Drawback of this approach is that only one *select* operation can be performed at a time. In the Viterbi butterfly two *compare-select* operations have to be done; one for each output state of the Viterbi butterfly.

The problem, which arises with the *compare-select* operation, is that the operation merges the data and the control path. Control-oriented functions are actually not part of the MONTIUM algorithm domain. Since the *compare-select* operation is very important in the Viterbi algorithm, we propose to add the *compare-select* operation to the ALUs of the MONTIUM. This enables the *compare-select* operation to run independently from the sequencer instructions. Consequently, the *compare-select* operation can run in parallel in all 5 ALUs of the MONTIUM.

Conditional calls in the MONTIUM sequencer program are typically based on the status bit (SB) feedback information, which comes from the ALUs. The SB information is created in *level 1* of the ALU, as described in Section 3.5.1. To implement the *compare-select* operation in hardware, the SB information should be used as a control to create a conditional multiplexer. Figure 6.7 shows a schematic overview of the ACS operation supported in hardware by the MONTIUM. The multiplexer in Figure 6.7 can be configured to select its input source based on the provided SB control signal. The SB signal is schematically shown in Figure 6.7 by dotted signal lines. The dotted lines represent the SB information from the function units. A condition check on the SB information generates the control signal for the conditional multiplexer in *level 2* of the ALU. As an example, the data out of *function unit 3* or *function unit 4* can be selected based on the value of the SB. The SB information is generated as result of e.g. a *compare* operation, which is performed in one of the function units.

The *compare-select* functionality can be added to the MONTIUM with a minor change in the existing architecture. As a result of this minor change in the architecture a larger coverage of the wireless communication domain is obtained. Baseband and channel coding algorithms of different wireless communication standards can now be performed on the same coarse-grained reconfigurable MONTIUM architecture.

With the ACS hardware support implemented in the ALU of the MONTIUM multiple ACS operations can be performed in parallel. In each Viterbi butterfly, two ACS operations have to be performed in parallel; one ACS operation for each output state of the Viterbi butterfly.

Memory unit

For the Viterbi algorithm two kinds of information need to be stored. Firstly, the *path metric* of each state in the trellis has to be stored. Secondly, the *decoded bit sequence* for

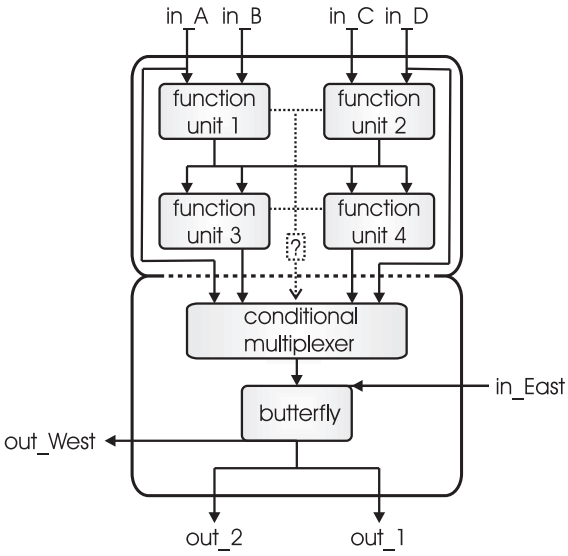

Figure 6.7: *Schematic overview of one* MONTIUM *ALU with hardware ACS support.*

each state in the trellis has to be stored. In [90] these kinds of information are referred to as *scores* and *hypotheses*, respectively.

The memory organization is a main bottleneck in the design of the Viterbi decoder. The Viterbi butterfly, as shown in Figure 6.4, requires two path metrics as input and produces two path metrics as output. Moreover, the decoded bit sequence is generated for the two output states. In addition, when the RE approach is applied for survivor sequence updating, the decoded bit sequences of the input states of the Viterbi butterfly are also required as an input.

Path metrics Since many values have to be read and written instantaneously, we chose to read the input information from two separate memories and to write the results back to another pair of memories. So, one pair of memories acts as input producer and another pair of memories acts as output consumer. The functions of the memory pairs are interchanged in the next stage of the trellis. The Viterbi implementation on the MONTIUM uses in total four local memories in order to store all path metrics; two memories store the old path metrics and two store the new path metrics[2]. This implementation with separate input and output memories strikes with the ideas in [90], where in-place computation is proposed.

For the MONTIUM architecture, and for coarse-grained reconfigurable hardware in general, one looks for highly regular processing patterns in the algorithms that have to be mapped on the target architecture. The in-place computation [90] minimizes the re-

[2] The Fast Fourier Transform (FFT) butterflies are mapped on the MONTIUM in a similar manner.

quired memory size, but the organization of the memory addressing becomes irregular for different stages in the trellis. Consequently, many memory addressing instructions are required to map the in-place computation on the MONTIUM. Furthermore, many local memories are already available in the MONTIUM, which yields enormous memory bandwidth and eliminates the need for in-place computation. The local memories inside the MONTIUM are implemented with single port SRAMs, which makes the in-place computation approach also less attractive.

Survivor sequences The other kind of information, the survivor sequences, are stored in memory using the RE approach. The RE approach for storing the survivor sequences is applied, because bit sequences can be stored more efficiently than single bits, as the data path of the MONTIUM is 16-bits wide.

The length of the survivor sequence depends on the *decision depth* used in the Viterbi decoder. The *decision depth* indicates how many states in the trellis have to be considered before all minimum survivor paths have merged to the same decoded output. This means that a delay exists between path updating and generating the decoded output bits: The path metrics are updated for the current stage of the trellis. For this stage the minimum path metric is determined and, accordingly, the minimum survivor sequence is selected. The final decoded output bits are the bits in the minimum survivor sequence that belong to the stage in the trellis that occurred a number of *decision depth* stages earlier than the current stage. Hence, the delay between path metric updating and decision bit output is *decision depth* trellis stages. As a rule of thumb, a *decision depth* of five times the *constraint length* is in practice sufficient. When puncturing is applied to the convolutional code, the *decision depth* becomes larger [117].

Since the survivor sequences can only be stored in sequences of at most 16 bits in the MONTIUM, the entire survivor sequence should be stored in multiple memory addresses when the *decision depth* of the Viterbi decoder is larger than 16. The advantage of such an approach is that we do not need to handle all the bits of the survivor sequence (i.e. loading and storing the entire survivor sequence during survivor sequence updating). By using memory pointers only the last part of the survivor sequence has to be dealt with.

Figure 6.8 gives an overview of the memory organization of the RE contents in the MONTIUM memory. In one memory address, part of the survivor sequence is stored together with a pointer value. The pointer value directs to the memory location containing the remaining part of the survivor sequence. For the DAB system the memory pointers are 6-bits wide, so 10 bits in the memory are available to store the decision bits.

In the example of Figure 6.8 the Viterbi algorithm processes 10 stages of the trellis during one *phase* of the RE survivor sequence updating process. A memory block of 64 addresses has been allocated for each *phase* in case of DAB. These 64 memory locations are used to store the 10 bits of the RE survivor sequences belonging to the 64 corresponding states. The survivor sequences are appended by 1 bit for each stage of the trellis. Once the survivor sequence is 10 bits long, a new block of memory is allocated. Every block of 64 memory locations is assigned to a *phase* in the RE survivor sequence updating process. The example in Figure 6.8 shows that the survivor sequence of 3 *phases* is stored in

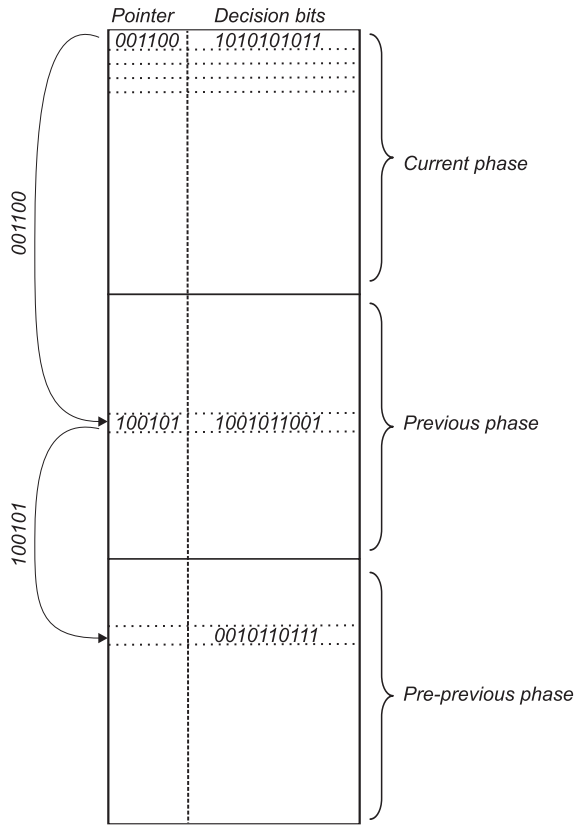

Figure 6.8: *Schematic overview of the RE memory organization implemented with pointers.*

the memory. Hence, in total 30 stages of the trellis have been processed. In the example the survivor of *State 0* has been selected as minimum survivor. Thus, the decoder assigns *0010110111 1001011001 1010101011* to be the minimum survivor sequence.

However, not the entire minimum survivor sequence is used for generating the decoded output bits. The *decision depth* parameter of the Viterbi decoder determines which part of the minimum survivor sequence is used to generate the decoded output bits. When the decision depth is set to 20 and the minimum survivor sequence in Figure 6.8 is used, the Viterbi decoder outputs the sequence *0010110111* as decoded bits during the current phase of the RE survivor sequence updating process.

The survivor sequences are stored using the same approach as the path metrics. The survivor sequences of the input states and the output states are stored in separate pairs of memory. Thus, totally four memories in the MONTIUM are used to store all survivor sequences in the trellis. Consequently, eight of the ten local memories are employed to store all kinds of information in the Viterbi decoder.

Mapping on the MONTIUM

Figure 6.9 shows the mapping of the main loop for Viterbi butterfly operations, using the ALUs with ACS support: the mapping only shows the *path metric updating* and *survivor sequence updating*. Hence, the branch metrics have already been calculated and are temporarily stored in the local registers (i.e. register banks *A* and *C*) of ALU1 and ALU3. The path metrics are updated using ALU1 and ALU3. ALU5 appends one decision bit position to the survivor sequences of the two input states by shifting the entire survivor sequence one position to the left. Register banks *B* and *D* of ALU5 already keep the number of bits that have to be shifted (i.e. *1* in this case). ALU2 and ALU4 select the minimum survivor sequence using the conditional multiplexer based on the information from the *path metric updating* process in ALU1 and ALU3, respectively. ALU2 generates the survivor sequence and decision bit *0* that is associated with the upper state of the Viterbi butterfly (refer to Figure 6.4). ALU4 generates the survivor sequence and decision bit *1* for the lower state of the Viterbi butterfly.

The path metrics associated with the states are stored in MEM1, MEM2 and MEM5, MEM6. One memory pair stores the old state metrics (i.e. the input states of the Viterbi butterfly) while the other memory pair stores the new state metrics (i.e. the output states of the Viterbi butterfly). The survivor sequences are stored in the memory pairs MEM3, MEM4 and MEM7, MEM8. The mapping in Figure 6.9 uses MEM10 to store the encoded bit sequence. These encoded bits are the input for the Viterbi decoder and are used to calculate the branch metrics.

The Viterbi decoder in Figure 6.9 uses the block communication principle but can easily be modified to streaming communication. In the streaming alternative, the encoded input bits are not temporarily stored in memory but directly streamed via the global busses into the registers of the ALUs.

Note that the *path metric updating* and *survivor sequence updating* processes are pipelined: a decision on the origin of a survivor is made based on the result of the *path metric updating*.

Configuration The total configuration size of the MONTIUM Viterbi implementation is $1\,356$ bytes. This configuration can be written in the MONTIUM's configuration memory in 6.78 μs when the configuration clock frequency is 100 MHz.

Once the MONTIUM is configured as Viterbi decoder, only partial reconfiguration has to be performed to adjust the *constraint length*, *decision depth* or *rate*.

The constraint length defines the number of states in the trellis. Hence, the memory organization in the MONTIUM is influenced by changing the constraint length (e.g. reconfiguring the AGU instructions). Adjusting the rate of the Viterbi decoder directly influences the branch metric calculation. Thus, ALU instructions have to be reconfigured when branch metrics for another code rate have to be computed. The decision depth of the Viterbi decoder defines the number of stages that have to be processed before any decoded bit is generated. So, the decision depth directly defines the number of RE phases (Figure 6.8) that need to be stored in the MONTIUM memories. The part of the MONTIUM sequencer program that is responsible for look-up of the decision bit sequence in the RE

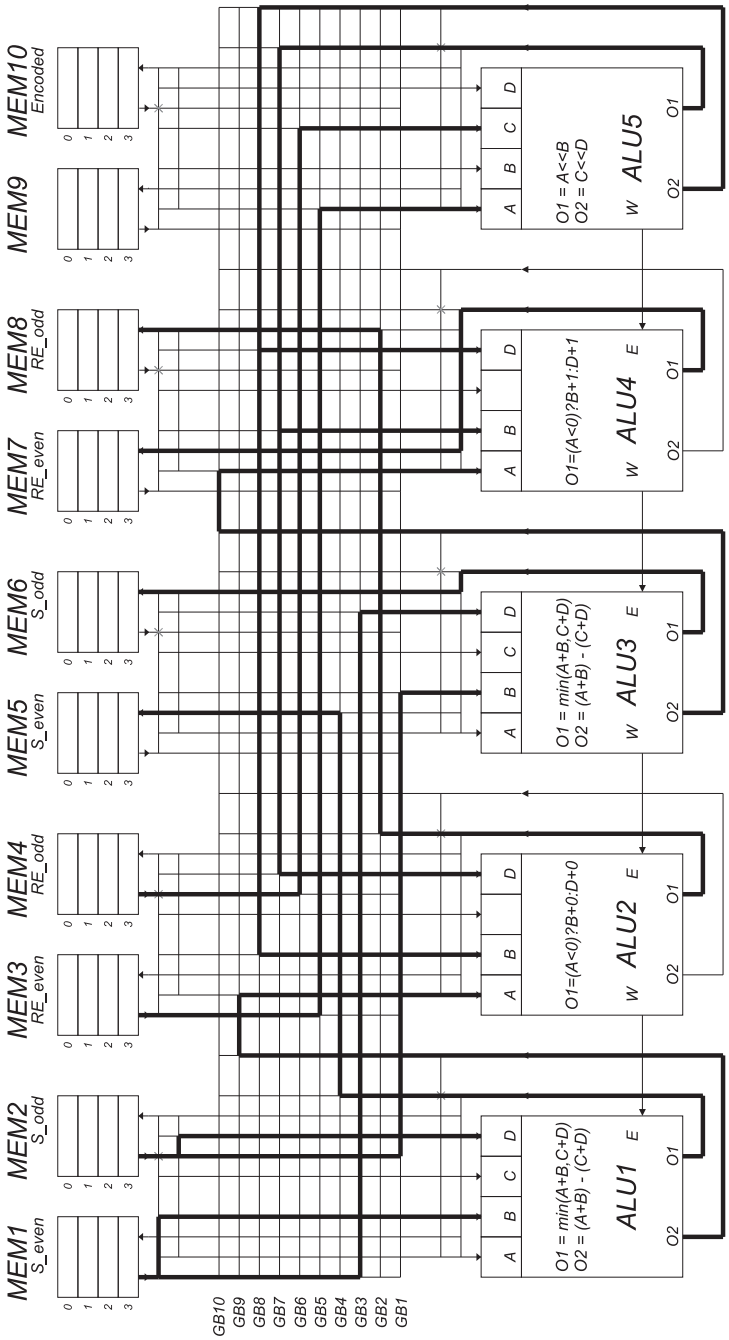

Figure 6.9: *Main loop of Viterbi butterfly operations mapped on the* MONTIUM.

Table 6.1: *Cycle-count of the Viterbi decoder mapped on the* MONTIUM *for DAB.*

	Branch metric calculation [cycles/stage]	Path metric updating [cycles/stage]	Survivor decision look-up [cycles/look-up]
DAB decoder $d = 50, k = 7, R = 1/4$	8	34	50
general decoder d, k, R	$2^{\frac{1}{R}-1}$	$2^{k-2}+2$	$2^{k-2}+3+\frac{d}{16-(k-1)}\cdot 3$

d = decision depth, k = constraint length, R = rate

contents needs to be reconfigured (i.e. changing the number of look-up iterations) when the decision depth is modified.

Especially the *decision depth* depends heavily on the conditions of the wireless channel. Thus, adjusting the *decision depth* can be typically performed at run-time via dynamic reconfiguration.

Throughput The number of clock cycles required for Viterbi decoding mapped on the MONTIUM are summarized in Table 6.1. The number of clock cycles for branch metric calculation and path metric updating are given per stage of the trellis. In the implemented Viterbi decoder, which complies with the DAB standard, always 10 bits are generated during the survivor decision phase.

Whenever 10 stages of the trellis have been processed, the Viterbi decoder decides on the decoded bit sequences of the foregoing stages. Hence, after processing 10 stages of the trellis (referred to as *phase* in Figure 6.8), the Viterbi decoder has to search for the bit sequence of the path with the minimum path metric. This procedure first requires a search operation through the 64 path metrics, looking for the minimum path metric, and costs 35 additional clock cycles. The look-up operation in the RE contents using the memory pointer approach requires 15 additional clock cycles.

Thus, 10 decision bits are generated in $10 \times (8 + 34) + 50 = 470$ clock cycles. On average 47 clock cycles are required to decode one output bit. The output rate of the DAB Viterbi decoder implemented on the MONTIUM is 2.1 $Mbps$ when the clock frequency of the MONTIUM is 100 MHz. This rate is sufficient for DAB, which requires an output rate of 1.8 $Mbps$.

The number of clock cycles that are needed for the Viterbi decoder mapped on the MONTIUM in the general case with *constraint length*, k, *rate*, R, and *decision depth*, d, are also included in Table 6.1.

Energy consumption The normalized power consumption of the MONTIUM during Viterbi decoding is 0.309 mW/MHz, which indicates that the consumed energy per clock cycle is 309 pJ. Using the results from Table 6.1 we known that the DAB Viterbi

Table 6.2: *Energy/technology comparison of different Viterbi decoder implementations.*

	Energy consumption $[nJ/bit]$	CMOS technology $[nm]$
ASIC	1.2	130
MONTIUM	15	130
ARM9	5 000	130

decoder uses on average 47 clock cycles per bit decision. Hence, the energy consumption of the implemented Viterbi decoder, mapped on the MONTIUM, is 14.5 nJ/bit.

Energy estimations of the Viterbi algorithm implemented on General Purpose Processors (GPPs) were reported in [61]. Estimates of the energy consumption are given for a DAB Viterbi decoder in the ARM9 architecture. From simulations we conclude that the energy consumption of the Viterbi decoder implemented on the ARM9 is about 5 $\mu J/bit$. Moreover, the ARM9 does not have enough processing power to deliver an output rate of 1.8 $Mbps$ for DAB.

Among others, a hardwired Viterbi decoder for DAB was implemented in an Application Specific Integrated Circuit (ASIC) by Atmel. From discussions with developers from Atmel we know that their implementation of the Viterbi decoder uses about 8 mW in 0.21 μm technology with an output rate of 1.8 $Mbps$. The average energy consumption of their implementation is about 4.4 nJ/bit. When we scale their implementation to 0.13 μm technology, the decoder will use about 2.2 mW. Hence, the average energy consumption will be about 1.2 nJ/bit.

Table 6.2 summarizes the energy consumption of the different Viterbi decoder implementations. Detailed information on the power estimation experiments of the MONTIUM-based Viterbi decoder is given in Appendix B.

6.3.5 Verification

The functional behaviour of the implemented Viterbi decoder has been verified by means of hardware / software co-simulations. Figure 4.17 shows the co-simulation environment with MATLAB [71] and MODELSIM [73], which has been used for functional simulations. The functionality of the Viterbi decoder mapped on the MONTIUM TP has been verified against a reference implementation of the decoder implemented in MATLAB.

We simulated a floating-point Viterbi decoder in MATLAB that is used as a reference decoder to compare the decoding accuracy. The reference implementation is compared against the RE approach Viterbi decoder, which has been mapped on the MONTIUM TP. The performance of the Viterbi decoder is expressed in BER of the encoded data (i.e. the information to be decoded, the Viterbi decoder input) versus the BER of the decoded data (i.e. the Viterbi decoder output). The Viterbi decoder has been evaluated with DAB settings. Hence, $10\,000$ encoded bits are simulated, resulting in $2\,500$ decoded bits. The Viterbi decoders were both configured for a *decision depth*, $d = 64$. All simulations that are reported in this section are performed with mentioned settings.

