Linköping Studies in Science and Technology
Dissertation No. 969

Design of Programmable Baseband Processors

Eric Tell

Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Linköping 2005

ISBN 91-85457-20-5
ISSN 0345-7524

Design of Programmable
Baseband Processors
Eric Tell
ISBN 91-85457-20-5

Linköping Studies in Science and Technology
Dissertation No. 969
ISSN 0345-7524

Department of Electrical Engineering
Linköping University
SE-581 83 Linköping
Sweden
Phone: +46 13 28 10 00

Author e-mail: eric.tell@ieee.org

Cover image
Visualization of the BBP1 architecture. Data memories and accelerator blocks are connected to the central DSP core via a configurable network. The image has been generated using POV-Ray 3.5.

Printed by UniTryck, Linköping University
Linköping, Sweden, 2005

Abstract

The world of wireless communications is under constant change. Radio standards evolve and new standards emerge. More and more functionality is put into wireless terminals. E.g. mobile phones need to handle both second and third generation mobile telephony as well as Bluetooth, and will soon also support wireless LAN functionality, reception of digital audio and video broadcasting, etc.

These developments have lead to an increased interest in software defined radio (SDR), i.e. radio devices that can be reconfigured via software. SDR would provide benefits such as low cost for multi-mode devices, reuse of the same hardware in different products, and increased product life time via software updates.

One essential part of any software defined radio is a programmable baseband processor that is flexible enough to handle different types of modulation, different channel coding schemes, and different trade-offs between data rate and mobility.

So far, programmable baseband solutions have mostly been used in high end systems such as mobile telephony base stations since the cost and power consumption have been considered too high for handheld terminals.

In this work a new low power and low silicon area programmable baseband processor architecture aimed for multi-mode terminals is presented. The architecture is based on a customized DSP core and a number of hardware accelerators connected via a configurable network. The architecture offers a good tradeoff between flexibility and performance

through an optimized instruction set, efficient hardware acceleration of carefully selected functions, low memory cost, and low control overhead.

One main contribution of this work is a study of important issues in programmable baseband processing such as software-hardware partitioning, instruction level acceleration, low power design, and memory issues. Further contributions are a unique optimized instruction set architecture, a unique architecture for efficient integration of hardware accelerators in the processor, and mapping of complete baseband applications to the presented architecture.

The architecture has been proven in a manufactured demonstrator chip for WLAN applications. WLAN firmware has been developed and run on the chip at full speed. Silicon area and measured power consumption have proven to be similar to that of a non-programmable ASIC solution.

Preface

The contents of this thesis present the research that I have done during the last 4 years. Various parts of the presented architecture have been described in the following conference papers:

- **Eric Tell**, Anders Nilsson, and Dake Liu, "A Low Power and Low Area Programmable Baseband Processor Architecture", in *Proceedings of the 5th International Workshop on System-on-chip for Real-Time Applications (IWSOC)*, Banff, Canada, July 2005
- **Eric Tell**, Anders Nilsson, and Dake Liu, "A Programmable DSP Core for Baseband Processing", in *Proceedings of the IEEE Northeast Workshop on Circuits and Systems (NEWCAS)*, Quebec City, Canada, June 2005
- **Eric Tell**, Anders Nilsson, and Dake Liu, "Implementation of a Programmable Baseband Processor", in *Proceedings of Radiovetenskap och Kommunikation (RVK)*, Linköping, Sweden, June 2005
- **Eric Tell** and Dake Liu, "A hardware architecture for a multi-mode block interleaver", in *Proceedings of the International Conference on circuits and systems for communications (ICCSC)*, Moscow, Russia, July 2004
- **Eric Tell**, Mikael Olausson, and Dake Liu, "A General DSP processor at the cost of 23k gates and 1/2 a man-year design time", in *Proceedings of the International Conference on Acoustics, Signal and Speech Processing (ICASSP)*, Hong Kong, April 2003

- Anders Nilsson, **Eric Tell**, and Dake Liu, "An accelerator architecture for programmable multi-standard baseband processors", in *Proceedings of the International Conference on Wireless Networks and Emerging Technologies (WNET)*, Banff, Canada, July 2004

Some early parts of my work are covered in the following book chapter:

- Dake Liu and **Eric Tell**, "Chapter 23 - Low Power Programmable Baseband Processors", in *Low Power Electronics Design*, editor Christian Piguet, CRC Press, July 2004

The following draft journal manuscript is a work in progress:

- **Eric Tell**, Anders Nilsson, and Dake Liu, "Low Area and Low Power Programmable Baseband Processing", to be sumitted

The following papers are also more or less related to the presented work:

- **Eric Tell**, Olle Seger, and Dake Liu, "A converged hardware solution for FFT, DCT, and Walsh transform", in *Proceedings of the International Symposium on Signal Processing and its Applications (ISSPA)*, Paris, France, July 2003