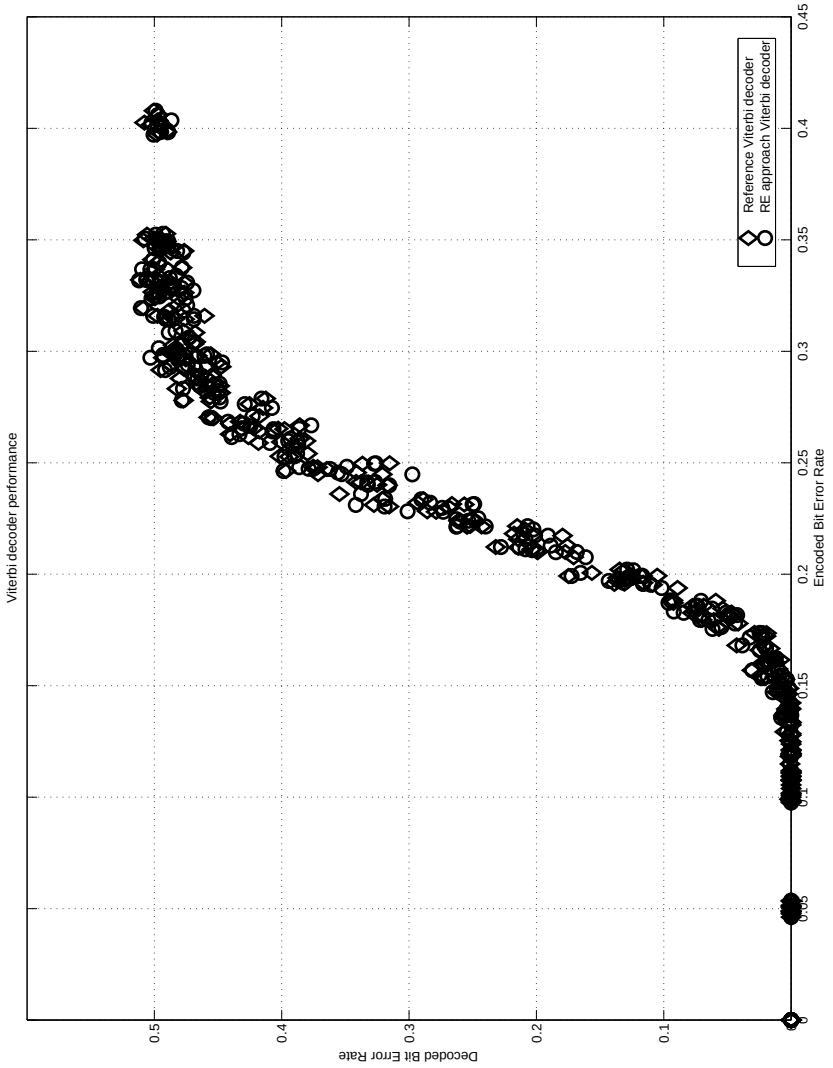

Figure 6.10: *The BER performance of the* MATLAB *reference Viterbi decoder and the applied RE approach Viterbi decoder.*

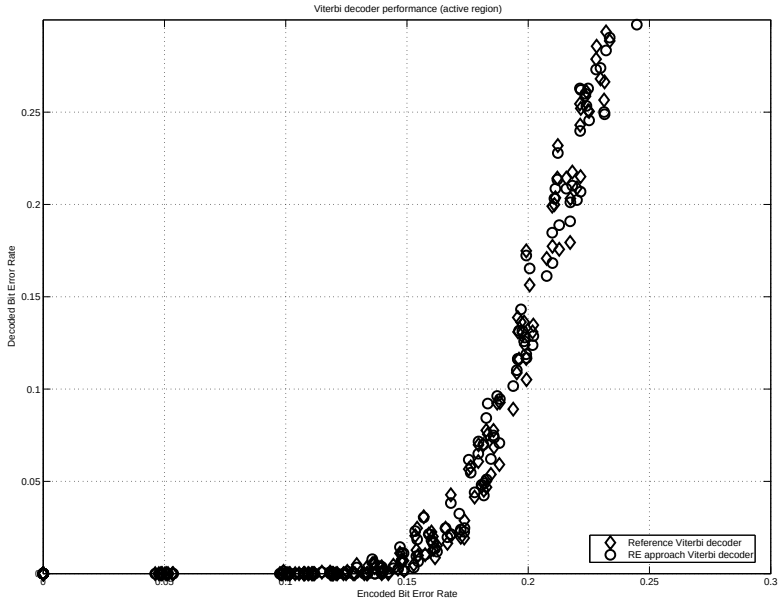

Figure 6.11: *The BER performance in the active region of the* MATLAB *reference Viterbi decoder and the applied RE approach Viterbi decoder.*

Figure 6.10 shows the performance of both the reference Viterbi decoder and the RE approach Viterbi decoder, which is mapped on the MONTIUM TP. The graph shows the BER of the Viterbi output (on the *y-axis*) versus the BER of the Viterbi input (on the *x-axis*). All individual simulation points are depicted in Figure 6.10. The reference Viterbi decoder and the MONTIUM-based Viterbi decoder show equal average performance in terms of BER.

The graph in Figure 6.10 shows the decoding capabilities of the Viterbi decoder. The Viterbi decoder is capable to reduce the number of bit errors when the BER of the encoded input data is less than 0.22. This active region of the Viterbi decoder is depicted in Figure 6.11.

For better comparison of the decoding capabilities of both the reference Viterbi decoder and the RE approach Viterbi decoder, Figure 6.12 compares the decoded BER of the RE approach Viterbi decoder (on the *y-axis*) against the decoded BER of the reference Viterbi decoder (on the *x-axis*). On average both implementations show similar decoding performance.

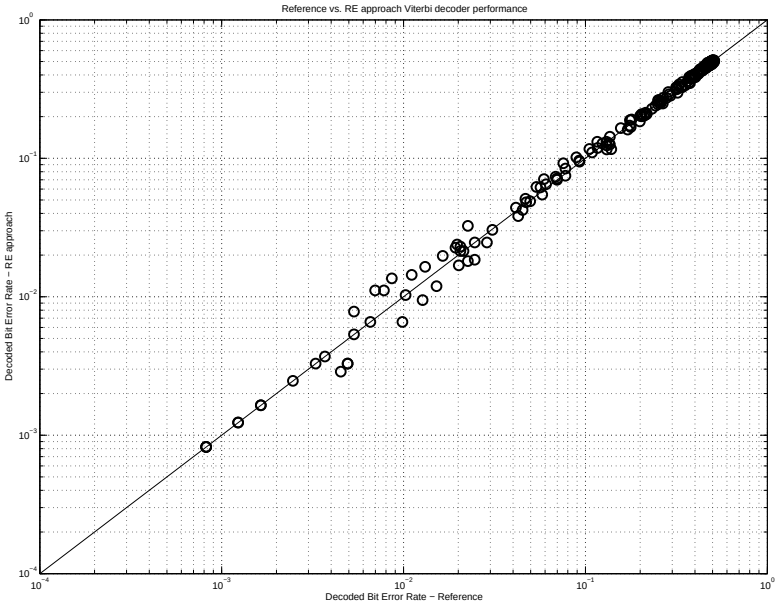

Figure 6.12: *The BER performance of the* MATLAB *reference Viterbi decoder versus the applied RE approach Viterbi decoder.*

6.4 Turbo decoder

Turbo codes, which are more formally known as parallel-concatenated Recursive Systematic Convolutional (RSC) codes, were first proposed by Berrou in [17] and [16]. The basic idea of Turbo error correcting codes is to include redundant information in series of bits (i.e. parity information). The redundant information (i.e. parity information) is generated by the RSC encoder blocks in the Turbo encoder (Figure 6.3). The interleaver in the Turbo encoder ensures that the parity information is spread over the entire message. The interleaver should reorder the parity information in such a way, that the new sequence is uncorrelated with the original sequence as much as possible to maximize the transmitted information over a noisy channel.

Decoding of Turbo codes is performed in an iterative way. The Turbo decoder consists of a de-interleaver (identified with $\mathtt{I}^{-1}$) and two decoder blocks, as depicted in Figure 6.3. Those decoder blocks are mostly referred to as Soft-Input-Soft-Output (SISO) decoders. Each SISO decoder estimates the Log-Likelihood Ratio (LLR), which denotes the logarithm of the probability that a *1* is transmitted divided by the probability that a *0* is transmitted based on its input signals. The input signals of the SISO decoder are the *parity input* and *systematic input*, which is also called the *intrinsic information*, and the

feedback information derived by the previous SISO decoder, which is called the *extrinsic information*. The systematic input and parity input are labelled in Figure 6.3 as S and C_x, respectively. *Intrinsic information* denotes the information that is delivered to the decoder by the channel, whereas *extrinsic information* is obtained by the decoding process. Each iteration of Turbo decoding adds extra information, i.e. *extrinsic information*, to make a better decision on the decoded bit stream.

The SISO decoders in the Turbo decoder can be implemented using several algorithms [95]. The Log-MAP algorithm is the best performing algorithm for a Turbo decoding SISO. This algorithm is, however, very computationally intensive because a large number of logarithms and exponents have to be calculated. We mapped the simplified version of the Log-MAP algorithm, the Max-Log-MAP (MLM) algorithm, on the MONTIUM. The MLM algorithm uses only additions and maximum operations. This algorithm has a regular, optimized structure and achieves near-optimal BER.

The MLM algorithm consists of three processing phases: *forward recursion*, *backward recursion* and *LLR calculation*. The information from the forward and backward recursion are used to estimate the LLR information. Because the LLR calculation can be done while the backward recursion is performed, the backward estimations do not need to be stored in memory. However, all the forward estimations have to be stored in memory. Hence, in order to be compliant with the Third Generation (3G) UMTS standard, at most 5114×8 forward estimates have to be stored for full block length. The required memory to store the forward estimates can be reduced by applying the *sliding window* approach [29]. This approach divides a block of information into smaller blocks, *windows*, on which the algorithm is applied. The number of forward estimates that need to be stored is now equal to the window length.

6.4.1 Branch metric calculation

In the forward recursion, a branch metric is computed for each possible state transition in the trellis. The branch metric, $\gamma_{ij}[k]$, of the transition from state i to state j at time k is:

$$\gamma_{ij}[k] = S_{ij}[k] \cdot L[k-1] + Q_{ij}[k] \cdot P[k-1], \tag{6.2}$$

where $S_{ij}[k]$ is the data bit and $Q_{ij}[k]$ is the parity bit associated with a transition from state i to state j at time k. $L[k-1]$ denotes the intrinsic input, and $P[k-1]$ gives the parity input. The *intrinsic input* consists of the systematic bits and the feedback information (i.e. *extrinsic information*) generated by earlier SISO decoding iterations.

6.4.2 Forward and backward recursion

The operations of (6.3) and (6.4) are performed on Viterbi butterflies, which can generally be identified in the trellis of convolutional codes. The generalized Viterbi butterfly notation used in (6.2), (6.3), (6.4) and (6.5) is shown in Figure 6.13. (6.3) and (6.4) show the formulas to calculate the forward and backward state metrics, respectively. During

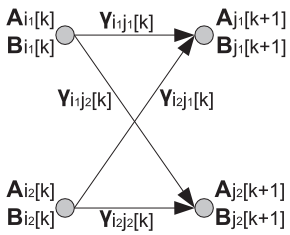

Figure 6.13: *Generalized Viterbi butterfly with notation used in (6.2), (6.3), (6.4) and (6.5).*

forward recursion the trellis is read in forward direction, while during backward recursion the trellis is read in the opposite direction. $A_x[k]$ denotes the forward state metric of state x at time k, and $B_x[k]$ identifies the backward state metric of state x at time k:

$$A_j[k] = \max(A_{i_1}[k-1] + \gamma_{i_1 j}[k], A_{i_2}[k-1] + \gamma_{i_2 j}[k]), \tag{6.3}$$

$$B_i[k] = \max(B_{j_1}[k+1] + \gamma_{i j_1}[k+1], B_{j_2}[k+1] + \gamma_{i j_2}[k+1]) \tag{6.4}$$

6.4.3 Log-Likelihood Ratio calculation

The LLR calculation is performed immediately after the backward recursion. The LLR calculation is done according to:

$$\begin{aligned} \lambda[k] &= \max_{i,j} \left(A_i[k] + B_j[k+1] + \gamma_{ij}[k]\right)\Big|_{S_{ij}[k]=1} \\ &- \max_{i,j} \left(A_i[k] + B_j[k+1] + \gamma_{ij}[k]\right)\Big|_{S_{ij}[k]=0} \end{aligned} \tag{6.5}$$

For each transition associated with a systematic *1*, the *forward state metric* of the originating state is added with the *backward state metric* of the destination state and the *branch metric* of the transition. For each transition associated with a systematic *0* the same operations are performed.

The LLR represents the likelihood of the decoded bit being either *1* (positive LLR) or *0* (negative LLR). By subtracting the original *intrinsic information*, $L[k-1]$, from the LLR, $\lambda[k]$, the *extrinsic information* can be obtained. This *extrinsic information* adds extra information to make a better decision on the decoded bit stream. The *extrinsic information* can be used by the SISO in the next Turbo iteration.

```
@start:
   // perform MLM forward recursion
   for stage = 1:cycles
      calculate_forward_metrics();
   end;
   normalize_forward_metrics();

   // perform MLM backward recursion and LLR
   for stage = cycles:1
      calculate_backward_metrics();
      calculate_LLR();
   end;
   normalize_backward_metrics();

goto @start;
```

Figure 6.14: *Pseudo-code of the Max-Log-MAP algorithm mapped on the* MONTIUM.

6.4.4 Implementation of Max-Log-MAP

The operations to be performed for the calculation of (6.3), (6.4) and (6.5) can be done in a very regular way. So, these operations are suitable to be mapped on the MONTIUM architecture. The MLM calculations are done in a regular fashion and are structured in the MONTIUM using loops. Figure 6.14 shows the program flow of the MLM algorithm in pseudo-code, which is mapped on the MONTIUM.

Mapping on the MONTIUM

In the MONTIUM mapping of the MLM algorithm, all branch metrics are calculated explicitly. So, the forward metrics, $A_x[k]$, are calculated by evaluating (6.2) and (6.3) with given systematic, $S_{ij}[k]$, and parity, $Q_{ij}[k]$, information that is generated for a given Turbo encoder. Hence, the forward recursion is *explicitly* determined in the MONTIUM in the following manner:

$$\begin{aligned}
A_0[k+1] &= \max(A_0[k], A_1[k] + L[k] + P[k]) \\
A_1[k+1] &= \max(A_2[k] + L[k], A_3[k] + P[k]) \\
A_2[k+1] &= \max(A_4[k] + P[k], A_5[k] + L[k]) \\
A_3[k+1] &= \max(A_6[k] + L[k] + P[k], A_7[k])
\end{aligned}$$

$$\begin{aligned}
A_4[k+1] &= \max(A_0[k] + L[k] + P[k], A_1[k]) \\
A_5[k+1] &= \max(A_2[k] + P[k], A_3[k] + L[k]) \\
A_6[k+1] &= \max(A_4[k] + L[k], A_5[k] + P[k]) \\
A_7[k+1] &= \max(A_6[k], A_7[k] + L[k] + P[k])
\end{aligned}$$

Calculating the forward recursion takes only two clock cycles per decision bit, because four *max()* operations can be performed on the MONTIUM in parallel. The operations are scheduled on the ALUs of the MONTIUM in such a way that the input state metrics, i.e. $A_j[k]$ and $B_i[k]$, do not have to be moved twice. Hence, $A_0[k+1]$ and $A_4[k+1]$ are calculated on the same ALU in consecutive clock cycles. The mapping of the forward recursion on the MONTIUM TP is shown in Figure 6.15 and 6.16. The forward state metrics, $A_0[k]$ to $A_7[k]$, are stored in the memories MEM1 to MEM8, respectively. The operations in Figure 6.15 and 6.16 are performed consecutively in a loop until the entire forward recursion is completed. Thus, $A_0[k+1]$ is calculated in Figure 6.15 and $A_4[k+1]$ in Figure 6.16. The forward recursion is controlled by a *soft counter* that is updated in ALU5 (in Figure 6.15).

The backward recursion can be performed in the same way, only the trellis has to be read in reverse direction:

$$\begin{aligned}
B_0[k-1] &= \max(B_0[k], B_4[k] + L[k-1] + P[k-1]) \\
B_2[k-1] &= \max(B_1[k] + L[k-1], B_5[k] + P[k-1]) \\
B_4[k-1] &= \max(B_2[k] + P[k-1], B_6[k] + L[k-1]) \\
B_6[k-1] &= \max(B_3[k] + L[k-1] + P[k-1], B_7[k])
\end{aligned}$$

$$\begin{aligned}
B_1[k-1] &= \max(B_0[k] + L[k-1] + P[k-1], B_4[k]) \\
B_3[k-1] &= \max(B_1[k] + P[k-1], B_5[k] + L[k-1]) \\
B_5[k-1] &= \max(B_2[k] + L[k-1], B_6[k] + P[k-1]) \\
B_7[k-1] &= \max(B_3[k], B_7[k] + L[k-1] + P[k-1])
\end{aligned}$$

The operations of the backward recursion are scheduled in a similar way as the forward recursion; $B_0[k-1]$ and $B_1[k-1]$ are calculated on the same ALU in consecutive clock cycles. The operations of the backward recursion have been mapped on the MONTIUM TP as depicted in Figure 6.17 and 6.18.

Actually, the backward recursion operations are combined with the LLR calculations. The *backward recursion* and the *LLR operations* for one stage of the trellis are combined, as the *backward state metrics* have not to be stored in the memory of the MONTIUM. The LLR, as defined in (6.5), is explicitly calculated in the MONTIUM by a set of calculations. The LLR for one stage of the trellis is calculated on the MONTIUM in 4 clock cycles. Hence, the operations mapped on the MONTIUM in Figure 6.17 and 6.18 are the first two clock cycles of the combined *backward recursion* and *LLR calculation*. Those two clock cycles, in which the *backward state metrics* are calculated, are followed by 4 clock cycles performing the LLR operations. The set of combined backward recursion and LLR operations is controlled by a *soft counter* that is updated in ALU5 (in Figure 6.18).

Figure 6.15, 6.16 and Figure 6.17, 6.18 show that identical ALU operations are applied in both *forward recursion* and *backward recursion*, respectively. The input values of the MLM decoder, i.e. the systematic and parity information ($L[k]$ and $P[k]$), are read

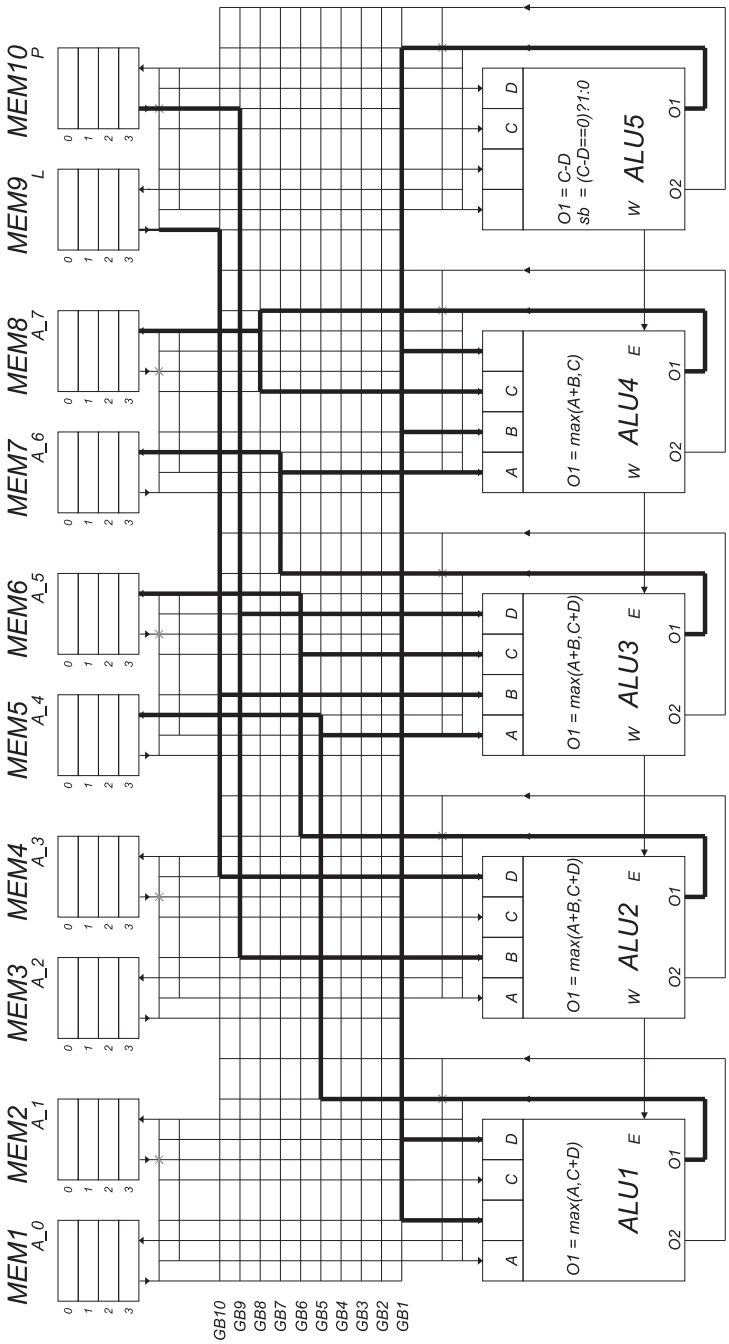

Figure 6.15: *First operation cycle of forward recursion of the MLM algorithm mapped on the* MONTIUM.

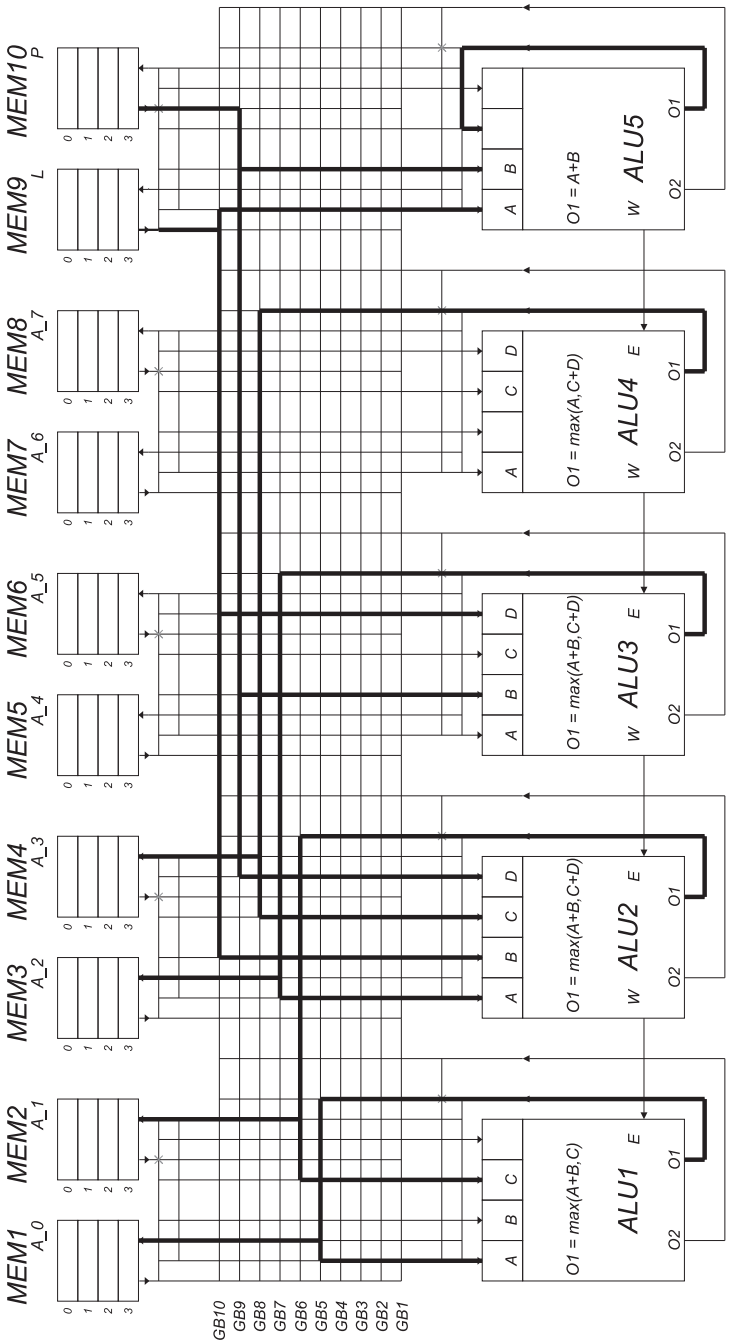

Figure 6.16: *Second operation cycle of forward recursion of the MLM algorithm mapped on the* MONTIUM.

from MEM9 and MEM10 during the forward recursion. The LLR result, $\lambda[k]$, after backward recursion and LLR calculation is finally stored in MEM9 of the MONTIUM.

The MLM algorithm is relatively regular, but several different combinations of data are needed for the calculations. During forward and backward recursion, alternating register locations and memories are used in consecutive clock cycles. Thus, the register allocation and global bus allocation, as seen in Figure 6.15 through 6.18, need to be done carefully and is demanding for MONTIUM decoder instructions.

Input scaling and normalization The numeric range and frequency of normalization obviously control how soon the values of the *state metrics* will saturate. Saturation might remove the difference between state metrics and make decoding impossible. Therefore, normalization is applied every few recursion steps. Another important choice is, obviously, the scaling factor that is applied on the input values of the MLM algorithm.

From an energy point-of-view, normalization should not happen too frequently, since extra clock cycles are spent during the normalization phase of the MLM algorithm on the MONTIUM. Furthermore, normalization causes data lines to change their value extensively. Hence, in the 2-complement system, the slightest fluctuation around zero causes the binary representation to jump from all *zeros* to all *ones*.

Throughput The Turbo codes used in the UMTS communication system have *constraint length* $k = 4$, which means that 8 states exist in the trellis of the Turbo code. Hence, 8 forward recursion operations and 8 backward recursion operations have to be performed consecutively for each time instant of the trellis (within each *stage* in Figure 6.2). The existence of multiple parallel ALUs and memories in the MONTIUM provides resources to calculate the forward and backward recursion in 4 clock cycles for one time instant (or *stage*) of the trellis.

Immediately after the calculation of the backward state metrics, the LLR is calculated. The LLR calculation in the MONTIUM is performed in 4 clock cycles per time instant (*stage*) of the trellis. Consequently, 8 clock cycles are required to apply the MLM algorithm for one time instant (*stage*) of the trellis.

Due to the limited size of the local memories in the MONTIUM, the maximum block sizes that can be used for Turbo decoding is limited. However, the limited memory in the MONTIUM is not problematic when using the already mentioned *sliding window* approach [29].

The maximum channel data rate of the UMTS communication system is 1.92 $Mbps$, which means that the maximum decoded data rate after Turbo decoding is 640 $kbps$, since the *rate* is $R = 1/3$. Hence, 640 $kbps$ of data needs to be decoded by the Turbo decoder. To perform Turbo decoding with 10 iterations, the MONTIUM needs to run at a clock speed of about 100 MHz[3].

[3] The MLM algorithm is applied twice during one Turbo decoding iteration, i.e. for the inner and the outer decoder. This means that totally $8 \times 20 \times 640 \cdot 10^3 = 102.4 \cdot 10^6$ clock cycles per second are needed by the MONTIUM to perform 10 Turbo decoding iterations. Notice that we have not considered the interleaving process between the inner and the outer decoder.

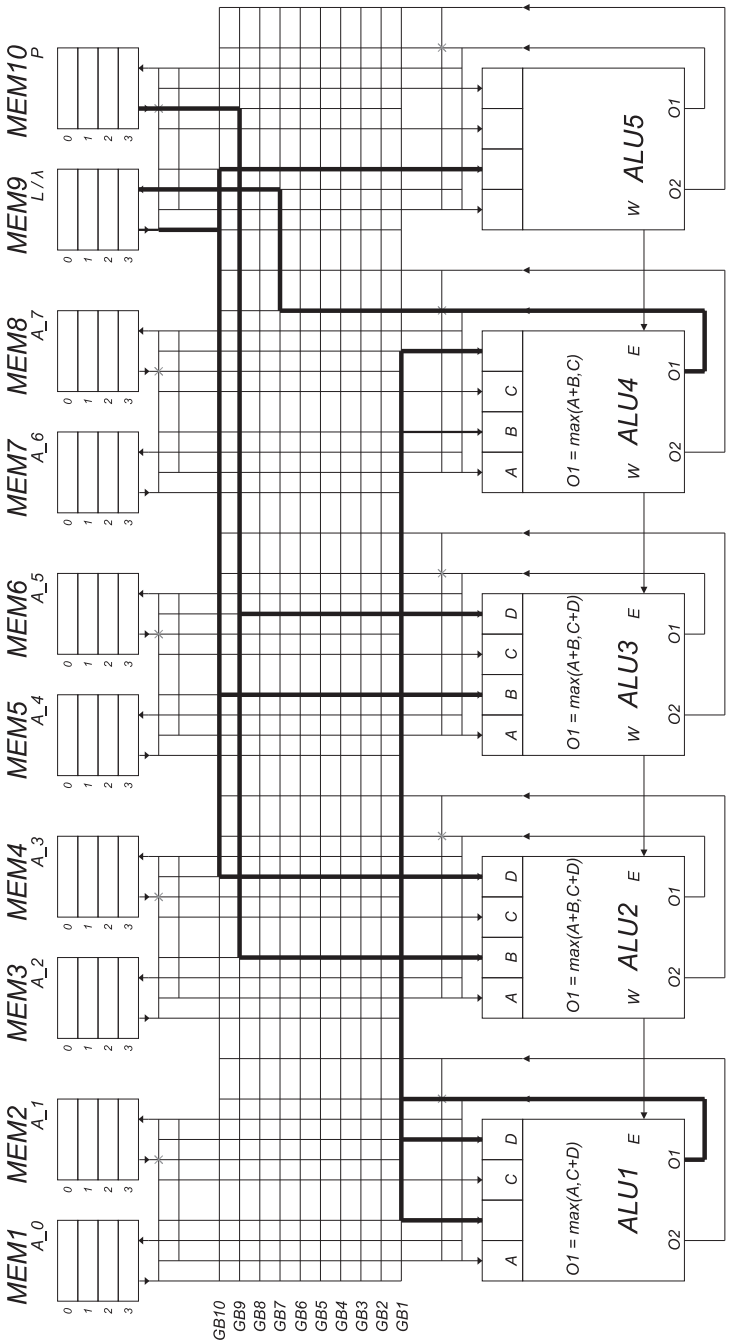

Figure 6.17: *First operation cycle of backward recursion of the MLM algorithm mapped on the* MONTIUM.

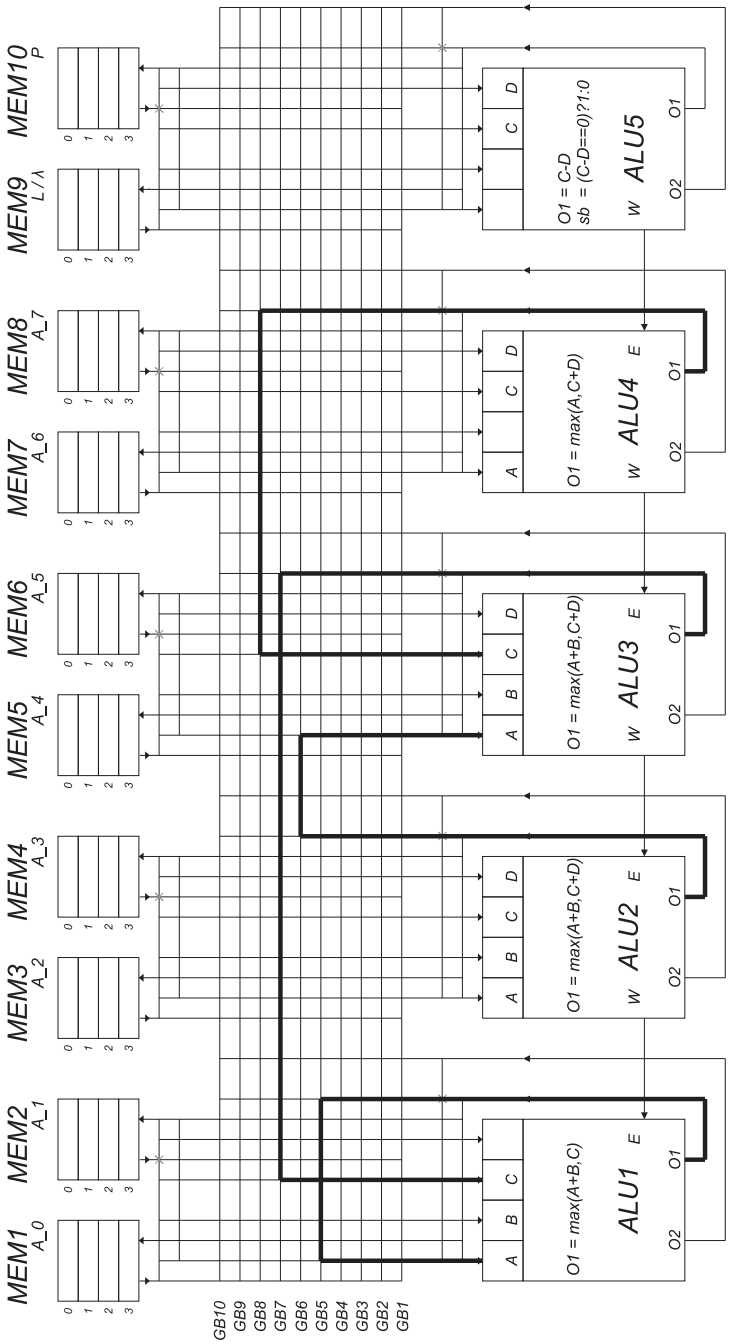

Figure 6.18: *Second operation cycle of backward recursion of the MLM algorithm mapped on the* MONTIUM.

Configuration The total configuration size of the MONTIUM MLM implementation is 1 262 bytes. This configuration is stored in the configuration memory of the MONTIUM TP in $6.36\ \mu s$ when the configuration clock frequency is 100 MHz. Mapping the MLM algorithm on the MONTIUM TP showed that especially the register allocation and global bus allocation need considerable attention. The mapping of the MLM algorithm fits in the configuration space of the MONTIUM, however, all instructions in the decoders of the register fabric and the decoders of the buses are fully used.

6.4.5 Verification

The MLM algorithm mapped on the MONTIUM has been verified in the context of a complete Turbo decoder. The performance of the mapped MLM algorithm on the MONTIUM is compared with a Log-MAP decoder and a MLM decoder using floating-point arithmetic. From [95] is known that the MLM algorithm achieves near-optimal BER performance. It is expected that the Log-MAP decoder performs best in terms of Bit Error Rate (BER).

The verification process is based on a framework for simulation that was developed in the context of research on Turbo codes. This simulation framework developed by Wu [120] was extended and integrated in the hardware / software co-simulation framework of Figure 4.17. The extended and integrated simulation framework provides decoding functionality for three different Turbo decoders: *Log-MAP decoder*, *floating-point MLM decoder* and MONTIUM-*based MLM decoder*. Because of extremely long simulation cycles for the MONTIUM-based MLM decoder, the MLM decoder has been modelled in MATLAB with the appropriate quantization and saturation properties of the MONTIUM[4].

We have determined the performance of the implemented MLM algorithm on the MONTIUM TP in terms of BER versus Signal-to-Noise Ratio (SNR). Using the simulation framework we have simulated frames of 1 000 decoded information bits. The information bits were Turbo encoded with the Turbo code that is specified in the UMTS standard [1]. Decoding of the encoded data, which was distorted with Additive White Gaussian Noise (AWGN), was done until all three Turbo decoder implementations (i.e. *Log-MAP*, *floating-point MLM* and MONTIUM-*based MLM*) had decoded the bit sequence with a maximum of 10 iterations. Frames were continuously decoded for a certain SNR point until a total of 25 frame errors had occurred with the MONTIUM-based Turbo decoder implementation. This simulation process was repeated 5 times for every SNR setting.

The resulting Frame Error Rate (FER) of the Turbo decoder implementations has been depicted in Figure 6.19. The FER performance of the Turbo decoders has been included to justify the number of decoded bits that have been simulated to generate the graphs of

[4] The MONTIUM-based MLM decoder has been modelled in MATLAB. The MATLAB model was verified against MONTIUM hardware simulations for extreme cases. For this verification both typical Turbo code values and extreme random values were used. The simulation results for both the MATLAB simulation and the MONTIUM hardware simulation were *exactly* the same under these extreme conditions. Hence, we may conclude that the MATLAB model of the MONTIUM-based MLM algorithm can be used to measure the BER performance of the MONTIUM implementation.

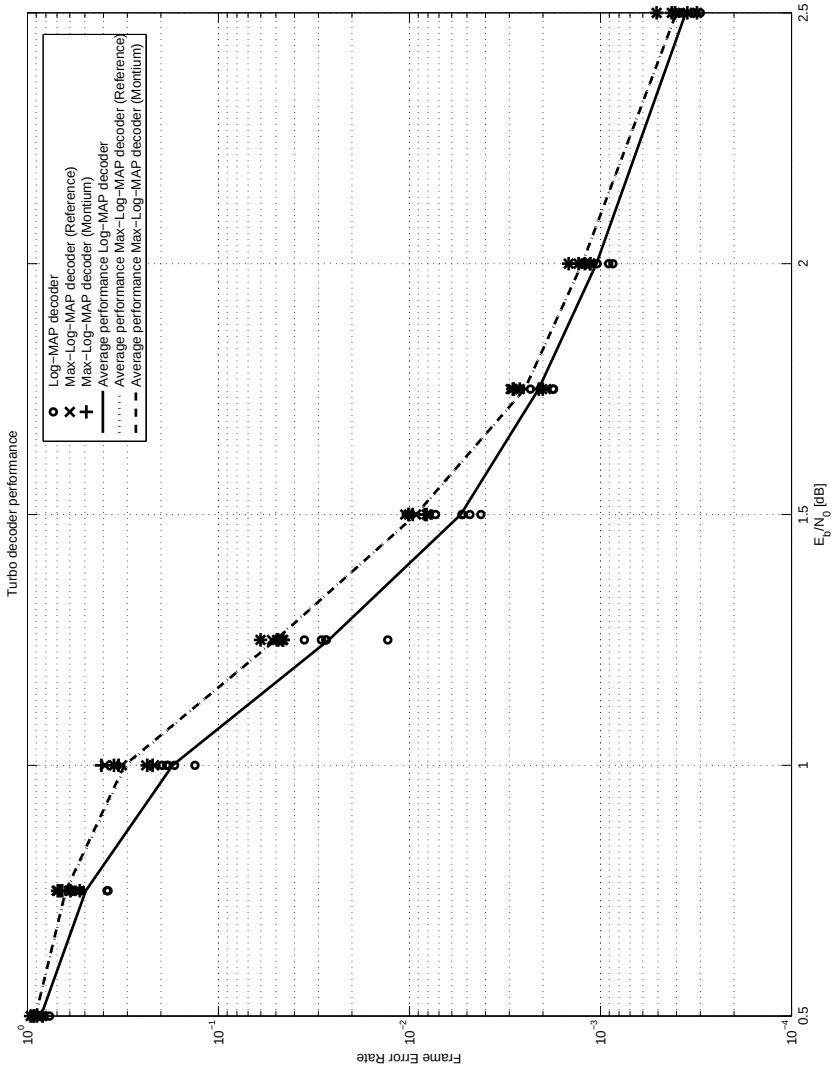

Figure 6.19: *The FER performance of the different Turbo decoders:* Log-MAP decoder, floating-point MLM decoder *and* MONTIUM-based MLM decoder.

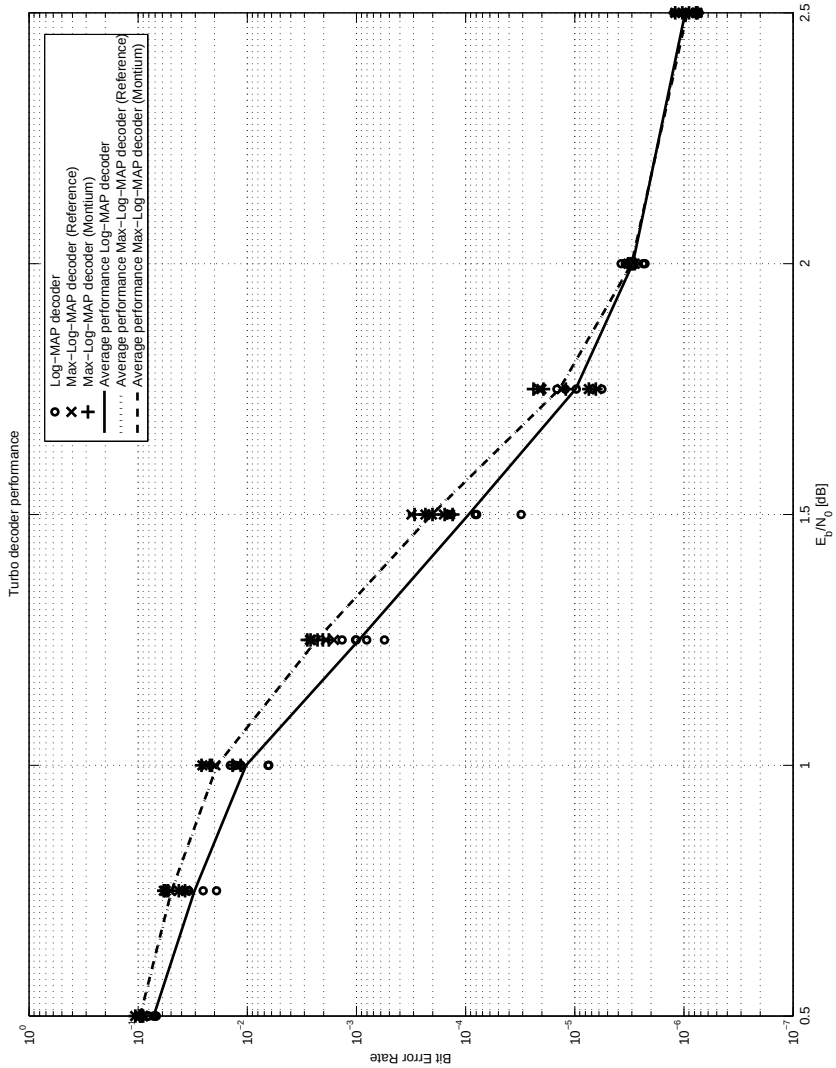

Figure 6.20: *The BER performance of the different Turbo decoders:* Log-MAP decoder, floating-point MLM decoder *and* MONTIUM-based MLM decoder.

Figure 6.20. Every simulated frame contains $1\,000$ decoded bits, and simulations for a certain SNR setting are done until 25 frames are received with errors. The total number of simulated frames can be derived from the FER as given in Figure 6.19. Hence, the justification of the total number of simulated bits turns out directly from the FER, since the number of bits per frame is constant as well as the number of erroneous frames. For instance in case of an SNR of $2.0\ dB$ the FER equals 10^{-3} and since 25 frames are decoded erroneously, the total number of simulated frame adds up to $\sim 25 \cdot 10^3$. Accordingly, $\sim 25 \cdot 10^6$ bits have been decoded during simulation of one SNR point to generate the FER and BER results that are depicted in Figure 6.19 and 6.20.

Figure 6.20 depicts the BER results of the simulations, as described above. All simulation results of the 5 independent simulation times are depicted in Figure 6.20 as separate simulation points. The performance of all three different Turbo decoder implementations, i.e. based on *Log-MAP*, *floating-point MLM* and MONTIUM-*based MLM*, is given in the BER versus SNR plot. The average BER versus SNR performance of 5 simulation iterations is also given in Figure 6.20.

The Log-MAP decoder has been depicted to illustrate the optimum BER performance. As expected from [95], the Log-MAP decoder shows the best BER versus SNR simulation results. The MONTIUM implementation of the MLM decoder shows equal performance as the floating-point MLM decoder. Furthermore, the results show that the decoding performance of the Log-MAP and MLM decoder converge to equal value in case of large SNR.

For better comparison of the decoding capabilities of both the floating-point MLM algorithm and the MONTIUM-based MLM algorithm, Figure 6.21 compares the BER after Turbo decoding using the floating-point MLM algorithm (on the *y-axis*) against the BER after Turbo decoding using the the MONTIUM-based MLM algorithm (on the *x-axis*). On average both implementations show similar decoding performance.

6.5 De-interleaving and de-puncturing

Channel coding comes normally together with interleaving and puncturing[5]. In Chapter 4 and 5 the basic building blocks of common Orthogonal Frequency Division Multiplexing (OFDM) and Wideband CDMA (WCDMA) communication systems were already identified. Most OFDM standards apply block interleavers, all with different sizes. The size of the interleavers can even vary within one standard, since different modes in OFDM standards have been defined for different channel characteristics (e.g. interleaving is applied on 1 or 5 OFDM symbols in the Digital Radio Mondiale (DRM) standard).

Interleaving in communication systems denotes a way to arrange data in a particular way to protect against burst errors and to improve the performance of data transmission. Interleaving can be performed either on bits or on QAM symbols (i.e. sub-carriers). De-interleaving is not a computationally intensive process, but rather a memory intensive

[5] De-interleaving and de-puncturing are considered in this section because these are important operations involved in channel decoding.

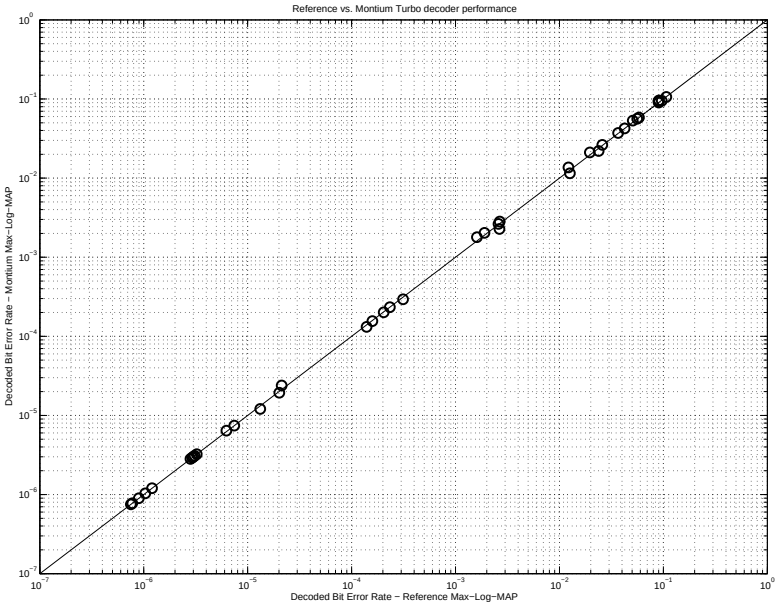

Figure 6.21: *The BER performance of the* MONTIUM*-based MLM Turbo decoder versus the floating-point MLM Turbo decoder.*

process with regular semi-random memory access patterns. The interleaving patterns have been investigated for HiperLAN/2, DAB and DRM in Chapter 4.

In the HiperLAN/2 [36] communication system all de-interleaving operations are performed on bits. All encoded bits in HiperLAN/2 are interleaved by a block interleaver, which has a block size equal to the number of bits in a single OFDM symbol.

In the DAB [38] receiver frequency de-interleaving is applied on the sub-carriers of one OFDM symbol. Hence, the block size of the DAB frequency de-interleaver is equal to the OFDM symbol size. Frequency de-interleaving is performed on every OFDM symbol. Interleaving on sub-carriers is applied to ensure that adjacent QPSK symbols are not mapped to adjacent sub-carriers and to combat frequency-selective fading effects. Furthermore, time de-interleaving is applied in the DAB receiver to combat burst errors. Time interleaving is applied in DAB on all encoded bits of the Main Service Channel (MSC). The time interleaving rules in DAB are defined according to Table 4.13. Time interleaving and, therefore, also de-interleaving are applied over 16 logical frames with each frame containing $55\,296$ coded bits.