- **Eric Tell** and Dake Liu, "A Suitable Channel Equalization Scheme for IEEE 802.11b", in *Proceedings of the Swedish system-on-chip conference (SSoCC)*, Eskilstuna, Sweden, April 2003

- Anders Nilsson, **Eric Tell**, and Dake Liu, "Design Methodology for memory-efficient multi-standard baseband processors", in *Proceedings of the Asia-Pacific Communications Conference (APCC)*, Perth, Australia, October 2005

- Anders Nilsson, **Eric Tell**, and Dake Liu, "A Programmable SIMD-based Multi-standard Rake-receiver Architecture", in *Proceedings of European Signal Processing Conference (EUSIPCO)*, Antalya, Turkey, August 2005

- Anders Nilsson, **Eric Tell**, and Dake Liu, "A fully programmable rake-receiver architecture for multi-standard baseband processing", in *Proceedings of the International Conference on Networks and Communication Systems (NCS)*, Krabi, Thailand, May 2005

- Dake Liu, **Eric Tell**, Anders Nilsson, and Ingemar Söderquist, "Fully flexible baseband DSP processors for future SDR/JTRS", in *Proceedings of Western European Armaments Organization CEPA2 Workshop*, Brussels, Belgium, March 2005

- Dake Liu, **Eric Tell**, and Anders Nilsson, "Implementation of Programmable Baseband Processors", in *Proceedings of CCIC*, Hangzhou, China, November 2004

- Haiyan Jiao, Anders Nilsson, **Eric Tell**, and Dake Liu, "MIPS cost estimation for OFDM-VBLAST systems", submitted conference paper

Acknowledgments

There are many people who deserve my gratitude for their contributions to my research and for making my time as a PhD student so enjoyable. I would especially like to thank the following:

- My supervisor professor Dake Liu, for accepting me as his PhD student, for coming up with the idea for this project, and for support and encouragement.

- My closest cooperator Lic. Eng. Anders Nilsson, for many fruitful discussions over the last two years as well as more concrete help with coding, baseband algorithm development, and PCB design.

- Dr. Olle Seger for sharing his expertise in signal processing.

- Dr. Anders Edman for interesting discussions and ideas.

- My former and current fellow PhD students at Computer Engineering: Dr. Daniel Wiklund, Dr. Thomas Henriksson, Dr. Ulf Nordqvist, Lic. Eng. Mikael Olausson, Andreas Ehliar, Johan Eilert, Per Karlsson, and Di Wu. Special thanks to Daniel for proofreading my dissertation.

- Anders Edqvist for a good job on the C-compiler for the BBP1 processor.

- ICC Shanghai for help with chip back-end work and fabrication.

- Ylva Jernling for making all administrative tasks so simple and Anders Nilsson (Sr) for technical support.
- The rest of the Computer Engineering group, for providing a pleasant working environment.
- And finally, my parents for always supporting me and Lena for inspiration as well as much needed distraction from work.

This research was funded by the Socware program.

Eric Tell
Linköping, August 2005

Contents

List of Figures

List of Tables

Abbreviations

3G	Third generation mobile telecommunication
ADC	Analog to digital converter *or* conversion
AFC	Automatic frequency control
AGC	Automatic gain control
ASIC	Application specific integrated circuit
BPSK	Binary phase shift keying
CCK	Complementary code keying
CDMA	Code division multiple access
DAB	Digital audio broadcasting
DAC	Digital to analog converter *or* conversion
DBPSK	Differential binary phase shift keying
DQPSK	Differential quadrature phase shift keying
DSP	Digital signal processor *or* processing
DSSS	Direct sequence spread spectrum
DVB	Digital video broadcasting
FIR	Finite impulse response
FPGA	Field programmable gate array
GPS	Global positioning system
HDL	Hardware description language
IDE	Integrated design environment
IP	Intellectual Property
ISA	Instruction set architecture
ISI	Inter-symbol interference

ISS	Instruction set simulator
JTC	Joint Technical Committee
GUI	Graphical user interface
MAC	Multiply-accumulate *or* Media access control
MIMO	Multiple input multiple output
MIPS	Million instructions per second
modem	Modulator-demodulator
OFDM	Orthogonal frequency division multiplexing
PC	Program counter
PHY	Physical layer
QAM	Quadrature amplitude modulation
QPSK	Quadrature phase shift keying
RISC	Reduced instruction set computer
RTL	Register transfer level
SDR	Software defined radio
SDRF	Software defined radio forum
SIFS	Short intra-frame spacing
SIMD	Single instruction multiple data
SNR	Signal-to-noise ratio
UMTS	Universal mobile telephony system
VLIW	Very long instruction word
WCDMA	Wideband CDMA
WLAN	Wireless local area network

Part I

Background

Chapter 1
Introduction

1.1 Background

The evolution of wireless devices and systems is not slowing down. New standards keep coming up in cellular telephony, wireless computer networks, and audio/video broadcasting. Another trend is convergence of wireless devices. In other words, more and more functions are put into wireless terminals. A device such as a mobile phone has to handle multiple radio standards: both second and third generation mobile telephony as well as Bluetooth are becoming mandatory. Many devices will in addition support networking standards such as WiFi and WiMax, reception of broadcast radio and television, and possibly GPS.