In the DRM [37] receiver also two kinds of de-interleavers have been identified. The time de-interleaver in DRM is applied on QAM symbols containing MSC information. The QAM symbols are shuffled in time with the option of short or long interleaving, de-

pending on the respective channel conditions. For short interleaving, the symbols are shuffled within one DRM transmission frame. Long interleaving is applied on QAM symbols in five DRM transmission frames. The duration of one DRM transmission frame is $400\ ms$[6]. Since different transmission modes with diverse numbers of sub-carriers are defined in DRM, the number of QAM symbols per multiplex frame is variable. Forward Error Correction (FEC) coding has been applied in the DRM system on all information in the DRM transmission super frame[7] by means of multi-level coding (MLC). The individual bits in every level of the QAM symbols are convolutionally encoded, punctured, if necessary, and interleaved. The interleaving in MLC is performed on the individual bits of the FAC, SDC or MSC. The block size of the interleavers[8], specified in number of bits, is maximally 130, $1\,692$ and $12\,236$ for the FAC, SDC and MSC, respectively.

Interleaving in UMTS [1] is incorporated at different levels of the communication system. The Turbo encoder possesses the characteristic of an interleaver step that is applied in between the concatenated Recursive Systematic Convolutional (RSC) encoders. The Turbo decoder accordingly applies a de-interleaver in between the outer and inner decoders. The block size of the interleaver in the Turbo encoder / decoder ranges between 40 and 5114 bits for the UMTS standard. The UMTS communication system generally has two levels of interleavers defined, regardless of the coding applied (e.g. Turbo or convolutional). The interleavers in both levels of the system are block interleavers with both different inter-column permutations. This *first* and *second* interleaving process is done on all bits of the radio frames in one UMTS Transmission Time Interval (TTI)[9].

In all communication systems *rate matching* is normally applied in the transmitter by means of bit puncturing. Bit puncturing denotes the removal of encoded bits from the encoded bit stream. Therefore, de-puncturing has to be applied in the receiver. The de-puncturing process inserts meaningless data at the positions of the punctured bits in the received encoded bit stream. After de-puncturing the received bit stream, the channel decoding algorithms are performed. The de-puncturing schemes can optionally be combined with the de-interleaving process in the communication receiver. Disadvantage of combining the de-interleaving and de-puncturing process is that the regular memory access patterns of the interleaver may be destroyed.

Most communication systems require large block interleavers, which can be implemented as memory operations. Small interleavers, like the interleaver that is applied in HiperLAN/2, can be implemented efficiently together with the baseband processing of OFDM symbols. However, when large amounts of data have to be processed, large memories are required to store all bits of an entire frame. Therefore, interleavers cannot effectively be integrated in the streaming baseband processing, which is performed on e.g. an OFDM symbol basis, but interleavers need to be implemented with special memory structures (e.g. special tiles in the System-on-Chip (SoC) which have large memory

[6] One DRM transmission frame contains 15, 20 or 24 OFDM symbols for robustness mode A and B, C or D, respectively.

[7] One DRM transmission super frame consists of three DRM transmission frames.

[8] Multiple interleavers are applied in the MLC process for all different levels of the QAM symbols.

[9] The UMTS TTI is either 10, 20, 30 or $40\ ms$.

capacity and generate proper memory access patterns.). Ideas on the design of these special memory tiles are given in Chapter 7.

6.6 Adaptive channel decoder

The Turbo and convolutional code schemes have been adopted by many wireless communication standards. In the 3G UMTS system both coding schemes are employed; turbo coding has been used for data channels and convolutional coding for voice channels [1].

In [70] it has been reported that 50% or more of the computational complexity in the wireless receiver is due to error coding algorithms, like Viterbi or Turbo decoding. This complexity counts for about 60% of the energy consumption in the digital processing part of a typical wireless receiver [19]. Consequently, power reduction should be achieved in the error decoding part of the receiver. New hardware oriented design methodologies are proposed in ASIC design for Viterbi [45] and Turbo decoders [69], such as full-speed parallel solutions and architecture structures with shared resources. Power reduction by exploiting variations in system characteristics due to changing noise conditions are the focus in [53, 65]. The latter can be achieved using dynamic reconfiguration in reconfigurable hardware.

In [18] a channel decoder chip was proposed that is compliant with the 3G wireless communication standard. However, this chip is a dedicated solution for the 3G UMTS system, which cannot be used in other wireless communication standards. We implemented both the Turbo and the Viterbi decoder in the coarse-grained reconfigurable MONTIUM architecture, which can also be used to implement the baseband processing of different wireless communication standards (Chapter 4 and 5). The flexible coarse-grained reconfigurable MONTIUM enables the implementation of flexible baseband processing and flexible channel decoding in multi-mode communication systems.

The unified channel decoder chip in [18] has been implemented in 0.18 μm CMOS technology, therefore, it cannot fairly be compared with the MONTIUM architecture, which has been implemented in 0.13 μm CMOS technology.

In [21] another reconfigurable architecture for Viterbi and Turbo decoding was reported. That architecture, *Viturbo*, can be configured to decode convolutionally encoded data and Turbo encoded data. The *Viturbo* decoder is only aimed at channel decoding, whereas the MONTIUM architecture is more flexible, and suitable for baseband processing as well. The area of the MONTIUM is slightly larger than the *Viturbo* decoder (~ 250 kGates vs. ~ 200 kGates).

A multi-standard embedded processor for Viterbi decoding has been presented in [88]. The Viterbi processor targets multiple audio and video broadcasting as well as data communication standards. The flexible Viterbi processor is a specific instantiation of the general AVISPA architecture template [67]. Finally, a parameterized multi-standard ASIC has been created from the specific AVISPA instantiation in 0.13 μm CMOS technology. The area optimized Viterbi processor counts 130 kGates and has an area size of 1.2 mm^2.

Trends in wireless communication systems show that the underlying Digital Signal Processing (DSP) algorithms become more adaptive. Three levels of adaptivity can be identified for the DSP functions of wireless communication receivers: *standards level*, *algorithm-selection level* and *algorithm-parameter level*. The levels of adaptivity are introduced in Chapter 2. The implemented Viterbi and Turbo decoder can be used to make an adaptive channel decoder. The three levels of adaptivity also apply for the adaptive channel decoder:

1. *Standards level*: Different channel decoders are used for different wireless communication standards. In the DAB standard and UMTS standard convolutional coding is employed. But in the UMTS standard Turbo coding is also applied. Consequently, the usage of the Viterbi or Turbo decoder depends on the requested communication standard;

2. *Algorithm-selection level*: Within the UMTS communication system both convolutional and Turbo coding schemes are defined. For data channels the Turbo coding schemes are used and for voice channels convolutional coding schemes are applied. This means that in a UMTS mobile terminal both the Viterbi and the Turbo decoder need to be available. Another kind of algorithm-selection adaptivity is the usage of different algorithms for the SISO decoders in the Turbo decoder. The kind of algorithm that will be used, depends heavily on the conditions of the wireless environment (e.g. [53, 65]);

3. *Algorithm-parameter level*: The characteristics of the decoders can be varied by changing the implementation parameters. The mapped Viterbi decoder on the MONTIUM is universal in the sense that different *rates*, *constraint lengths* and *decision depths* can be handled. The Turbo decoder can also handle different *rates* and *constraint lengths*. Moreover, the number of Turbo *iterations* can be adjusted. The parameters *rate* and *constraint length* are dictated by the different wireless communication standards. The *decision depth* and number of Turbo *iterations* are, however, depending on the quality of the wireless environment, in which the communication system is used. Reducing the decision depth and the number of iterations yields reduction in energy consumption of the channel decoder (e.g. Table 6.1 and [53, 65]).

An adaptive channel decoder has to be utilized because of multiple standards that are implemented in mobile terminals. Hardware costs in the mobile terminal are reduced by reusing reconfigurable hardware to implement multiple decoding algorithms. An adaptive channel decoder also yields large energy savings because the decoding algorithm that performs best in a given environment with minimal effort can be applied by reconfiguring the hardware to the desired functionality.

6.7 Summary

This chapter covers channel decoding algorithms that are applied in different wireless communication systems. Two different channel decoders are highlighted in this chapter: the *Viterbi decoder* and the *Turbo decoder*. Requirements for implementing the Digital Signal Processing (DSP) algorithms on reconfigurable hardware have been investigated. Multi-standard wireless communication receivers are able to support different channel decoding strategies.

6.7.1 Conclusions

The Viterbi decoder has been mapped efficiently on the MONTIUM TP. The Viterbi decoder is used to decode convolutionally encoded information. The Viterbi algorithm comprises *min()* operations and regular memory access patterns. Large memory bandwidth is required during Viterbi decoding, since the state metrics as well as the survivor bit sequences have to be administered.

When implementing the Viterbi algorithm on coarse-grained reconfigurable hardware, i.e. the MONTIUM TP, the Register Exchange (RE) approach is an appropriate method to store all survivor bit sequences. The RE approach stores the entire decoded output sequence for each state during path metric updating. Therefore, in each stage of the trellis the decoded output sequence is known and there is no need to traceback. So, the decoded output sequence can be generated at a high speed. Moreover, bit sequences (Register Exchange) are stored more efficiently than individual bits (Traceback) in the coarse-grained MONTIUM. Since the data path in the MONTIUM TP is limited to 16-bits, the Viterbi algorithm has been implemented in a hybrid Register Exchange / Traceback manner.

All operations in the Viterbi algorithm are performed on generalized Viterbi butterfly structures. The *Add Compare Select (ACS)* operation is applied to every Viterbi butterfly structure. The ACS operation comprises basically a conditional multiplexer (*select* operation), which selects one operand based on the result of a *compare* operation.

The problem, which arises from the ACS operation, is that the operation merges the data path and the control path. Control-oriented functions are actually not part of the MONTIUM algorithm domain. However, to efficiently implement the Viterbi algorithm, the ACS operation has to be supported by the hardware on which the algorithm is mapped. Because of the high degree of parallelism in the Viterbi algorithm, parallel ACS support is required. Adding the ACS operation to the ALUs of the MONTIUM enables the *compare-select* operation to run independently from the sequencer instructions. Consequently, the *compare-select* operation can run in parallel in all 5 ALUs of the MONTIUM.

The performance of the Viterbi algorithm, implemented on the MONTIUM, has been verified against a reference floating-point implementation. The MONTIUM-based Viterbi algorithm yields on average equal performance in terms of Bit Error Rate (BER) compared with the floating-point reference implementation.

The MONTIUM-based Viterbi decoder is compared with a dedicated decoder implemented in an Application Specific Integrated Circuit (ASIC) and an alternative implemen-

tation in an embedded ARM processor. We conclude that the MONTIUM-based Viterbi decoder yields similar performance in terms of decoding throughput as an ASIC solution. The ARM processor cannot offer the required throughput when performing the Viterbi algorithm. We noticed that the ASIC implementation of the Viterbi decoder is most energy efficient, but the implementation does not have any flexibility. On the other hand, flexibility comes with a penalty of increased energy consumption per decoded bit. Although the MONTIUM is a flexible reconfigurable architecture, the energy consumption of the MONTIUM TP is closer to that of an ASIC implementation than to a GPP solution. The ARM-based Viterbi decoder performs worse in terms of energy consumption as well as in terms of decoding throughput.

The Turbo decoder contains soft-output decoders, like the Max-Log-MAP (MLM) algorithm. The MLM algorithm requires more calculations than the Viterbi algorithm, since a *forward* and *backward* recursion through the entire trellis have to be performed. The basic operations in the MLM algorithm are similar in nature as the Viterbi operations. The operations are again performed in a regular structure. Similar with Viterbi decoding, many state metrics have to be stored in the MLM algorithm. The performance of the MLM algorithm has been verified in the Turbo decoder context. The performance of the MLM algorithm mapped on the MONTIUM TP is similar in terms of BER versus SNR to a floating-point reference implementation of the MLM algorithm.

The configuration overhead of the Viterbi or Turbo decoder is relatively small, as the configuration files of both decoders are small. Hence, changing the functionality of the channel decoder from Turbo to Viterbi, or vice versa, can be done dynamically, because of the short reconfiguration times. The reconfiguration time of the Viterbi or Turbo decoder implementation on the MONTIUM TP is less than 7 μs. Depending on the desired communication standard, one can configure the reconfigurable hardware in the mobile wireless multi-standard receiver to implement the right channel decoder.

In this chapter the following adaptivity features were identified for an adaptive channel decoder implemented with reconfigurable hardware:

- *Standards level*
  Different channel decoders are used for different wireless communication standards. In the DAB standard and UMTS standard convolutional coding is employed. But in the UMTS standard Turbo coding is also applied. Consequently, the usage of the Viterbi or Turbo decoder depends on the requested communication standard. The MONTIUM TP can be reconfigured for Viterbi or Turbo decoding in less than 7 μs;

- *Algorithm-selection level*
  Within the UMTS communication system both convolutional and Turbo coding schemes are defined. This means that in a UMTS mobile terminal both the Viterbi and the Turbo decoder need to be available. Switching from Viterbi to Turbo decoder, or vice versa, is achieved through reconfiguring the MONTIUM TP in less than 7 μs;

- *Algorithm-parameter level*
  The characteristics of the decoders can be varied by changing the implementation parameters. The mapped Viterbi decoder on the MONTIUM is universal in the sense that different *rates*, *constraint lengths* and *decision depths* can be handled. The Turbo decoder can also handle different *rates* and *constraint lengths*. Moreover, the number of Turbo *iterations* can be adjusted.

6.7.2 Discussion

Channel decoding is normally accompanied by de-interleaving and de-puncturing. We investigated the applied de-interleaving and de-puncturing strategies in different wireless communication standards. The investigation is done on the OFDM-based communication standards in Chapter 4 and the WCDMA-based communication standard in Chapter 5.

De-interleaving and de-puncturing is usually applied on large sequences of bits or symbols. De-interleaving is not a computationally intensive process, but rather a memory intensive process with regular memory access patterns. De-puncturing schemes are generally applied on the same large sequences of bits. The de-puncturing schemes can optionally be combined with the de-interleaving process in the wireless communication receivers.

We concluded that de-interleaving and de-puncturing cannot efficiently be integrated in the coarse-grained reconfigurable hardware that is responsible for baseband processing. De-interleavers need to be implemented with special memory structures in heterogeneous reconfigurable System-on-Chips (SoCs). These memory structures are special tiles in the SoC, which have large memory capacity and generate autonomously proper memory access patterns. Advantage of these special memory tiles is that the de-interleavers are integrated closely with the coarse-grained reconfigurable processing tiles in the SoC, preserving the locality of reference principle. Moreover, special memory tiles with functions for address generation are expected to be more efficient than General Purpose Processors (GPPs). In Chapter 7, initial design ideas of these special memory tiles are further discussed.

Adaptivity in wireless communication receivers is utilized to improve the efficiency of algorithms in different conditions and to gain performance. In literature [53, 65], channel decoding algorithms have been identified as a suitable candidate to efficiently utilize adaptivity. Different adaptive approaches of channel decoding have been proposed which all require flexible hardware implementations. Flexible hardware implementations are proposed which are, however, only suitable to perform different channel decoding functions. The MONTIUM architecture, on the other hand, is flexible and capable of performing channel decoding functions and baseband processing functions. The MONTIUM TP is only slightly larger in area compared to several proposed flexible channel decoders.

Chapter 7

System-on-Chip Architecture for Software Defined Radio Receivers

Scenarios for implementing multi-standard multi-mode adaptive wireless communication receivers on System-on-Chip (SoC) architectures are investigated in this chapter.

The CHAMELEON *SoC template has inspired the design of a prototype SoC architecture. The prototype* ANNABELLE *SoC has been developed for the implementation of digital radio broadcasting standards. The* ANNABELLE *SoC contains four* MONTIUM *Tile Processors (TPs) to perform Digital Signal Processing (DSP) algorithms.*

Since channel decoding in wireless communication systems involves interleaving mechanisms, a special memory tile that is integrated in future SoC architectures is proposed. The memory tile has to be capable of performing different interleaving strategies.

Parts of this chapter have been published in [P18].

7.1 Introduction

In the previous chapters we illustrated our approach of implementing adaptive multi-standard wireless communication receivers in heterogeneous reconfigurable System-on-Chip (SoC) architectures. The template of the heterogeneous SoC, as described in Chapter 3 and shown in Figure 3.3, is applied to illustrate the mapping of a multi-standard wireless communication receiver, comprising the baseband processing and channel decoding of Orthogonal Frequency Division Multiplexing (OFDM) and Wideband CDMA (WCDMA) wireless communication standards. The results obtained from Chapter 4 and 5 concerning baseband processing and from Chapter 6 with respect to channel decoding resulted in the design of a prototype SoC architecture.

The chip is intended to be used for multi-standard digital radio broadcasting receivers (e.g. DRM and DAB) and contains four coarse-grained MONTIUM processing tiles, which is sufficient for these digital radio applications. The prototype SoC architecture, called ANNABELLE, is described in Section 7.2.

Investigations on the channel decoding showed that in all wireless communication standards de-interleaving and de-puncturing operations are defined. In Chapter 6 we already concluded that special memory structures need to be incorporated in the SoC architecture to support multi-standard wireless communication standards. In Section 7.3 the requirements and initial design ideas are discussed for such a memory architecture.

Moreover, a control-manager is needed for controlling all receiver tasks in the heterogeneous reconfigurable SoC. The control-manager that is used for implementing an adaptive Universal Mobile Telecommunications System (UMTS) / Wireless LAN (WLAN) receiver is discussed in Section 7.4.

Section 7.5 summarizes this chapter with conclusions and discussions on future SoC architectures.

7.2 ANNABELLE System-on-Chip

The CHAMELEON System-on-Chip (SoC) template accommodates the initial ideas of a heterogeneous reconfigurable tiled SoC. The CHAMELEON SoC template has been proposed in [54]. Motivations for using this tiled SoC approach are discussed in Chapter 3.

In this section the prototype ANNABELLE SoC is described according to the CHAMELEON SoC template, which is intended to be used for digital radio broadcasting receivers (e.g. DAB, DRM)[1].

7.2.1 Architecture

Figure 7.1 shows the overall architecture of the ANNABELLE SoC. The ANNABELLE SoC consists of an ARM926 General Purpose Processor (GPP) with a 5-layer AMBA Advanced

[1] The ANNABELLE SoC has been developed in the context of the SMART CHIPS FOR SMART SURROUNDINGS (4S) research project (FP6-IST-2004-001908) [6].

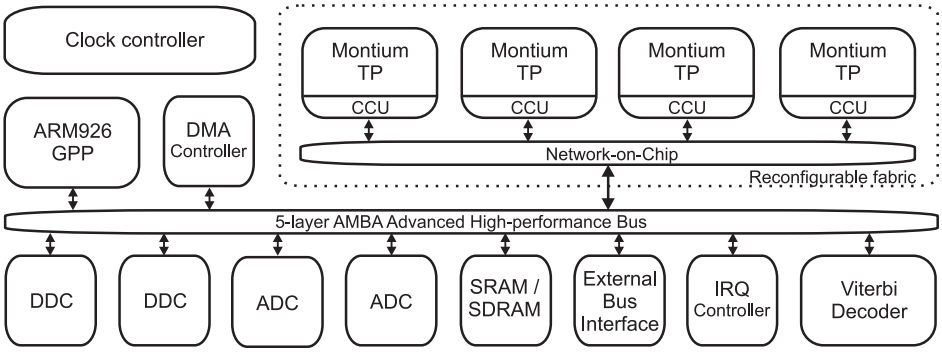

Figure 7.1: *Block diagram of the* ANNABELLE *System-on-Chip.*

High-performance Bus (AHB), four MONTIUM Tile Processors (TPs), a Network-on-Chip (NoC), a Viterbi decoder, two Analog-to-Digital Converters (ADCs), two Digital Down Converters (DDCs), a Direct Memory Access (DMA) controller, SRAM / SDRAM memory interfaces and external bus interfaces.

The four MONTIUM TPs and the NoC are arranged in a reconfigurable subsystem, labelled *reconfigurable fabric*. The reconfigurable fabric is connected to the AHB bus and serves as a slave to the Advanced Microcontroller Bus Architecture (AMBA) system. A configurable clock controller generates the clocks for the individual MONTIUM TPs. Every individual MONTIUM TP has its own adjustable clock and runs at its own speed.

A prototype chip of the ANNABELLE SoC[2] has been developed using the Atmel ATC13 library [11].

7.2.2 Reconfigurable fabric

The reconfigurable subsystem of the ANNABELLE SoC inherits the initial design concepts of the CHAMELEON SoC template. The reconfigurable fabric that is integrated in the ANNABELLE SoC is shown in more detail in Figure 7.2.

The reconfigurable fabric is integrated as an AHB slave device in the ANNABELLE SoC. In fact, the reconfigurable fabric acts as a reconfigurable co-processor for the ARM926 processor. Computationally intensive Digital Signal Processing (DSP) algorithms are typically offloaded from the ARM926 processor and processed on the coarse-grained reconfigurable MONTIUM TPs inside the reconfigurable fabric.

The reconfigurable fabric contains four MONTIUM TPs which are connected via a HYDRA Communication and Configuration Unit (CCU) to a circuit-switched NoC. The reconfigurable fabric is interconnected with the AMBA system through a AHB-NoC bridge interface.

The MONTIUM TPs are individually reconfigurable at run-time. The reconfigurable fabric provides *block mode* and *streaming mode* computation services.

[2] The work described in this thesis focuses on the reconfigurable fabric only.

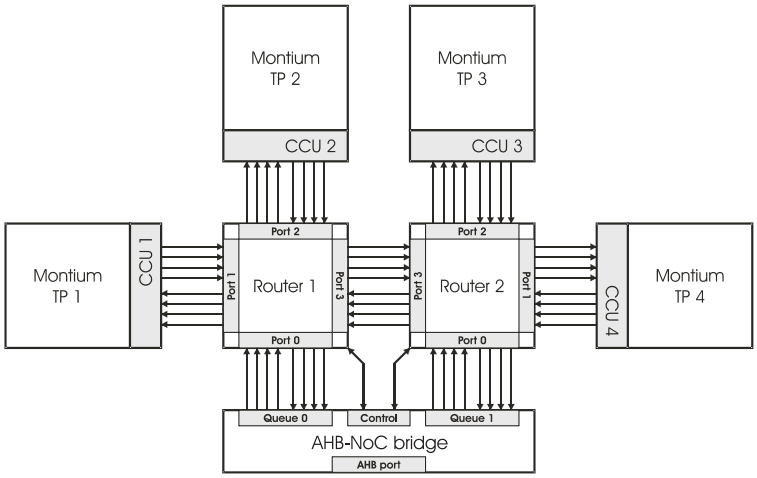

Figure 7.2: *The* ANNABELLE *SoC reconfigurable fabric.*

AHB-NoC bridge

The reconfigurable fabric is connected to the AMBA system through the AHB-NoC bridge. The AHB-NoC bridge provides *configuration* and *communication* services for the reconfigurable fabric:

- *Configuration*
  The individual routers of the circuit-switched NoC are configured via the AHB-NoC bridge. Thus, via the AHB-NoC bridge the communication channels between a MONTIUM TP and AHB or between two MONTIUM TPs are configured; all point-to-point links in the NoC (e.g. between MONTIUM TP1 and Router1) contain four lanes. Hence, eight communication channels (four channels from the AHB bus to the reconfigurable fabric, and four reverse channels) can be set-up through Router 1 and eight through Router 2 (analogous to Router 1);

- *Communication*
  When the communication channels in the reconfigurable fabric are set up correctly, data can be read from / written in the respective lanes of the NoC by the AHB-NoC bridge.

  The data that is written in the channel is received e.g. by a MONTIUM TP. The data is handled by the HYDRA CCU of the corresponding MONTIUM TP. The data written to the MONTIUM TP consists of either:

  – MONTIUM TP configuration data;
  – HYDRA CCU commands or;
  – Data that has to be processed by the MONTIUM TP.

Figure 7.2 shows that the *control* port of the AHB-NoC bridge connects to the circuit-switched routers of the NoC. The control port provides *configuration* services of the reconfigurable fabric. The *Queues* in the AHB-NoC bridge provide the *communication* services for every individual lane in the links from the AHB-NoC bridge to the circuit-switched routers.

The *Control* ports and *Queue* ports are integrated in the memory map of the entire ANNABELLE SoC. To reduce overhead of the ARM926 processor during communication, the data transfer from the AMBA system to the AHB-NoC bridge can be performed via DMA transfers. Therefore, the AHB-NoC bridge is implemented with a DMA interface.

Network-on-Chip

The MONTIUM TPs and the AHB-NoC bridge are connected to a fully flexible circuit-switched NoC [118]. The reconfigurable circuit-switched NoC creates dedicated connections between processing tiles in the reconfigurable fabric. The circuit-switched NoC provides channels with guaranteed throughput, but with minimal amount of control in the data path. Unlike packet-switched NoCs [64], no arbitration is needed in the circuit-switched NoC which increases the energy efficiency per transported bit.

The channels in the circuit-switched NoC of the reconfigurable fabric are set up by reconfiguring the routers. The routers provide full connectivity in the reconfigurable fabric that can be reconfigured dynamically at run-time.

HYDRA CCU

The MONTIUM TPs are connected with the circuit-switched NoC via the HYDRA CCU [20]. The CCU provides *communication* and *configuration* services to the MONTIUM TP, as described in Section 3.5.1 on page 29.

MONTIUM Tile Processor

The reconfigurable fabric contains four coarse-grained reconfigurable MONTIUM TPs. The architecture of the MONTIUM TP is described in Section 3.5.1. The designed reconfigurable fabric incorporates the MONTIUM TP with Add Compare Select (ACS) support.

Since the ANNABELLE SoC is intended to be used in digital broadcasting receivers, the design time parameters of the MONTIUM TP have been customized for Digital Audio Broadcasting (DAB) and Digital Radio Mondiale (DRM) systems. Hence, the local memories of the MONTIUM TP have a capacity of $1\,024$ addresses of 16-bit words each. The memory size is sufficient to efficiently implement Fast Fourier Transform (FFT) algorithms with a size of up to $2\,048$ points on the MONTIUM TP for DAB.

Dedicated output pins of the MONTIUM TPs are used to generate Interrupt Requests (IRQs). The IRQs are handled by the interrupt controller of the ARM926 processor. In this way the reconfigurable fabric and the ARM926 processor can operate autonomously. Interaction between the concurrent process running on the reconfigurable fabric and the ARM926 is provided by IRQ signalling.

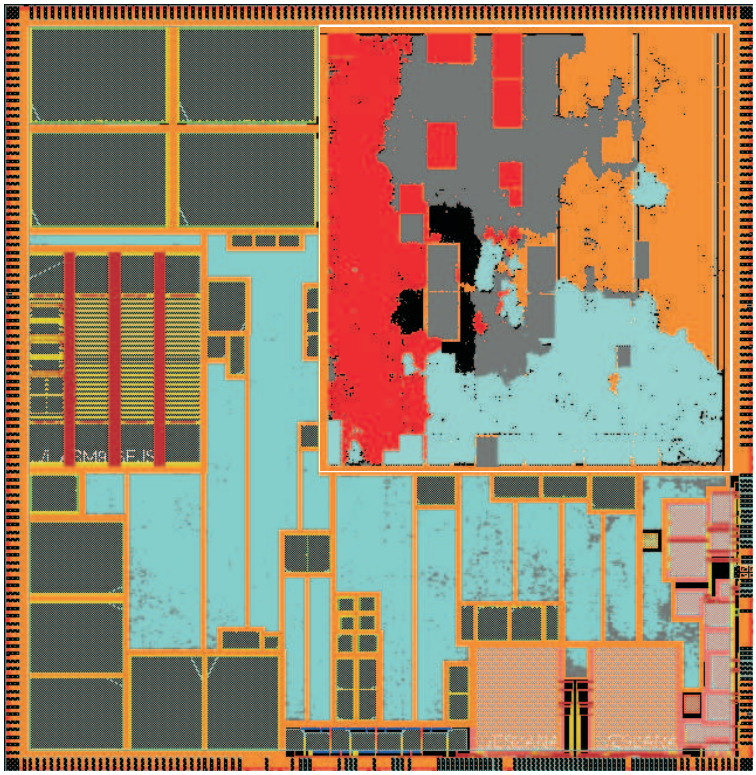

Figure 7.3: *Layout of the* ANNABELLE *System-on-Chip.*

7.2.3 ASIC synthesis of the reconfigurable fabric

The ANNABELLE SoC has been synthesized in 130 nm CMOS technology with a supply voltage of 1.2 volt. A prototype chip has been developed using the Atmel ATC13 library [11]. The ATC13 library contains a comprehensive list of combinational logic and storage cells. Furthermore, single port Static RAM (SRAM) modules have been compiled for the 130 nm CMOS technology. Figure 7.3 depicts the layout of the ANNABELLE SoC with the reconfigurable fabric in the top right corner.

The total area of the reconfigurable fabric synthesized in 130 nm CMOS technology is $\sim 15\ mm^2$. Table 7.1 depicts the area of the individual building blocks in the reconfigurable fabric without the area of SRAM cells. The area of the MONTIUM TP does not include the area of the SRAM cells, which adds another $0.87\ mm^2$. The gate[3] density of the used CMOS technology process is ~ 140 kGates / mm^2.

[3] An equivalent gate is a unit of measure to express the amount of 2-input NAND gates. One 2-input NAND gate contains typically 4 transistors.

Table 7.1: *Area of the synthesized building blocks (excluding SRAM) in the reconfigurable fabric.*

Component	Area [mm^2]
AHB-NoC bridge	0.181
Circuit-switched router	0.091
HYDRA CCU	0.104
MONTIUM TP	2.762

7.3 System-on-Chip memory tile

In Section 6.5 has been analyzed that channel decoding is strongly coupled with de-interleaving[4] and de-puncturing. Although many wireless communication standards incorporate interleaving mechanisms, the interleaving approaches are highly diverse in different wireless communication systems.

Future heterogeneous SoC platforms will definitely incorporate special memory architectures. The customized memory architectures are tightly coupled in the SoC to the processing tiles where baseband processing and channel decoding algorithms are performed (i.e. exploiting locality of reference). Special interleaving memory architectures are needed because interleaving operations are inefficiently performed on GPPs. The interleaving patterns do not fit in the local cache of a GPPs and, furthermore, caching mechanisms are not effective. Therefore, interleaving operations performed on a GPP involve large control overhead.

Moreover, the memory architecture can act as additional storage in the tiled heterogeneous reconfigurable SoC. The additional storage capacity is advantageous in case the storage capacity of the processing elements in the SoC (e.g. the MONTIUM TP) is not sufficient. In this case the memory tile provides another level of storage in the memory hierarchy of the heterogeneous SoC. Hence, the locality of reference principle is maintained since off-chip memory access is prevented by introducing hierarchy in the memory structure of the heterogeneous SoC.

Those memory architectures need to comply with special requirements for performing de-interleaving functions:

- Large storage capacity;
- Bidirectional communication, e.g. simultaneously reading / writing;
- Different communication modes:
  - Block mode;
  - Streaming mode;

[4] Interleaving and de-interleaving involve the same mechanisms. Therefore, the terminology interleaving or de-interleaving is arbitrarily used in this thesis.

- Flexible memory address pattern generation:
  - Address generator with modulo support;
- Different interleaving levels:
  - Interleaving of individual bits;
  - Interleaving of clusters of bits, e.g. bytes;
- Low configuration overhead.

Figure 7.4 depicts an initial conceptual design of a memory tile that is capable of performing de-interleaving functions in the context of a SoC. The memory tile consists of a Communication and Control Interface (CCI), an Address Generation Unit (AGU), a simple state machine and a memory unit. The solid arrows depict data streams in the memory tile, while the dashed arrows indicate internal control signals.

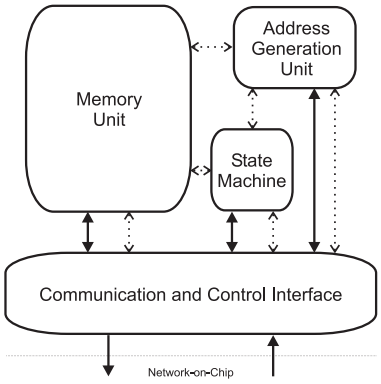

Figure 7.4: *The reconfigurable memory tile template.*

The CCI connects the memory tile to the NoC, analogous to the CCU in combination with the MONTIUM TP. The CCI provides *block* mode and *streaming* mode communication services. Moreover, the CCI is used to adjust the settings of the memory tile through reconfiguration of the state machine, the AGU and the CCI itself. The entire memory tile is controlled by a simple state machine.

The frames of data that are subject to de-interleaving are stored in the local memory of the memory tile. The memory tile has to support bidirectional data streams. Thus, the interleaved data needs to be written in the SRAM and the de-interleaved data is read out of the SRAM simultaneously. It is evident that the output data stream is a delayed de-interleaved version of the interleaved input data stream, or vice versa. The delay between the input and output streams is determined by the interleaving depth (e.g. measured in MAC frames). The capacity of the SRAM has to suffice the interleaving depth.

The AGU generates the memory access patterns for reading and writing the memory unit. The address patterns are created using complicated modulo operations; *power of 2* and *non-power of 2* modulo operations are adopted in interleaving mechanisms. Furthermore, single bits or clusters of bits need to be accessed in the memory unit. Special features are required to access single or multiple columns of bits in the memory unit of the memory tile.

7.4 Controlling tasks in the System-on-Chip

In Chapter 4 and 5 the baseband processing of two different standards has been implemented on the same reconfigurable hardware. Both communication standards serve as a basis for an adaptive Universal Mobile Telecommunications System (UMTS) / Wireless LAN (WLAN) receiver. To employ an adaptive wireless communication system which can adapt the baseband processing functions in the physical layer, a centralized *control-manager* is needed.

The control-manager is implemented on the GPP tile of the heterogeneous SoC, which is perfectly suited for control-oriented tasks. Tasks of the control-manager are:

- Configuring the processing elements in the SoC;
- Configuring the routing elements of the NoC;
- Administering the status of the entire SoC and NoC;
- Controlling the different levels of adaptivity:
  - Switching the wireless communication standard based on availability and Quality of Service (QoS) requirements;
  - Switching the algorithms, used to implement the functions within the receiver for a wireless communication standard, based on channel conditions and QoS requirements;
  - Adapting the parameters of an algorithm based on channel conditions and QoS requirements.

For the adaptive UMTS / WLAN receiver this means that the control-manager decides whether the receiver is in UMTS or in WLAN mode. Furthermore, adaptivity within one mode can be applied. Thus, the control-manager can decide whether in the UMTS mode the Rake functionality or the equalizer functionality will be applied. Moreover, the control-manager controls the parameters of a certain algorithm, e.g. the number of Rake fingers that should be used or the modulation mode in HiperLAN/2.

7.4.1 HiperLAN/2 to UMTS switching

In HiperLAN/2 mode, 4 MONTIUM TPs in the SoC are configured for *Prefix removal, Frequency offset correction, Inverse OFDM* and *Equalization, Phase offset correction and De-mapping*. Furthermore, the General Purpose Processor (GPP) in the SoC is programmed to perform the *Channel estimation* function. This task generates the estimated frequency offset and the equalization coefficients for the *Frequency offset correction* and *Equalization, Phase offset correction and De-mapping* tasks, respectively. The mapping of the HiperLAN/2 receiver on the heterogeneous reconfigurable SoC has been depicted in Figure 4.13.

Switching to UMTS means that 2 MONTIUM TPs have to be configured as *pulse-shape filter* and *Rake* functions. One Field Programmable Gate Array (FPGA) tile has to be configured as *scrambling code generator*. Furthermore, the *channel estimator* and *path searcher* need to be programmed on the GPP. Figure 5.7 shows the mapping of the UMTS receiver on the heterogeneous reconfigurable SoC.

After the decision of the control-manager to switch from HiperLAN/2 to UMTS, the control-manager has to write the new configurations to the reconfigurable processing tiles. Furthermore, the communication channels between the different processing elements in the SoC have to be configured. Communication channels are defined between the pulse-shape filter and the Rake receiver, between the scrambling code generator and the Rake receiver, and between the GPP and the Rake receiver. Storing the configuration files in the MONTIUM TPs requires a few microseconds as seen in Section 5.4.4. The control-manager also adapts the clocks of the processing elements to the required frequencies as discussed in Section 5.4.5.

7.4.2 Rake versus Equalizer

According to [60, 107] the Wideband CDMA (WCDMA) receiver can be implemented in different ways, using a Rake receiver or an equalizer. Equalizer-based WCDMA receivers yield better performance than Rake receivers under severe Multiple Access Interference (MAI) conditions. Depending on the conditions of the environment, the control-manager can decide whether the Rake function or equalizer is used in the UMTS receiver. This means that in UMTS mode one processing tile is configured as *equalizer* and one as *pulse-shape filter*. At a certain moment the control-manager decides to reconfigure the MONTIUM TP that acts as equalizer to the Rake function. So, the control-manager has to write the new configuration to the MONTIUM TP. All communication channels between the tiles remain the same and are not reconfigured.

7.4.3 Adapting the number of Rake fingers

The number of Rake fingers affects the total signal power that the WCDMA receiver receives. The environment influences the number of reflected signals, so-called multipaths, that arrive at the receiving terminal. Based on the number of reflected signals the control-

manager decides how many fingers are applied in the Rake receiver to receive maximum signal power.

After the decision of the control-manager to change the number of fingers in the Rake receiver, the control-manager has to change the configuration of the processing element which implements the Rake function. In Section 5.4.4 the number of Rake fingers are adapted by dynamic reconfiguration. The reconfiguration time is in the order of nanoseconds. By reducing the number of fingers in the Rake receiver, the clock frequency of the MONTIUM TP which implements the Rake function can be reduced as well. The control-manager also handles the settings of the system clock of the individual MONTIUM TPs.

7.5 Summary

In this chapter the implementation of a System-on-Chip (SoC) for Software Defined Radio (SDR) applications has been analyzed. The ANNABELLE SoC has been presented as an instant of the initial CHAMELEON SoC template approach.

7.5.1 Conclusions

The ANNABELLE SoC has been designed for digital radio broadcasting applications. The prototype chip contains an ARM926 General Purpose Processor (GPP), a reconfigurable subsystem and peripherals which interface with memories and provide communication with off-chip components. The reconfigurable subsystem inherits the conceptual architecture of the CHAMELEON SoC. The reconfigurable subsystem contains four MONTIUM Tile Processors (TPs) with HYDRA Communication and Configuration Units (CCUs). The MONTIUM processing tiles are connected to each other by a circuit-switched Network-on-Chip (NoC). The entire reconfigurable subsystem is connected with an Advanced High-performance Bus (AHB) to the ARM926 processor.

The MONTIUM TPs are customized at design time for performing Fast Fourier Transform (FFT) algorithms that appear in Digital Audio Broadcasting (DAB) and Digital Radio Mondiale (DRM) systems. The MONTIUM TPs are typically used as reconfigurable co-processor accelerators for the ARM926 processor.

7.5.2 Discussion

Channel decoding in wireless communication systems usually involves mechanisms for interleaving in the transmitter and receiver. However, the characteristics of interleaving for the different wireless communication standards are highly diverse. One additional element that is capable of efficiently performing interleaving operations in the SoC is required. An initial reconfigurable memory tile template has been proposed which is capable of providing block mode and streaming mode communication. The memory tile incorporates a flexible Address Generation Unit (AGU) with advanced modulo support.

Experiences from mapping the baseband processing and channel decoding functions of different wireless communication receivers in the previous chapters provided valuable

insight in multi-standard multi-mode adaptive receivers. Based on this information, scenarios are anticipated how multi-standard multi-mode adaptive wireless communication receivers are implemented on heterogeneous reconfigurable SoC platforms.

Most baseband processing and channel decoding functions are implemented on coarse-grained reconfigurable processing elements in the heterogeneous SoC. However, a centralized control-manager has to be implemented which can control the entire SoC. The coordination functions of the control-manager are typically implemented on top of a light-weight (distributed) run-time Operating System (OS) that runs on a GPP [32, 101].

Chapter 8

Conclusion

Wireless communication systems are progressing towards multi-standard multi-mode communication systems. These communication systems are capable of dynamically adapting to various conditions while achieving optimal performance. The focus of the work in this thesis is on the implementation of adaptive multi-standard multi-mode wireless communication systems on heterogeneous reconfigurable hardware.

This chapter concludes the work described in this thesis. Furthermore, directions for future research are shortly discussed.

8.1 Introduction

This thesis addresses issues in the architecture and design of adaptive wireless communication systems. In this chapter the final conclusions of the work described in this thesis are drawn. The conclusions are derived based on the partial conclusions of the separate chapters in this thesis.

The objectives of this work are given in Chapter 1. The final conclusions in Section 8.2 reflect these objectives. Section 8.3 evaluates the lessons learned. Directions for future research are given in Section 8.4 accompanied by a discussion.

8.2 Conclusions

The work described in this thesis contributed to different areas of reconfigurable computing and Software Defined Radio:

- Important Digital Signal Processing (DSP) algorithms in wireless communication systems that are based on Orthogonal Frequency Division Multiplexing (OFDM) and Wideband CDMA (WCDMA) techniques were investigated. The investigation established the requirements for implementing multi-standard multi-mode wireless communication receivers on heterogeneous reconfigurable System-on-Chips (SoCs); (*Chapter 4, 5 and 6*)
- We showed that a single SoC platform can support various standards with a performance similar to an Application Specific Integrated Circuit (ASIC) implementation:
    1. The digital baseband processing of the HiperLAN/2 receiver was mapped on the MONTIUM TP as an illustration for OFDM wireless communication receivers; (*Chapter 4*)
    2. The Rake receiver was mapped on the MONTIUM TP to illustrate the mapping of WCDMA wireless communication receivers; (*Chapter 5*)
    3. The Viterbi and Turbo decoder were mapped on the MONTIUM TP in order to complement the baseband processing algorithms; (*Chapter 6*)
- The Arithmetic Logic Units (ALUs) of the MONTIUM Tile Processor (TP) were modified by adding Add Compare Select (ACS) support; (*Chapter 6*)
- We showed that a coarse-grained architecture with partial reconfiguration capability can be used efficiently for adapting receiver settings dynamically and we showed that energy can be saved by adaptivity; (*Chapter 5*)
- We developed a prototype implementation of a heterogeneous reconfigurable System-on-Chip (SoC). The SoC contains four coarse-grained reconfigurable MONTIUM Tile Processors (TPs) and one General Purpose Processor (GPP). (*Chapter 7*)

Wireless communication techniques Trends identified in Third Generation (3G) and Fourth Generation (4G) wireless communication systems show the fast appearance of new wireless communication standards as well as the quick evolution of existing wireless communication standards. The multitude of 3G / 4G wireless communication standards results in the concept of multi-standard multi-mode wireless communication devices.

Moreover, 3G and 4G wireless communication systems are provided with adaptive functions. The abundance of wireless communication standards and the adaptivity features result in wireless communication terminals that can adapt to Quality of Service (QoS) requirements and to the wireless channel conditions at a certain location.

The complexity of OFDM standards increases considerably due to e.g. increasing bandwidth requirements. The complexity of de-multiplexing the various information streams of the Medium Access Control (MAC) transmission frame in the OFDM receiver strongly depends on the manner of sending information on the sub-carriers of the OFDM symbol. Early OFDM standards employed simple data modulation techniques (e.g. DAB), whereas multi-level QAM modulation was introduced in more recent OFDM systems (e.g. HiperLAN/2 and DRM).

In early OFDM systems consecutive sub-carriers in an OFDM symbol were modulated with equal QAM constellations. Furthermore, the allocation of data to the separate sub-carriers was always fixed. However, in e.g. the DRM standard the function of the allocated sub-carriers differs for every sub-carrier in the OFDM symbol. The varying modulation of the consecutive sub-carriers introduces additional complexity in the demodulation process of the OFDM receiver.

Adaptivity Trends in wireless communication systems show many opportunities for exploiting adaptivity in DSP algorithms. Three levels of adaptivity have been identified: *standards level*, *algorithm-selection level* and *algorithm-parameter level*. The Software (Defined) Radio concept is a key enabler to efficiently employ the different levels of adaptivity.

The standards level of adaptivity allows the terminal to adapt the communication standard that is used to satisfy the user requirements. The algorithm-selection level of adaptivity allows the terminal to select the algorithms that satisfy the QoS requirements in the given communication environment in the most efficient manner. The algorithm-parameter level of adaptivity allows the terminal to change the parameters of a specific algorithm.

A reconfigurable and adaptive terminal is able to track the QoS requirements and communication environment on a much finer grained scale than a traditional non-adaptive terminal.

MONTIUM extensions Experiences from mapping DSP baseband processing and channel decoding algorithms resulted in modifications of the MONTIUM's reconfigurable architecture. We observed that conditional operations (e.g. conditional multiplexer functionality like ACS operations) that are heavily applied in channel decoding algorithms were not well supported in the MONTIUM architecture. The problem which arises from

the ACS operation is that the operation merges the data path and the control path. With very limited additional logic in the MONTIUM architecture it is possible to bypass the MONTIUM sequencer and implement conditional operations in hardware. The advantage is that sequencer instructions are executed by the sequencer, while conditional operations are applied continuously as well. Consequently the conditional *compare-select* operation can run in parallel in all 5 ALUs of the MONTIUM TP.

Dynamic reconfiguration The configuration sizes of the MONTIUM TP are small for the different DSP functions in the HiperLAN/2 receiver. The configuration size of DSP kernels in the HiperLAN/2 receiver is typically less than 1 kB of configuration data. This means that the MONTIUM TPs are typically reconfigured for HiperLAN/2 baseband processing in less than 5 μs[1].

The configuration size of the Rake receiver with 4 fingers mapped on the MONTIUM TP is only 858 bytes. Hence, the MONTIUM is configured in 4.29 μs[1]. Mapping the Rake receiver on the MONTIUM TP gives the flexibility to adapt the receiver quickly to typical operating conditions. Adaptation of the Rake receiver to typical environmental conditions, such as changing path-delay profile, is done by partial reconfiguration. The characteristics of the Rake receiver are semi-instantly changed through partial reconfiguration of the MONTIUM TP. For instance, the number of fingers of the Rake receiver can be reconfigured in 12 clock cycles, which requires 120 ns[1]reconfiguration time.

The configuration overhead of the Viterbi or Turbo decoder is relatively small, as the configuration files of both decoders are small. Hence, changing the functionality of the channel decoder from Turbo to Viterbi, or vice versa, can be done dynamically because of the short reconfiguration times. The reconfiguration time of the Viterbi or Turbo decoder implementation on the MONTIUM TP is less than 7 μs[1]. Depending on the desired communication standard, one can configure the reconfigurable hardware in the mobile wireless multi-standard receiver to implement the right channel decoder according to the desired standard.