This has lead to an increased interest in the concept of Software Defined Radio (SDR). The idea is that it should be possible to alter the functionality of a radio device at run-time by simply replacing its software. This means that the same hardware should be able to handle many different frequency bands and modulation types as well as different demands on data rate and mobility. This would make it possible for manufacturers to reuse the same hardware for different products and ultimately for users to connect to any system that happens to be available at any given time and place.

One important component in any modern radio device is the modem

(modulator/demodulator) or *baseband processor* which handles the heavy digital signal processing required in the physical layer (PHY) of the radio system. In order to realize SDR, a programmable baseband processor is needed.

Although SDR has been discussed for many years, it is still only used to some extent in large, high-cost components such as mobile telephony base stations and military systems. In these systems, the SDR functionality is typically handled by standard DSP processors or FPGA-type solutions.

One reason why SDR has not yet had a breakthrough is that it has been considered to expensive and power consuming for handheld, battery powered devices. In order to achieve the low cost and power consumption required for such devices, the baseband processor must be optimized for the types of operations that are needed in baseband processing.

Existing processors for SDR tend to be based on traditional architectures where the required computing capacity is reached by e.g. traditional VLIW and SIMD solutions. This usually leads to solutions which are too area and power consuming for many applications. New architectures are needed.

With this background, a research project on programmable baseband processing was started at the Computer Engineering group at Linköping University in the spring of 2002. This thesis is one result of that project.

1.2 Scope of the Thesis

The goal of the research project has been to find new efficient architectures for programmable baseband processing. The focus has been on wireless terminals, implying that features such as low cost and low power consumption are essential.

The central part of the thesis is the presentation of the baseband processor architecture which is the main result of the work in this project. It also describes the implementation of a demonstrator chip, mainly focused on Wireless LAN (WLAN) applications, and the implementation of

WLAN standards on this chip.

This thesis also includes an introduction to design of application specific processors and an overview of baseband processing tasks and challenges. General ideas for design of efficient programmable baseband processors are also discussed.

A discussion of the design approach as well as experiences and conclusions made during the design work is also presented. Some parts regarding e.g. acceleration techniques and low-power issues are not baseband specific but valid for application specific processors in general.

1.3 Contributions

The main contributions of the presented work can be summarized in the following three points:

- Review of important issues in programmable baseband processing such as SW-HW partitioning, instruction level acceleration, memory issues, and low power design, and their impact in this specific case.
- Development and demonstration of an area and power efficient architecture for baseband processing . This includes design of an instruction set including a new type of instructions operating on vectors of complex numbers and a scheme for integrating hardware acceleration blocks with the programmable DSP core. The feasibility of the architecture as well as low silicon area and power consumption has been proven by measurements on a manufactured demonstrator chip.
- Mapping of baseband applications to the proposed architecture. The WLAN standards IEEE 802.11a and IEEE 802.11b have been scheduled onto the processor. Firmware has been developed and demonstrated on the manufactured chip, proving the correctness of the hardware and that the programmable architecture can meet the requirements and challenges in radio baseband processing.

Although not presented in this thesis, significant work has also been put into investigating and evaluating various baseband processing algorithms from the perspectives of performance and suitability for software implementation.

1.4 Organization of the Thesis

The thesis is divided into four parts. The first part contains the introduction and a short background on SDR.

The second part of the thesis contains an introduction to design of application specific processors in chapter 3 and an introduction to baseband processing in chapter 4. It also provides a short general discussion about issues in programmable baseband processors in chapter 5 and a brief overview of existing solutions in chapter 6.

The third part is the core of the thesis. Chapter 7 presents the proposed processor architecture in general and chapter 8 describes the implementation of a demonstrator chip as well as firmware design and design tools.

The last part contains conclusions in chapter 9 and directions and ideas for future work in chapter 10.

Some additional details regarding application profiling, benchmarking, hardware mapping, and scheduling for WLAN standards are found in appendices.

Chapter 2
Multi-standard Radio Systems and Software Defined Radio

2.1 Trends in Wireless Systems and Devices

The constant change in the world of wireless systems and the increasing convergence of wireless devices was touched upon already in the introduction of this thesis. Wireless devices like mobile phones (or their future equivalent) will need to handle an increasing amount of different functions. The processing requirements of modern standards are increasing due to increased bandwidth in combination with high mobility. At the same time power consumption continues to be very important for battery powered devices.