Performance All implementations mapped on the MONTIUM TP have been verified against floating-point reference models. Simulations with typical HiperLAN/2 channel models show that the implemented MONTIUM-based HiperLAN/2 receiver is capable of receiving data with the minimum required sensitivity as defined by the standard.

Simulations for typical Universal Mobile Telecommunications System (UMTS) scenarios show hardly any difference in performance between the reference model and the MONTIUM-based Rake receiver implementation ($< 2\%$ BER difference). Experiments show that proper input scaling of data influences the performance accuracy of the MONTIUM-based Rake receiver considerably.

Moreover, the performance of the MONTIUM-based Viterbi decoder and the Turbo decoder in terms of Bit Error Rate (BER) versus Signal-to-Noise Ratio (SNR) is similar to a floating-point reference implementation. The BER performance of the MONTIUM-based Viterbi decoder is in worst case $3 \cdot 10^{-3}$ higher than the BER obtained by the

[1] Assuming the configuration clock runs at 100 MHz.

floating-point reference implementation. Furthermore, the MONTIUM-based decoder implementations can achieve (about 10 times) higher throughput than decoders implemented in a General Purpose Processor.

Power consumption The dynamic power consumption of the MONTIUM-based Rake receiver in 130 nm CMOS technology is estimated to be 0.470 mW/MHz^2. The dynamic power consumption of Rake functionality implemented in an ASIC is about 3 to 7 times lower compared to the MONTIUM alternative. As expected, the ASIC implementation is more energy efficient than an implementation in reconfigurable hardware, however, the ASIC implementation is fixed and the functionality of the ASIC cannot be changed. The MONTIUM-based Rake receiver, on the other hand, is at least 10 times more energy efficient than a Digital Signal Processor (DSP) implementation.

The normalized dynamic power consumption of the MONTIUM-based Viterbi decoder is estimated to be 0.309 mW/MHz^2. This means that the energy consumption of the implemented Viterbi decoder mapped on the MONTIUM for DAB is 14.5 nJ per decoded bit.

8.3 Lessons learned

In Chapter 4, 5 and 6 various DSP algorithms are mapped on the coarse-grained reconfigurable MONTIUM TP. Furthermore, a prototype implementation of a heterogeneous reconfigurable SoC has been presented in Chapter 7. These achievements showed that a single SoC platform can be designed which can support various wireless communication standards. Moreover, there are several lessons that can be learned.

Design methodology The multidisciplinary design approach is leading in hardware / software co-design. System development requires knowledge from both the application domain as well as the (reconfigurable) hardware domain. To efficiently implement algorithms on e.g. the MONTIUM TP, it is common to understand why certain operations are applied instead of how they are applied.

Tiled System-on-Chip architecture The tiled SoC template, as presented in Figure 3.3, is suitable for implementation of streaming DSP algorithms that are applied in e.g. wireless communication receivers. The data streams between the various DSP kernels are transparently mapped on the channels of the flexible Network-on-Chip (NoC) in the SoC architecture. The DSP kernels are mapped on suitable processing tiles.

For an adaptive multi-standard multi-mode wireless communication receiver most DSP kernels will be mapped on coarse-grained reconfigurable processing tiles, such as the MONTIUM processing tile. Furthermore, a GPP is required in the SoC to control the entire adaptive multi-standard multi-mode wireless communication receiver.

[2] Including the power consumption of SRAM cells.

The number of required processing tiles depends entirely on the application that has to be mapped on the tiled SoC architecture. In Chapter 4 the baseband processing of the HiperLAN/2 receiver has been ingenuously mapped on four MONTIUM processing tiles. Enhanced partitioning will reduce the number of MONTIUM processing tiles to three for baseband processing. One additional MONTIUM processing tile is required for Forward Error Correction (FEC) decoding (e.g. the Viterbi algorithm).

All wireless communication receivers incorporate de-interleaver functions. The de-interleaver functions require special memory resources in the tiled SoC architecture. Ideally, the interleaved data is streamed in a memory tile via the NoC and the de-interleaved data is streamed out of the memory tile via the NoC.

Reconfigurable hardware Customizing the MONTIUM TP for dedicated DSP algorithm domains can be beneficial in terms of chip area and power costs. For FEC decoding algorithms the multiplier in *level 2* of the ALU is superfluous. Removing the multiplier reduces the chip area of the MONTIUM TP as well as the delay of the worst-time critical path.

Introducing pipelining in the ALU of the MONTIUM TP will increase the performance of the MONTIUM TP. Pipelining does not only increase the performance of the coarse-grained reconfigurable processor, but will also relieve the task of the chip synthesis tool. Moreover, test patterns can be generated more efficiently for hardware verification using scan chains.

Algorithms Multiple DSP kernels in wireless communication applications have been mapped on the MONTIUM TP. The MONTIUM TP can be used to implement common DSP algorithms that are applied in wireless communication systems.

Multiple OFDM receivers have been investigated. Although the OFDM receivers are all designed according to the generic OFDM template, the degree of complexity for different OFDM receivers varies enormously. The allocation of the different sub-carriers in the OFDM signal can highly influence the decoding complexity of the receiver; e.g. in Digital Radio Mondiale (DRM) every sub-carrier takes multiple functions as well as multiple constellation schemes which differ in time. In Digital Audio Broadcasting (DAB) on the other hand, every sub-carrier always takes the same function and is always modulation using the same constellation. The irregular modulation and function allocation of the different sub-carriers complicates the decoding of OFDM data streams. Hence, additional control is required during de-multiplexing of the data which cannot efficiently be performed on the MONTIUM TP.

Development tools Design automation is needed to guide engineers to develop their applications on the target architecture. Experiences from mapping applications on the MONTIUM TP showed that design automation can relieve the mapping task for *register allocation* and *bus allocation*; proper administration is indispensable.

Mapping algorithms / applications on the MONTIUM TP is an iterative process with many degrees of freedom. Thoroughly analyzing the algorithms that have to be imple-

mented is necessary to identify dependencies between ALU instructions, register instructions and interconnect instructions. The identification of dependencies between these instructions creates opportunities to reduce the degree of freedom by fixing the instructions with the highest dependency.

Automatically mapping algorithms on the MONTIUM TP can be facilitated by development tools that are guided by suggestions from application developers in case of complex algorithms with many dependencies.

8.4 Discussion & future directions

Application domain knowledge as well as knowledge about the reconfigurable target architecture are essential and should be an integral part of the design methodology. Two situations that appeared in Chapter 5 and 6 stress the importance of an integral design methodology:

- The importance of properly scaling the fixed-point input data in the MONTIUM-based Rake receiver is derived through experiments. In bad channel conditions, the floating-point reference receiver performs better than the MONTIUM-based Rake receiver ($< 20\%$ BER difference). Correctly scaling the fixed-point input data results in the situation that the floating-point reference receiver and the MONTIUM-based Rake receiver show equal BER versus SNR performance ($< 2\%$ BER difference);

- When implementing the Viterbi algorithm on the MONTIUM TP, the Register Exchange (RE) approach is obviously the best method to store all survivor bit sequences. The RE approach stores the entire decoded output sequence for each state during path metric updating. Therefore, in each stage of the trellis the decoded output sequence is known and there is no need to traceback. Moreover, bit sequences (Register Exchange) are stored more efficiently than individual bits (Traceback) in the coarse-grained MONTIUM TP. Since the data path in the MONTIUM TP is limited to 16-bits, the Viterbi algorithm has been implemented in a hybrid Register Exchange / Traceback manner.

The work presented in this thesis addresses several directions for future research:

- *Compiler design for reconfigurable architectures*
  In [48], the first steps toward the automated mapping and scheduling of DSP algorithms on the MONTIUM TP have been presented. Currently, algorithms can be mapped on the MONTIUM TP using an assembly-like MONTIUM Configuration Description Language (CDL). High-level programming tools are required to ease the implementation of applications on reconfigurable hardware;

- *Distributed Operating System in dynamically reconfigurable SoC architectures*
  Most baseband processing and channel decoding functions are implemented on

coarse-grained reconfigurable processing elements in the heterogeneous SoC. However, a centralized control-manager has to be implemented which can control the entire SoC. The coordination functions of the control-manager are typically implemented on top of a light-weight (distributed) run-time Operating System (OS) that runs on a GPP;

- *Cognitive Radio*
  In Chapter 2 the emerging interest in Cognitive Radio was already observed. A Cognitive Radio is a system that is aware of its operational environment and can be trained to dynamically and autonomously adjust its radio operating parameters [77]. Recent trends towards heterogeneous reconfigurable SoC architectures perfectly match with the characteristics of Cognitive Radios, which require flexible heterogeneous hardware architectures that can adapt its communication scheme rapidly to new conditions;

- *Customizing reconfigurable architectures for dedicated application domains*

  – Many DSP algorithms use only part of the data path resources that are available in the MONTIUM architecture. Investigations on channel decoder algorithms show that for Viterbi and Turbo decoding the multipliers in the MONTIUM TP are not used. Therefore, customizing the architecture by removing e.g. multipliers for a particular algorithm domain can result in architectures with smaller footprint;

  – Channel decoding in wireless communication systems usually involves mechanisms for interleaving in the transmitter and receiver. However, the characteristics of interleaving for the different wireless communication standards are highly diverse. Therefore, specialized memory architectures that are capable of efficiently performing (de-)interleaver operations in the SoC are required.

Bibliography

[1] 3^{rd} Generation Partnership Project. Technical Specification Group Radio Access Network; Multiplexing and channel coding (FDD). 3GPP TS 25.212 v4.3.0 (2001-12), January 2002.

[2] 3^{rd} Generation Partnership Project. Base Station (BS) radio transmission and reception (FDD). 3GPP TS 25.104 v6.4.0 (2003-12), January 2004.

[3] 3^{rd} Generation Partnership Project. Physical channels and mapping of transport channels onto physical channels (FDD). 3GPP TS 25.211 v6.0.0 (2003-12), January 2004.

[4] 3^{rd} Generation Partnership Project. Physical layer – General description. 3GPP TS 25.201 v6.0.0 (2003-12), January 2004.

[5] 3^{rd} Generation Partnership Project. Spreading and modulation (FDD). 3GPP TS 25.213 v7.0.0 (2006-03), March 2006.

[6] 4S. Smart chipS for Smart Surroundings. `http://www.smart-chips.net`, 2007.

[7] Altera. `http://www.altera.com`, 2007.

[8] Analog Devices. Engineer To Engineer Note – Estimating Power For the ADSP-TS101S, February 2003.

[9] Analog Devices. TigerSHARC Embedded Processor – ADSP-TS101S. Rev. B, December 2004.

[10] Analog Devices. Embedded Processing & DSP – TigerSHARC Processor. `http://www.analog.com/processors/tigersharc/`, 2006.

[11] Atmel Corporation. ATC13 Summary. `http://www.atmel.com`, 2007.

[12] Keukjoon Bang, Namshin Cho, Jaehee Cho, Heeyoung Jun, Kwangchul Kim, Hyuncheol Park, and Daesik Hong. A Coarse Frequency Offset Estimation in an OFDM System Using the Concept of the Coherence Phase Bandwidth. *IEEE Transactions on Communications*, 49(8):1320–1324, May 2001.

[13] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt. PACT XPP – A Self-Reconfigurable Data Processing Architecture. *Journal of Supercomputing*, 26(2):167–184, September 2003.

[14] B. Baumgartner, A. Hof, V. Sidorenko, and M. Bossert. Multilevel Codes: Maximum-Likelihood versus Iterative Multistage Decoding. In *Proceedings of the 7th International OFDM Workshop*, pages 148–152, Hamburg, Germany, September 2002.

[15] Anna Berno. Time and Frequency Synchronization Algorithms for HIPERLAN/2. Master's thesis, University of Padova, Italy, October 2001.

[16] Claude Berrou and Alain Glavieux. Near Optimum Error Correcting Coding And Decoding: Turbo-Codes. *IEEE Transactions on Communications*, 44(10):1261–1271, October 1996.

[17] Claude Berrou, Alain Glavieux, and Punya Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-Codes. In *Proceedings of IEEE ICC'93*, volume 2, pages 1064–1070, Geneva, Switzerland, May 1993.

[18] Mark A. Bickerstaff, David Garrett, Thomas Prokop, Charles Thomas, Benjamin Widdup, Gongyu Zhou, Linda M. Davis, Graeme Woodward, Chris Nicol, and Ran-Hong Yan. A Unified Turbo/Viterbi Channel Decoder for 3GPP Mobile Wireless in 0.18-μm CMOS. *IEEE Journal of Solid-State Circuits*, 37(11):1555–1564, November 2002.

[19] B. Bougard, S. Pollin, G. Lenoir, W. Eberle, L. Van der Perre, F. Catthoor, and W. Dehaene. Energy-Scalability Enhancement of Wireless Local Area Network Transceivers. In *Proceedings of the Fifth IEEE Workshop on Signal Processing Advances in Wireless Communication*, Lisboa, Portugal, July 2004.

[20] M.D. van de Burgwal, G.J.M. Smit, G.K. Rauwerda, and P.M. Heysters. Hydra: an Energy-efficient and Reconfigurable Network Interface. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'06)*, pages 171–177, Las Vegas, Nevada, USA, June 2006.

[21] Joseph R. Cavallaro and Mani Vaya. VITURBO: A Reconfigurable Architecture for Viterbi and Turbo Decoding. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'03)*, volume II, pages 497–500, Hong Kong, China, April 2003.

[22] R. W. Chang. Synthesis of band-limited orthogonal signals for multi-channel data transmission. *Bell Systems Technical Journal*, 46:1775–1796, 1966.

[23] J.W. Choi and Y.H. Lee. Adaptive Channel Estimation in DS-CDMA Systems. *ECTI Transactions on Electrical Engineering, Electronics, and Communications*, 2(1), February 2004.

[24] John M. Cioffi, Glen P. Dudevoir, M. Vedat Eyuboglu, and G. David Forney, Jr. MMSE Decision-Feedback Equalizers and Coding – Part I: Equalization Results. *IEEE Transactions on Communications*, 43(10):2582–2594, October 1995.

[25] John M. Cioffi, Glen P. Dudevoir, M. Vedat Eyuboglu, and G. David Forney, Jr. MMSE Decision-Feedback Equalizers and Coding – Part II: Coding Results. *IEEE Transactions on Communications*, 43(10):2595–2604, October 1995.

[26] James W. Cooley and John W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19:297–301, April 1965.

[27] Erik Dahlman, Per Beming, Jens Knutsson, Fredrik Oversjö, Magnus Persson, and Christiaan Roobol. WCDMA – The Radio Interface for Future Mobile Multimedia Communications. *IEEE Transactions on Vehicular Technology*, 47(4):1105–1118, November 1998.

[28] Marijn C. Damstra. Path search algorithms for application in W-CDMA systems. Master's thesis, University of Twente, Enschede, The Netherlands, August 2004.

[29] John Dielissen, Jef van Meerbergen, Marco Bekooij, Françoise Harmsze, Sergej Sawitzki, Jos Huisken, and Albert van der Werf. Power-efficient layered Turbo Decoder processor. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 246–251, Munich, Germany, March 2001.

[30] Andrew Duller, Gajinder Panesar, and Daniel Towner. Parallel Processing – the picoChip way! In *Proceedings of Communicating Process Architectures 2003*, pages 125–138, Enschede, The Netherlands, September 2003.

[31] Andrew Duller, Daniel Towner, Gajinder Panesar, Alan Gray, and Will Robbins. picoArray technology: the tool's story. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, volume 3, pages 106–111, Munich, Germany, March 2005.

[32] eCos. `http://ecos.sourceware.org`, 2007.

[33] Don Edenfeld, Andrew B. Kahng, Mike Rodgers, and Yervant Zorian. 2003 Technology Roadmap for Semiconductors. *IEEE Computer*, 37(1):47–56, January 2004.

[34] Hannes Ekström, Anders Furuskär, Jonas Karlsson, Michael Meyer, Stefan Parkvall, Johan Torsner, and Mattias Wahlqvist. Technical Solutions for the 3G Long-Term Evolution. *IEEE Communications Magazine*, 44(3):38–45, March 2006.

[35] ETSI. Broadband Radio Access Networks (BRAN); HiperLAN Type 2; Data Link Control (DLC) Layer Part 1: Basic Data Transport Functions. ETSI TS 101 761-1 v1.1.1 (2000-04), April 2000.

[36] ETSI. Broadband Radio Access Networks (BRAN); HiperLAN Type 2; Physical (PHY) Layer. ETSI TS 101 475 v1.2.2 (2001-02), February 2001.

[37] ETSI. Digital Radio Mondiale (DRM); System Specification. ETSI TS 101 980 v1.1.1 (2001-09), September 2001.

[38] ETSI. Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers. ETSI EN 300 401 v1.3.3 (2001-05), May 2001.

[39] ETSI. Digital Audio Broadcasting (DAB); Transport of Advanced Audio Coding (AAC) audio. ETSI TS 102 563 v1.1.1 (2007-02), February 2007.

[40] David Falconer, Sirikiat Lek Ariyavisitakul, Anader Benyamin-Seeyar, and Brian Eidson. Frequency Domain Equalization for Single-Carrier Broadband Wireless Systems. *IEEE Communications Magazine*, 40(4):58–66, April 2002.

[41] Gerhard Fettweis. Algebraic Survivor Memory Management Design for Viterbi Detectors. *IEEE Transactions on Communications*, 43(9):2458–2463, September 1995.

[42] G. David Forney, Jr. Burst-Correcting Codes for the Classic Bursty Channel. *IEEE Transactions on Communications Technology*, 19(5):772–781, October 1971.

[43] Robert G. Gallager. *Information theory and reliable communications*. John Wiley & Sons, USA, 1968.

[44] Roberto Garello, Guido Montorsi, Sergio Benedetto, and Giovanni Cancellieri. Interleaver Properties and Their Applications to the Trellis Complexity Analysis of Turbo Codes. *IEEE Transactions on Communications*, 49(5):793–807, May 2001.

[45] Tobias Gemmeke, Michael Gansen, and Tobias G. Noll. Implementation of Scalable Power and Area Efficient High-Throughput Viterbi Decoders. *IEEE Journal of Solid-State Circuits*, 37(7):941–948, July 2002.

[46] I.J. Good. The Interaction Algorithm and Practical Fourier Analysis. *Journal of the Royal Statistical Society. Series B (Methodological)*, 20(2):361–372, 1958.

[47] I.J. Good. The Interaction Algorithm and Practical Fourier Analysis: An Addendum. *Journal of the Royal Statistical Society. Series B (Methodological)*, 22(2):372–375, 1960.

[48] Yuanqing Guo. *Mapping Applications to a Coarse-Grained Reconfigurable Architecture*. PhD thesis, University of Twente, Enschede, The Netherlands, September 2006.

[49] Lajos Hanzo, Lie-Liang Yang, Matthias Münster, and Byoung-Jo Choi. Recital on Multicarrier Communications: Space-Time Coded versus Adaptive OFDM/MC-CDMA. In *Proceedings of SympoTIC'04*, pages X–XXVIII, Bratislava, Slovakia, October 2004.

[50] Lasse Harju, Mika Kuulusa, and Jari Nurmi. Flexible Implementation of a WCDMA Rake Receiver. *Journal of VLSI Signal Processing*, 39(1–2):147–160, January–February 2005.

[51] Reiner Hartenstein. A Decade of Reconfigurable Computing: a Visionary Retrospective. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 642–649, Munich, Germany, March 2001.

[52] John L. Hennessy, David A. Patterson, and Andrea C. Arpaci-Dusseau. *Computer architecture: a quantitative approach*. Morgan Kaufmann, Amsterdam, The Netherlands, fourth edition, 2006.

[53] Russell Henning and Chaitali Chakrabarti. An Approach for Adaptively Approximating the Viterbi Algorithm to Reduce Power Consumption While Decoding Convolutional Codes. *IEEE Transactions on Signal Processing*, 52(5):1443–1451, May 2004.

[54] Paul M. Heysters. *Coarse-Grained Reconfigurable Processors – Flexibility meets Efficiency*. PhD thesis, University of Twente, Enschede, The Netherlands, September 2004.

[55] Paul M. Heysters, Gerard J. M. Smit, and Egbert Molenkamp. A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems. *Journal of Supercomputing*, 26(3):283–308, November 2003.

[56] Botaro Hirosaki. An Orthogonally Multiplexed QAM System Using the Discrete Fourier Transform. *IEEE Transactions on Communications*, 29(7):982–989, July 1981.

[57] L.F.W. van Hoesel. Design and implementation of a software defined HiperLAN/2 physical layer model for simulation purposes. Master's thesis, University of Twente, Enschede, The Netherlands, August 2002.

[58] Frank Hofmann, Christian Hansen, and Wolfgang Schäfer. Digital Radio Mondiale (DRM) Digital Sound Broadcasting in the AM Bands. *IEEE Transactions on Broadcasting*, 49(3):319–328, September 2003.

[59] H. Holma and A. Toskala. *WCDMA for UMTS: Radio Access for Third Generation Mobile Communications*. John Wiley & Sons, England, 2001.

[60] Kari Hooli. *Equalization in WCDMA Terminals*. PhD thesis, University of Oulu, Oulu, Finland, December 2003.

[61] Michiel Horsman, Thijs Mutter, Marcel van der Veen, Maarten Witteman, and Karel Walters. Energy comparison of Viterbi algorithm on ARM and XScale architecture. University of Twente, Enschede, The Netherlands, 2004. Internal report.

[62] IEEE Standards Board. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: high-speed physical layer in the 5 GHz band. IEEE Std 802.11a-1999 (R2003), June 2003.

[63] International Technology Roadmap for Semiconductors. `http://www.itrs.net`, 2007.

[64] Nikolay K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. PhD thesis, University of Twente, Enschede, The Netherlands, January 2007.

[65] Jagadeesh Kaza and Chaitali Chakrabarti. Design and Implementation of Low-Energy Turbo Decoders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(9):968–977, September 2004.

[66] Lutger Kunst. Design and implementation of a RAKE-finger on a TigerSHARC DSP. Master's thesis, University of Twente, Enschede, The Netherlands, August 2002.

[67] Jeroen Leijten, Geoffrey Burns, Jos Huisken, Erwin Waterlander, and Antoine van Wel. AVISPA: A Massively Parallel Reconfigurable Accelerator. In *Proceedings of the International Symposium on System-on-Chip (SoC 2003)*, pages 165–168, Tampere, Finland, November 19–21 2003.

[68] Shu Lin and Daniel J. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, USA, 1983.

[69] Guido Masera, Gianluca Piccinini, Massimo Ruo Roch, and Maurizio Zamboni. VLSI Architecture for Turbo Codes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(3):369–379, September 1999.

[70] K. Masselos, S. Blionas, and T. Rautio. Reconfigurability requirements of wireless communication systems. In *Proceedings of the IEEE Workshop on Heterogeneous Reconfigurable Systems on Chip*, Hamburg, Germany, April 2002.

[71] The MathWorks. MATLAB - The Language of Technical Computing. `http://www.mathworks.com/products/matlab/`, 2006.

[72] J. Medbo and P. Schramm. Channel Models for HIPERLAN/2. ETSI/BRAN 3ERI085B, September 1998.

[73] Mentor Graphics. ModelSim. `http://www.model.com/products/`, 2006.

[74] Jeong-Ki Min and Hyoung-Kyu Song. Frequency Synchronization for Digital Audio Broadcasting. *IEEE Transactions on Consumer Electronics*, 49(2):290–295, May 2003.

[75] J. Mitola III. Software Radios – Survey, Critical Evaluation and Future Directions. In *Proceedings of the National Telesystems Conference (NTC-92)*, pages 13/15–13/23, May 1992.

[76] Joseph Mitola III. Technical Challenges in the Globalization of Software Radio. *IEEE Communications Magazine*, 37(2):84–89, February 1999.

[77] Joseph Mitola III. *Cognitive Radio – An Integrated Agent Architecture for Software Defined Radio*. PhD thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, May 2000.

[78] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, April 19, 1965.

[79] Gordon E. Moore. Progress in Digital Integrated Electronics. In *Proceedings of the IEEE International Electron Devices Meeting*, pages 11–13, 1975.

[80] Hyung G. Myung, Junsung Lim, and David J. Goodman. Single Carrier FDMA for Uplink Wireless Transmission. *IEEE Vehicular Technology Magazine*, 1(3):30–38, September 2006.

[81] Yrjö Neuvo. Cellular Phones as Embedded Systems. In *Proceedings of the ISSCC 2004*, pages 32–37, February 2004.

[82] Max Nilsson. Efficient ASIC implementation of a WCDMA Rake Receiver. Master's thesis, Luleå University of Technology, Stockholm, Sweden, April 2002.

[83] PACT XPP Technologies . `http://www.pactcorp.com`, 2007.

[84] Gajinder Panesar, Daniel Towner, Andrew Duller, Alan Gray, and Will Robbins. Deterministic Parallel Processing. *International Journal of Parallel Programming*, 34(4):323–341, August 2006.

[85] S.F. Peerlkamp. Mapping DRM Baseband Processing on a Heterogeneous Architecture. Master's thesis, University of Twente, Enschede, The Netherlands, May 2006.

[86] picoChip. `http://www.picochip.com`, 2007.

[87] Jordy Potman, Fokke Hoeksema, and Kees Slump. Tradeoffs between Spreading Factor, Symbol Constellation Size and Rake Fingers in UMTS. In *Proceedings of ProRISC 2003*, pages 543–548, Veldhoven, The Netherlands, November 2003.

[88] Marc Quax and Ingolf Held. Multi-Standard Embedded Processor for Viterbi Decoding. In *Proceedings of GSPx: TV to Mobile 2005*, Eindhoven, The Netherlands, May 2005.

[89] Marc Quax, Jos Huisken, and Jef van Meerbergen. A Scalable Implementation of a Reconfigurable WCDMA Rake Receiver. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 230–235, Munich, Germany, February 2004.

[90] Charles M. Rader. Memory Management in a Viterbi Decoder. *IEEE Transactions on Communications*, 29(9):1399–1401, September 1981.

[91] John L. Ramsey. Realization of Optimum Interleavers. *IEEE Transactions on Information Theory*, 16(3):338–345, May 1970.

[92] Theodore S. Rappaport. *Wireless Communications: Principles & Practice*. Prentice Hall PTR, Upper Saddle River, New Jersey, USA, 1996.

[93] Recore Systems. `http://www.recoresystems.com`, 2007.

[94] Arnaud Rivaton, Jérôme Quévremont, Qiwei Zhang, Pascal Wolkotte, and Gerard Smit. Implementing Non Power-of-Two FFTs on Coarse-Grain Reconfigurable Architectures. In *Proceedings of the International Symposium on System-on-Chip (SoC 2005)*, pages 82–85, Tampere, Finland, November 2005.

[95] Patrick Robertson, Emmanuelle Villebrun, and Peter Hoeher. A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain. In *Proceedings of IEEE ICC'95*, volume 2, pages 1009–1013, Seatle, USA, June 1995.

[96] Roel Schiphorst. *Software-Defined Radio for Wireless Local-Area Networks*. PhD thesis, University of Twente, Enschede, The Netherlands, September 2004.

[97] Timothy M. Schmidl and Donald C. Cox. Robust Frequency and Timing Synchronization for OFDM. *IEEE Transactions on Communications*, 45(12):1613–1621, December 1997.

[98] SDR Forum. `http://www.sdrforum.org`, 2007.

[99] C. E. Shannon. A Mathematical Theory of Communication. *Bell Systems Technical Journal*, 27:379–423,623–656, 1948.

[100] Silicon Hive. `http://www.siliconhive.com`, 2007.

[101] B.A. van Sisseren. Design of a Lightweight Real-Time Streaming Kernel. Master's thesis, University of Twente, Enschede, The Netherlands, August 2007.

[102] Lodewijk T. Smit. *Energy-Efficient Wireless Communication*. PhD thesis, University of Twente, Enschede, The Netherlands, January 2004.

[103] Srikathyayani Srikanteswara, Ramesh Chembil Palat, Jeffrey H. Reed, and Peter Athanas. An Overview of Configurable Computing Machines for Software Radio Handsets. *IEEE Communications Magazine*, 41(7):134–141, July 2003.

[104] Stretch. `http://www.stretchinc.com`, 2007.

[105] Peter Stuckmann and Rainer Zimmermann. Toward Ubiquitous and Unlimited-Capacity Communication Networks: European Research in Framework Programme 7. *IEEE Communications Magazine*, 45(5):148–157, May 2007.

[106] Synopsys. `http://www.synopsys.com/products/`, 2006.

[107] R. Tanner and J. Woodard, editors. *WCDMA Requirements and Practical Design*. John Wiley & Sons, England, 2004.

[108] Tensilica. `http://www.tensilica.com`, 2007.

[109] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung. Reconfigurable Computing: Architectures and Design Methods. *IEE Proceedings - Computers & Digital Techniques*, 152(2):193–207, March 2005.

[110] Walter Tuttlebee. *Software Defined Radio: Enabling Technologies*. John Wiley & Sons, England, 2002.

[111] Walter Tuttlebee. *Software Defined Radio: Origins, Drivers and Internal Perspectives*. John Wiley & Sons, England, 2002.

[112] Andrew J. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, April 1967.

[113] B.W. van der Wal. Mapping the Baseband Processing of a DRM receiver to the Montium Architecture. Master's thesis, University of Twente, Enschede, The Netherlands, January 2006.

[114] Robert H. Walden. Analog-to-Digital Converter Survey and Analysis. *IEEE Journal on Selected Areas in Communications*, 17(4):539–550, April 1999.

[115] Karel H.G. Walters. Cognitive Radio on a reconfigurable platform. Master's thesis, University of Twente, Enschede, The Netherlands, August 2007.

[116] S. B. Weinstein and Paul M. Ebert. Data Transmission by Frequency-Division Multiplexing Using the Discrete Fourier Transform. *IEEE Transactions on Communication Technology*, 19(5):628–634, October 1971.

[117] Richard D. Wesel, Xuething Liu, and Wei Shi. Trellis Codes for Periodic Erasures. *IEEE Transactions on Communications*, 48(6):938–947, June 2000.

[118] Pascal T. Wolkotte, Gerard J.M. Smit, Gerard K. Rauwerda, and Lodewijk T. Smit. An Energy-Efficient Reconfigurable Circuit Switched Network-on-Chip. In *Proceedings of the 12th Reconfigurable Architectures Workshop (RAW 2005)*, Denver, Colorado, USA, April 2005.

[119] Pascal T. Wolkotte, Marcel D. van de Burgwal, and Gerard J.M. Smit. Non-Power-of-Two FFTs: Exploring the Flexibility of the MONTIUM. In *Proceedings of the International Symposium on System-on-Chip (SoC 2006)*, pages 167–170, Tampere, Finland, November 2006.

[120] Yufei Wu. *Implementation of Serial and Parallel Concatenated Convolutional Codes*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, April 2000.

[121] Wm. A. Wulf and Sally A. McKee. Hitting the Memory Wall: Implications of the Obvious. *ACM SIGARCH Computer Architecture News*, 23(1):20–24, 1995.

[122] Xilinx. `http://www.xilinx.com`, 2007.

[123] R. E. Ziemer and R. L. Peterson. *Introduction to digital communication*. Prentice Hall, USA, second edition, 2001.

List of Publications

[P1] G. K. Rauwerda, G. J. M. Smit, L. F. W. van Hoesel, and P. M. Heysters. Mapping Wireless Communication Algorithms to a Reconfigurable Architecture. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'03)*, pages 242 – 251, Las Vegas, Nevada, USA, June 23–26, 2003. CSREA Press. ISBN 1-932415-05-X.

[P2] Gerard Rauwerda, Jordy Potman, Fokke Hoeksema, and Gerard Smit. Adaptive Wireless Networking. In *Proceedings of the 4th PROGRESS Symposium on Embedded Systems*, pages 205 – 211, Nieuwegein, The Netherlands, October 22, 2003. ISBN 90-73461-37-5.

[P3] Paul M. Heysters, Gerard J.M. Smit, Bert Molenkamp, and Gerard K. Rauwerda. Flexibility of the Montium Coarse-Grained Reconfigurable Processing Tile. In *Proceedings of the 4th PROGRESS Symposium on Embedded Systems*, pages 102 – 108, Nieuwegein, The Netherlands, October 22, 2003. ISBN 90-73461-37-5.

[P4] Gerard K. Rauwerda and Gerard J.M. Smit. Adaptivity and Reconfigurability in Wireless Communications. In *Proceedings of the 14th ProRISC workshop*, pages 549 – 554, Veldhoven, The Netherlands, November 26–27, 2003. ISBN 90-73461-39-1.

[P5] Paul M. Heysters, Gerard K. Rauwerda, and Gerard J.M. Smit. Implementation of a HiperLAN/2 Receiver on the Reconfigurable Montium Architecture. In *Proceedings of the 11th Reconfigurable Architectures Workshop (RAW 2004)*, Santa Fé, New Mexico, USA, April 26–27, 2004. IEEE Computer Society. ISBN 0-7695-2132-0.

[P6] Gerard K. Rauwerda, Paul M. Heysters, and Gerard J.M. Smit. Mapping Wireless Communication Algorithms onto a Reconfigurable Architecture. *Journal of Supercomputing*, 30(3):263 – 282, December 2004. ISSN 0920-8542.

[P7] Lodewijk T. Smit, Gerard J.M. Smit, Johann L. Hurink, and Gerard K. Rauwerda. BER Estimation for HiperLAN/2. In *Lecture Notes in Computer Science: Personal Wireless Communications: IFIP TC6 9th International Conference*, volume 3260, pages 164 – 179, Delft, The Netherlands, September 2004. Springer-Verlag. ISBN 3-540-23162-5, ISSN 0302-9743.

[P8] Gerard K. Rauwerda, Paul M. Heysters, and Gerard J.M. Smit. An OFDM Receiver Implemented on the Coarse-grain Reconfigurable Montium Processor. In *Proceedings of the 9th International OFDM Workshop (InOWo'04)*, pages 197 – 201, Dresden, Germany, September 15–16, 2004.

[P9] Gerard K. Rauwerda and Gerard J.M. Smit. Implementation of a Flexible RAKE Receiver in Heterogeneous Reconfigurable Hardware. In *Proceedings of the 2004 IEEE International Conference on Field-Programmable Technology*, pages 437 – 440, Brisbane, Australia, December 6–8, 2004. ISBN 0-7803-8651-5, ISBN 0-7803-8652-3.

[P10] Gerard K. Rauwerda and Gerard J.M. Smit. Software Defined Radio and Heterogeneous Reconfigurable Hardware. In *Proceedings of the 15th ProRISC workshop*, pages 125 – 132, Veldhoven, The Netherlands, November 25–26, 2004. ISBN 90-73461-43-X.

[P11] Pascal T. Wolkotte, Gerard J.M. Smit, Gerard K. Rauwerda, and Lodewijk T. Smit. An Energy-Efficient Reconfigurable Circuit Switched Network-on-Chip. In *Proceedings of the 12th Reconfigurable Architectures Workshop (RAW 2005)*, Denver, Colorado, USA, April 4–5, 2005. IEEE Computer Society. ISBN 0-7695-2312-9.

[P12] Gerard K. Rauwerda, Gerard J.M. Smit, and Werner Brugger. Implementing an Adaptive Viterbi Algorithm in Coarse-Grained Reconfigurable Hardware. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'05)*, pages 62 – 68, Las Vegas, Nevada, USA, June 27–30, 2005. CSREA Press. ISBN 1-932415-74-2.

[P13] Gerard J.M. Smit and Gerard K. Rauwerda. Reconfigurable Architectures for Adaptable Mobile Systems. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'05)*, pages 17 – 25, Las Vegas, Nevada, USA, June 27–30, 2005. CSREA Press. ISBN 1-932415-74-2.

[P14] Gerard K. Rauwerda, Gerard J.M. Smit, and Paul M. Heysters. Implementation of Multi-standard Wireless Communication Receivers in a Heterogeneous Reconfigurable System-on-Chip. In *Proceedings of the 16th ProRISC workshop*, pages 421 – 427, Veldhoven, The Netherlands, November 17–18, 2005. ISBN 90-73461-50-2.

[P15] Gerard K. Rauwerda, Gerard J.M. Smit, Casper R.W. van Benthem, and Paul M. Heysters. Reconfigurable Turbo/Viterbi Channel Decoder in the Coarse-Grained Montium Architecture. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'06)*, pages 110–116, Las Vegas, Nevada, USA, June 26–29, 2006. CSREA Press. ISBN 1-60132-011-6.

[P16] Marcel D. van de Burgwal, Gerard J.M. Smit, Gerard K. Rauwerda, and Paul M. Heysters. Hydra: an Energy-efficient and Reconfigurable Network Interface. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'06)*, pages 171–177, Las Vegas, Nevada, USA, June 26–29, 2006. CSREA Press. ISBN 1-60132-011-6.

[P17] Arjan C. Dam, Michel G.J. Lammertink, Kenneth C. Rovers, Johan Slagman, Arno M. Wellink, Gerard K. Rauwerda, and Gerard J.M. Smit. Hardware/Software Co-design Applied to Reed-Solomon Decoding for the DMB Standard. In *Proceedings of the 9th EUROMICRO Conference on DIGITAL SYSTEM DESIGN Architectures, Methods and Tools (DSD 2006)*, pages 447–455, Dubrovnik, Croatia, August 30 – September 1, 2006. IEEE Computer Society. ISBN 0-7695-2609-8.

[P18] Gerard K. Rauwerda, Jordy Potman, Fokke W. Hoeksema, and Gerard J.M. Smit. Adaptation in the Physical Layer Using Heterogeneous Reconfigurable Hardware. In *Adaptation Techniques in Wireless Multimedia Networks*, volume 6 of *Wireless Networks and Mobile Computing*. Nova Science Publishers, USA, 2007. ISBN 1-59454-883-8.

[P19] L.T. Smit, G.K. Rauwerda, A. Molderink, P.T. Wolkotte, and G.J.M. Smit. Implementation of a 2-D 8x8 IDCT on the Reconfigurable Montium Core. In *Proceedings of the 2007 International Conference on Field Programmable Logic and Applications*, pages 562–566, Amsterdam, The Netherlands, August 27–29, 2007. IEEE Computer Society. ISBN 1-4244-1060-6.

[P20] Gerard K. Rauwerda, Paul M. Heysters, and Gerard J.M. Smit. Towards Software Defined Radios using Coarse-Grained Reconfigurable Hardware. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(1), January 2008. ISSN 1063-8210.

Appendix A

Power estimation Rake receiver

Power estimations have been performed on the MONTIUM *Tile Processor (TP). The power consumption of the* MONTIUM *TP when configured as Rake receiver has been analyzed in this appendix.*

A.1 Power estimation

Power estimations have been performed on the MONTIUM Tile Processor (TP) netlist for 4-finger Rake processing. A netlist of the MONTIUM TP[1] has been generated in TSMC 130 nm technology using the *TCB013LVHP* library. The MONTIUM TP has been synthesized using SYNOPSYS DESIGN COMPILER [106]. The performance simulations, that are described in Chapter 5, are used to obtain estimations of the dynamic and static power consumption. The data of the performance simulations have been analyzed by SYNOPSYS PRIMEPOWER in order to come up with estimations of the power consumption [106].

The power consumption of the memories is not taken into account during power analysis. The MONTIUM TP has been synthesized, but no proper memory cells were available to integrate in the netlist. The memories in the MONTIUM TP were considered as *black boxes* during synthesis. The power simulations have been performed with behavioural memory models.

A.2 Average power consumption

Table A.1 shows the average power consumption[2] of the MONTIUM TP during Rake processing with 4 fingers running at a clock frequency of 25 MHz. In Table A.2 the average power consumption[2] of the MONTIUM TP is given for the same algorithm, but running at a clock frequency of 50 MHz. The results of the power consumption of the MONTIUM TP are grouped for the different modules that can be identified in the MONTIUM TP. Furthermore, the total power consumption of the MONTIUM TP has been divided in *static* and *dynamic* power. The static power consumption of the MONTIUM TP is typically independent of the signal activity inside the hardware. Only the dynamic power consumption depends on the typical signal activity of the algorithm and the clock frequency of the MONTIUM TP.

Figure A.1 depicts the contribution of the different modules in the MONTIUM TP to the average total power consumption. Figure A.2 and A.3 show the average dynamic and static power consumption, respectively.

From Table A.1 and Table A.2 can be concluded that the static power consumption is independent of the clock frequency. Furthermore, the typical signal activity inside the MONTIUM TP does not influence the static power consumption when comparing Figure A.3 and Figure B.3. The static power consumption shows similar behaviour independent of the performed DSP algorithm (e.g. Rake receiver or Viterbi decoder). This observation confirms that similar MONTIUM netlists are used during power consumption estimation of the different algorithms.

[1] MONTIUM *version 01.00.00* has been used for power estimations and performance simulations. The design of MONTIUM *version 01.00.00* is identical to the MONTIUM described in [54].

[2] The power consumption is estimated during configuration of the MONTIUM TP and during the actual processing of the algorithm. The average power consumption during configuration and processing has been depicted in Table A.1 and A.2.

Table A.1: *Power estimation results of the* MONTIUM *TP running at 25 MHz during 4-finger Rake processing.*

	Total power $[\mu W]$	Dynamic power $[\mu W]$	Static power $[\mu W]$
Processing Part Array	6 860	5 568	1 292
Sequencer	33.81	27.19	6.617
ALU decoder	204.0	142.2	61.77
Memory decoder	478.9	327.2	151.7
Register decoder	482.7	330.9	151.8
Crossbar decoder	1 452	999.1	452.9
Tile Processor	10 420	8 304	2 117

Table A.2: *Power estimation results of the* MONTIUM *TP running at 50 MHz during 4-finger Rake processing.*

	Total power $[\mu W]$	Dynamic power $[\mu W]$	Static power $[\mu W]$
Processing Part Array	12 370	11 079	1 291
Sequencer	61.10	54.48	6.617
ALU decoder	346.8	285.0	61.77
Memory decoder	806.4	654.6	151.8
Register decoder	813.7	661.9	151.8
Crossbar decoder	2 453	2 000	452.8
Tile Processor	18 685	16 569	2 116

The average normalized dynamic power consumption of the MONTIUM TP during Rake processing with 4 fingers is estimated to be 332 $\mu W/MHz$ (without considering the power consumption of the SRAM modules). The total dynamic power that is consumed by the SRAM modules in the MONTIUM TP[3] is estimated to be $7 \cdot 14 + 40 = 138\ \mu W/MHz$ [54]. Hence, the average normalized dynamic power consumption of the MONTIUM TP (including the contribution of the SRAMs) becomes $\sim 470\ \mu W/MHz$.

[3] The dynamic power consumption of one local memory in the MONTIUM Processing Part is 14 $\mu W/MHz$ according to [54, Appendix B]. Moreover, the dynamic power consumption of the sequencer memory is estimated to be 40 $\mu W/MHz$, which is a conservative estimation according to [54].

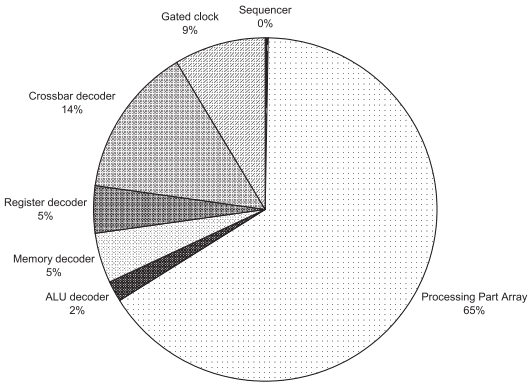

Figure A.1: *Average total power consumption of the* MONTIUM *TP during 4-finger Rake processing.*

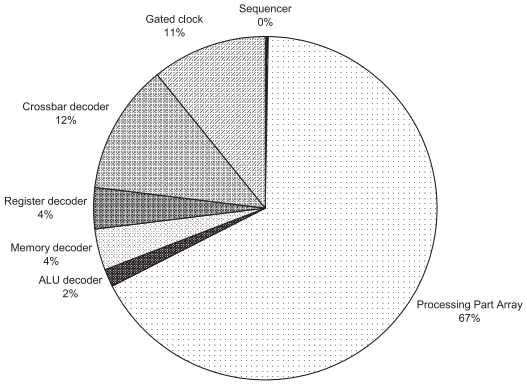

Figure A.2: *Average dynamic power consumption of the* MONTIUM *TP during 4-finger Rake processing.*

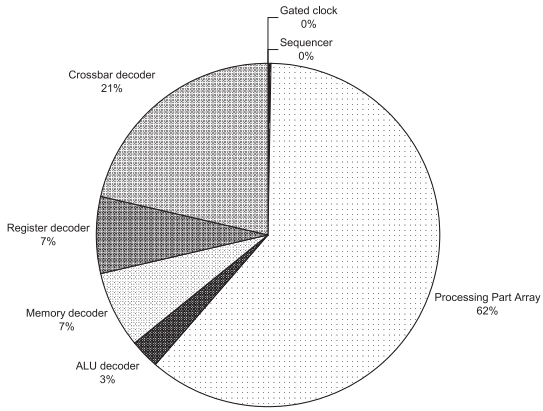

Figure A.3: *Average static power consumption of the* MONTIUM *TP during 4-finger Rake processing.*

A.3 Detailed power consumption

In Section A.2 the average power consumption of the MONTIUM TP has been depicted. In this section the total power consumption of the different modules in the MONTIUM TP is shown as a function of time. The figures are included to show that the power consumption of the MONTIUM TP is not constant over time, but different phases can be identified clearly. For visibility reasons, the results have been filtered by a *moving average* filter. Hence, the results in the following figures only show the average power consumption over time.

The configuration phase of the MONTIUM TP, which is done during the first 17.8 μs, is clearly visible in the presented graphs. After configuration, the configuration clock in the MONTIUM TP is switched off by the *clock gating* module in the MONTIUM TP. Especially in the power consumption results of the MONTIUM decoders the effect of clock gating is clearly visible. After configuration of the MONTIUM TP, the MONTIUM TP performs Rake signal processing during *running* mode.

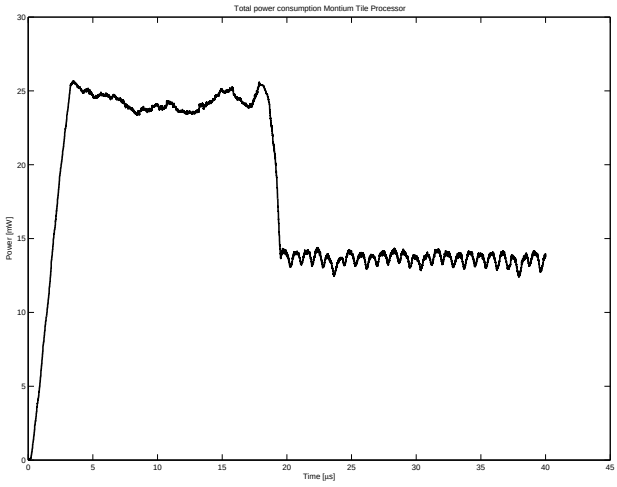

Figure A.4: *Total power consumption of the* MONTIUM *TP during 4-finger Rake processing.*

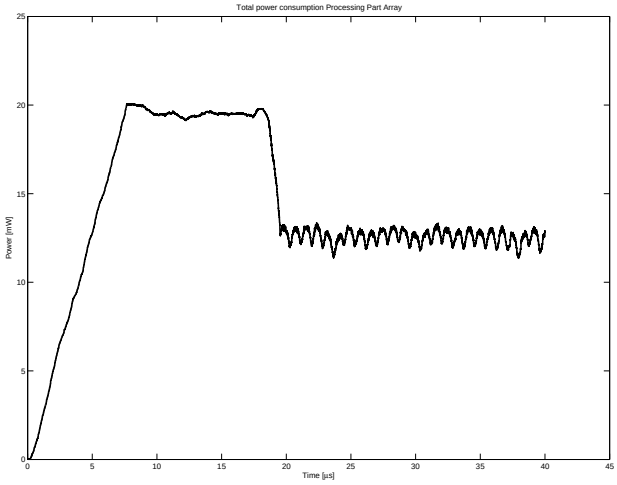

Figure A.5: *Total power consumption of the Processing Part Array during 4-finger Rake processing.*

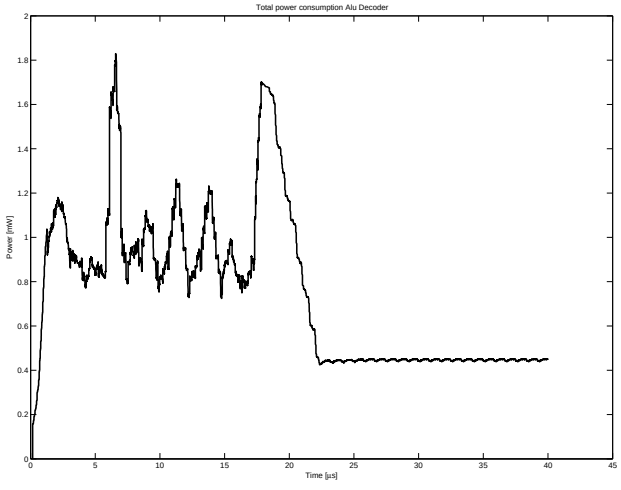

Figure A.6: *Total power consumption of the ALU Decoder during 4-finger Rake processing.*

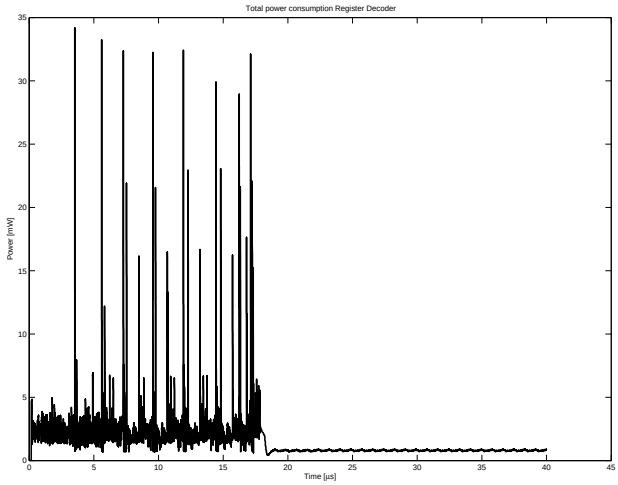

Figure A.7: *Total power consumption of the Register Decoder during 4-finger Rake processing.*

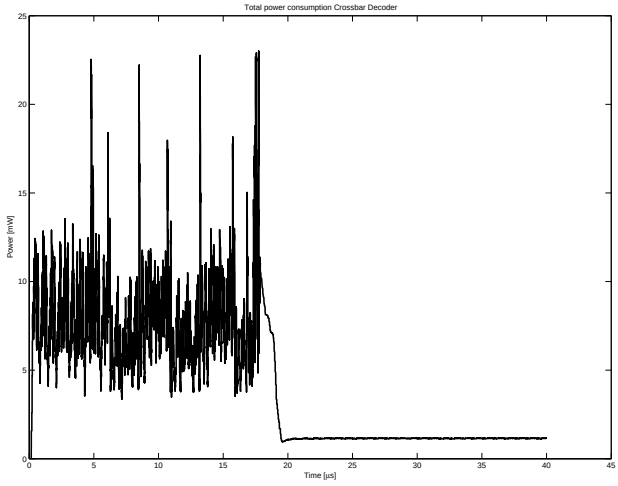

Figure A.8: *Total power consumption of the Crossbar Decoder during 4-finger Rake processing.*

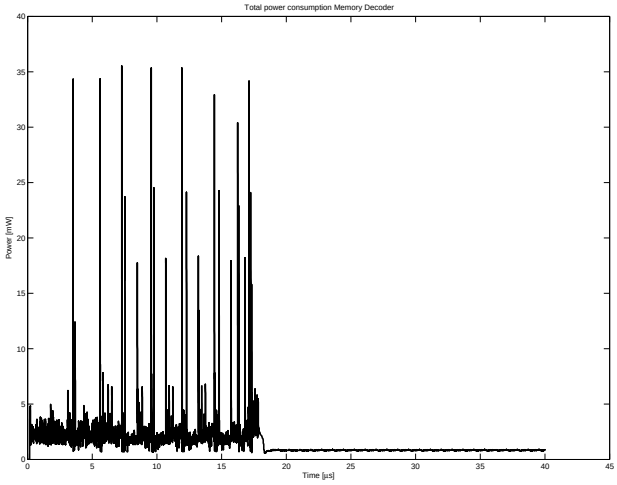

Figure A.9: *Total power consumption of the Memory Decoder during 4-finger Rake processing.*

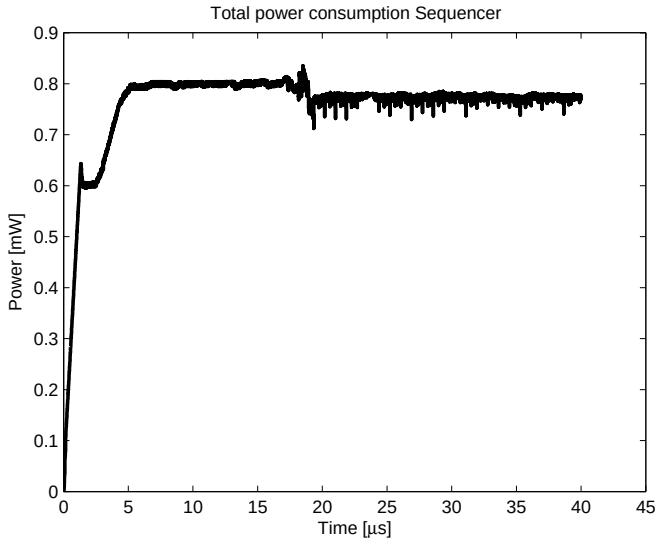

Figure A.10: *Total power consumption of the Sequencer during 4-finger Rake processing.*

Appendix B

Power estimation Viterbi decoder

Power estimations have been performed on the MONTIUM *Tile Processor (TP). The power consumption of the* MONTIUM *TP when configured as Viterbi decoder has been analyzed in this appendix.*

B.1 Power estimation

Power estimations have been performed on the MONTIUM Tile Processor (TP) netlist during Viterbi signal processing. A netlist of the MONTIUM TP[1] which is modified with ACS hardware support, as recommended in Chapter 6, has been generated in TSMC 130 nm technology using the *TCB013LVHP* library. The MONTIUM TP has been synthesized using SYNOPSYS DESIGN COMPILER [106]. The Viterbi performance simulations, that are described in Chapter 6, are used to obtain estimations of the dynamic and static power consumption. The data of the performance simulations have been analyzed by SYNOPSYS PRIMEPOWER in order to come up with estimations of the power consumption [106].

The power consumption of the memories is not taken into account during power analysis. The MONTIUM TP has been synthesized, but no proper memory cells were available to integrate in the netlist. The memories in the MONTIUM TP were considered as *black boxes* during synthesis. The power simulations have been performed with behavioural memory models.

B.2 Average power consumption

Table B.1 shows the average power consumption[2] of the MONTIUM TP during Viterbi processing running at a clock frequency of 25 MHz. In Table B.2 the average power consumption[2] of the MONTIUM TP is given for the same algorithm, but running at a clock frequency of 50 MHz. The results of the power consumption of the MONTIUM TP are grouped for the different modules that can be identified in the MONTIUM TP. Furthermore, the total power consumption of the MONTIUM TP has been divided in *static* and *dynamic* power. The static power consumption of the MONTIUM TP is typically independent of the signal activity inside the hardware. Only the dynamic power consumption depends on the typical signal activity of the algorithm and the clock frequency of the MONTIUM TP.

Figure B.1 depicts the contribution of the different modules in the MONTIUM TP to the average total power consumption. Figure B.2 and B.3 show the average dynamic and static power consumption, respectively.

From Table B.1 and Table B.2 can be concluded that the static power consumption is independent of the clock frequency. Furthermore, the typical signal activity inside the MONTIUM TP does not influence the static power consumption when comparing Figure B.3 and Figure A.3. The static power consumption shows similar behaviour independent of the performed DSP algorithm (e.g. Rake receiver or Viterbi decoder). This

[1] MONTIUM *version 01.00.00* has been modified with Add Compare Select (ACS) hardware support and is used for power estimations and performance simulations. The design of MONTIUM *version 01.00.00* is identical to the MONTIUM described in [54].

[2] The power consumption is estimated during configuration of the MONTIUM TP and during the actual processing of the algorithm. The average power consumption during configuration and processing has been depicted in Table B.1 and B.2.

Table B.1: *Power estimation results of the* MONTIUM *TP running at 25 MHz during Viterbi processing.*

	Total power $[\mu W]$	Dynamic power $[\mu W]$	Static power $[\mu W]$
Processing Part Array	4 268	2 977	1 291
Sequencer	31.31	24.84	6.469
ALU decoder	109.1	46.91	62.19
Memory decoder	257.4	114.4	143.0
Register decoder	261.3	116.3	145.0
Crossbar decoder	770.8	333.1	437.7
Tile Processor	6 009	3 924	2 085

Table B.2: *Power estimation results of the* MONTIUM *TP running at 50 MHz during Viterbi processing.*

	Total power $[\mu W]$	Dynamic power $[\mu W]$	Static power $[\mu W]$
Processing Part Array	7 245	5 955	1 290
Sequencer	56.15	49.68	6.468
ALU decoder	156.1	93.94	62.16
Memory decoder	371.8	228.8	143.0
Register decoder	377.6	232.6	145.0
Crossbar decoder	1 104	666.4	437.6
Tile Processor	9 937	7 853	2 085

observation confirms that similar MONTIUM netlists are used during power consumption estimation of the different algorithms.

The average normalized dynamic power consumption of the MONTIUM TP during Viterbi processing is estimated to be 157 $\mu W/MHz$ (without considering the power consumption of the SRAM modules). The total dynamic power that is consumed by the SRAM modules in the MONTIUM TP[3] is estimated to be $8{\cdot}14{+}40 = 152\,\mu W/MHz$ [54]. Hence, the average normalized dynamic power consumption of the MONTIUM TP (including the contribution of the SRAMs) becomes $\sim 309\ \mu W/MHz$.

[3] The dynamic power consumption of one local memory in the MONTIUM Processing Part is 14 $\mu W/MHz$ according to [54, Appendix B]. Moreover, the dynamic power consumption of the sequencer memory is estimated to be 40 $\mu W/MHz$, which is a conservative estimation according to [54].

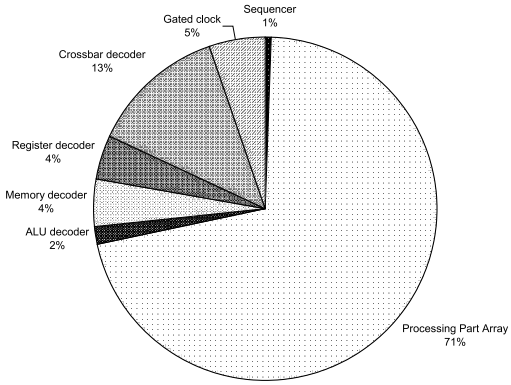

Figure B.1: *Average total power consumption of the* MONTIUM *TP during Viterbi processing.*

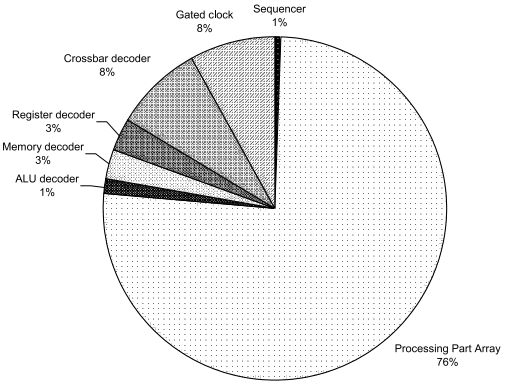

Figure B.2: *Average dynamic power consumption of the* MONTIUM *TP during Viterbi processing.*

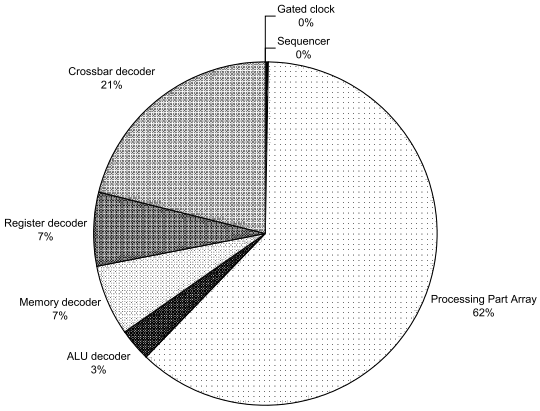

Figure B.3: *Average static power consumption of the* MONTIUM *TP during Viterbi processing.*

B.3 Detailed power consumption

In Section B.2 the average power consumption of the MONTIUM TP has been depicted. In this section the total power consumption of the different modules in the MONTIUM TP is shown as a function of time. The figures are included to show that the power consumption of the MONTIUM TP is not constant over time, but different phases can be identified clearly. For visibility reasons, the results have been filtered by a *moving average* filter. Hence, the results in the following figures only show the average power consumption over time.

The configuration phase of the MONTIUM TP, which is done during the first 27.3 μs, is clearly visible in the presented graphs. After configuration, the configuration clock in the MONTIUM TP is switched off by the *clock gating* module in the MONTIUM TP. Especially in the power consumption results of the MONTIUM decoders the effect of clock gating is clearly visible. After configuration the local memories are initialized with data. The memory initialization is performed from 27.3 μs till 47.8 μs. After initialization, the MONTIUM TP performs Viterbi signal processing during *running* mode.

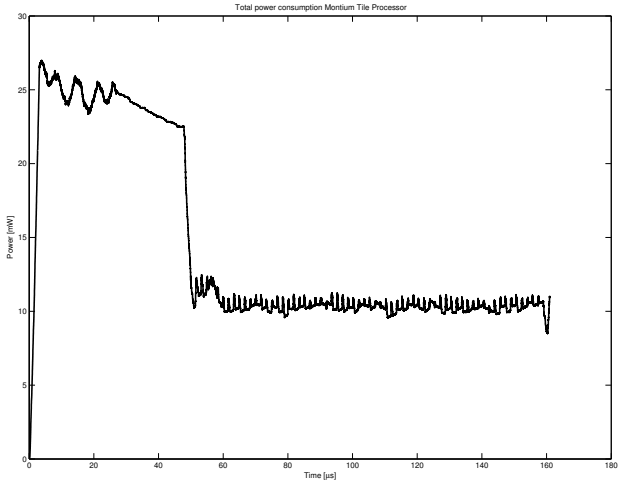

Figure B.4: *Total power consumption of the* MONTIUM *TP during Viterbi processing.*

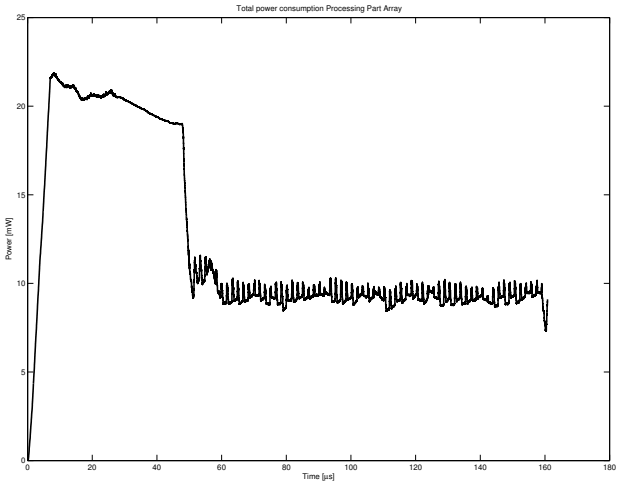

Figure B.5: *Total power consumption of the Processing Part Array during Viterbi processing.*

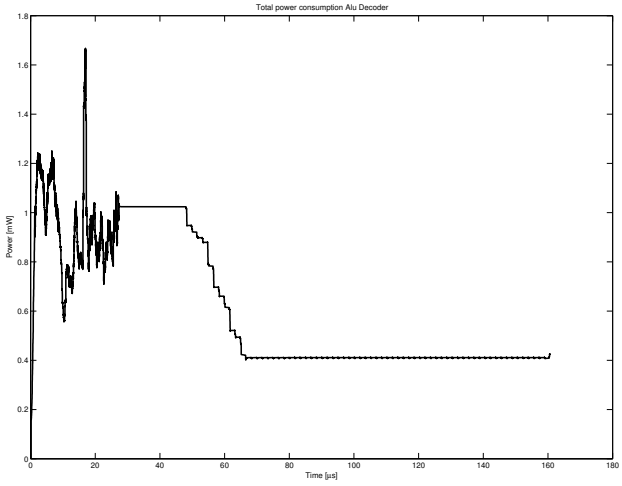

Figure B.6: *Total power consumption of the ALU Decoder during Viterbi processing.*

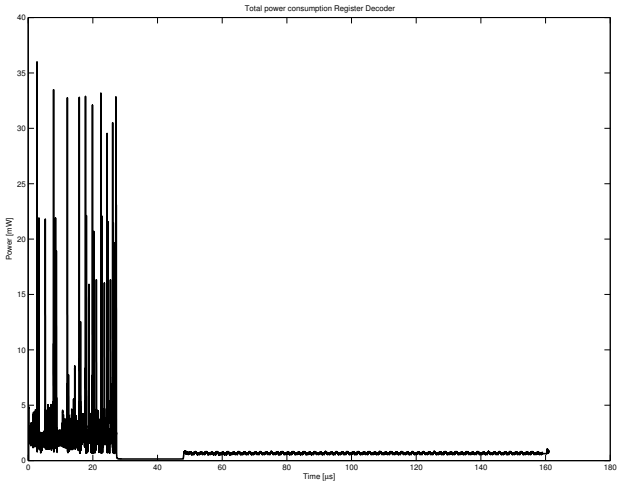

Figure B.7: *Total power consumption of the Register Decoder during Viterbi processing.*

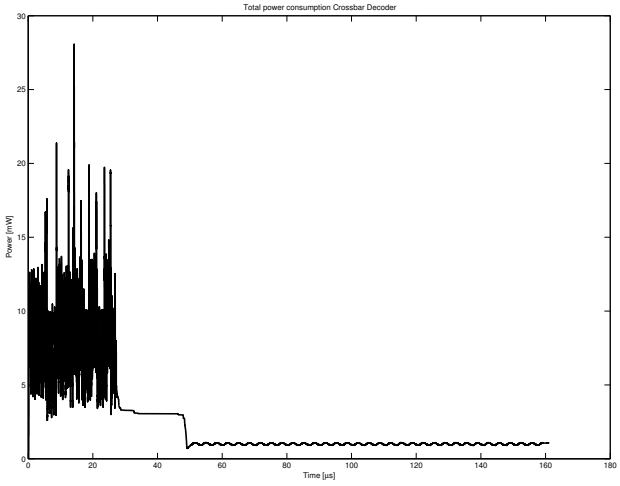

Figure B.8: *Total power consumption of the Crossbar Decoder during Viterbi processing.*

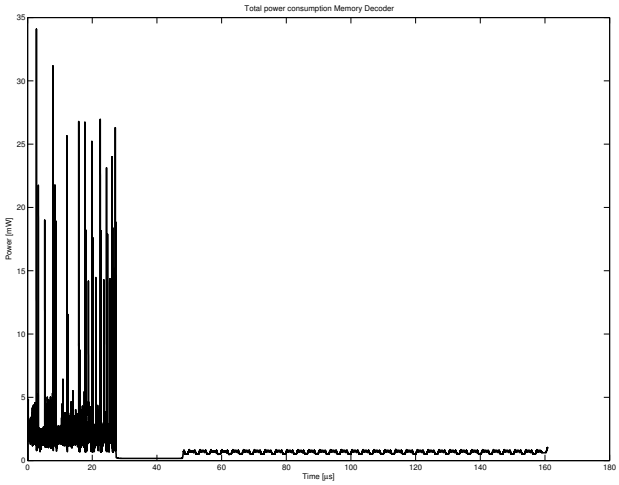

Figure B.9: *Total power consumption of the Memory Decoder during Viterbi processing.*

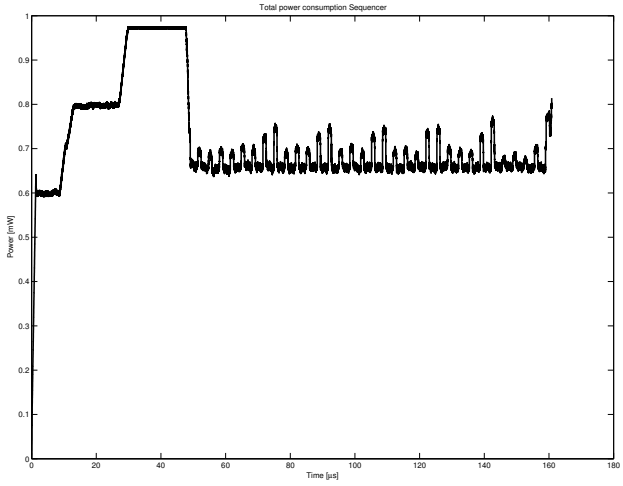

Figure B.10: *Total power consumption of the Sequencer during Viterbi processing.*