The amount of upcoming standards and the constant demand for new features and short development time increase the benefits of programmable baseband processors since the same hardware can potentially be used in many different products and product generations. Adding to this is the fact that it usually takes a long time for a standard to become stable and manufacturers have to start product development long before the

standard is finally fixed.

Programmable baseband is so far used in e.g. base station applications but the cost and power consumption have been considered too high for handheld terminals. Terminals would otherwise have the greater benefit of such technology due to multi-mode capability and the constant demand for new products and features.

2.2 Software Defined Radio

Due to the described developments there has been an increasing interest in SDR over the last years. Although the concept has been around for a considerable time, only recently circuit technologies (RF, analog and mixed signal as well as digital) have reached a point where the goal is within reach.

The following description of SDR is given by the Software Defined Radio Forum (SDRF) [1]:

> SDR is a collection of hardware and software technologies that enable reconfigurable system architectures for wireless networks and user terminals. SDR provides an efficient and comparatively inexpensive solution to the problem of building multimode, multiband, multifunctional wireless devices that can be adapted, updated, or enhanced by using software upgrades. As such, SDR can be considered an enabling technology that is applicable across a wide range of areas within the wireless industry.
>
> Radios built using SDR concepts can allow:
>
> - Standard, open, and flexible architectures for a wide range of communication products.
> - Enhanced wireless roaming for consumers by extending the capabilities of current and emerging commercial air-interface standards.

- Over-the-air downloads of new features and services as well as software patches.
- Advanced networking capabilities to allow truly portable networks.
- Unified communication across commercial, civil, federal, and military organizations.
- Significant life cycle cost reductions.

In other words, an SDR is a wireless transceiver whose function can be altered simply by executing another program. Parameters that can be altered include frequency band, modulation type, bit rate, channel coding schemes, and protocol stack.

In addition to the above mentioned benefits it has been shown [2] that the increased degree of hardware multiplexing resulting from programmability may make it possible to reach smaller silicon area than a corresponding direct mapped ASIC solution, even for systems only supporting a small number of standards, such as IEEE 802.11a, b, and g.

The SDRF has defined the following tiers for describing the capabilities of software defined radios:

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

Tier 1 - Software Controlled Radio (SCR)
Only the control functions of an SCR are implemented in software - thus only limited functions are changeable using software. Typically this extends to interconnects, power levels, etc. but not to frequency bands, modulation types, etc.

Tier 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.

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

Tier 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, and 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

Most existing systems belong to the first two tiers although partly software defined radios based on programmable DSP processors are employed in some larger systems such as mobile telephony base stations. Today's programmable DSP processors are typically not power and cost efficient enough to enable true SDR in handheld terminals such as smart mobile phones and PDAs.

Ideal Software Radio is currently hindered by the performance of analog to digital conversion rather than by the performance of digital circuits.

To enable SDR in low-cost handheld terminals two solutions must be available: low power configurable/tuneable radio front-ends and low cost, low power programmable baseband processors. When it comes to programmable baseband processors for terminals, few competitive results have been published. Some examples of existing programmable

baseband solutions can be found in chapter 6.

Developments in RF and analog circuits for SDR is outside the scope of this thesis. Generally, the availability of flexible radio front-ends for SDR is still low although some products have been announced [3].

2.3 JTRS and SCA

The Joint Tactical Radio System (JTRS) is an initiative by the US Department of Defense. JTRS is designed to provide a flexible approach to meeting the diverse communication needs of different kinds of military units, including everything from vehicles and fighter planes to soldier-carried terminals. The goal is to provide seamless real-time communication between warfighters, through voice, data and video. The programmability will also provide backward compatibility with a diverse set of legacy radio systems as well as systems used by coalition forces and allies.

JTRS is built upon the Software Communications Architecture (SCA) which is an open architecture framework that specifies how various hardware and software components should work together within the JTRS.

Among other things the SCA provides an abstraction layer between the so called waveform application which specifies frequency bands, modulation type, channel coding, etc, and the radio set. This allows porting of "waveform software" between different radio units. SCA also specifies form factors, interfaces, software operating system, etc. More information on JTRS and SCA can be found at the JTRS website[4].

The SCA is in the process of becoming endorsed as the open international commercial standard for software defined radio by the Object Management Group [5], an international standards body which maintains standards such as UML and Corba. SCA has also been chosen by SDRF as its implementation architecture for software defined radios.

Availability of such a standardized, platform independent "waveform API" together with programmable processors supporting this API would be a huge step towards proliferation of SDR in all kinds of radio systems. The development on this front will be of great interest for future research

on programmable baseband architectures and questions regarding how this can be supported on instruction set architecture (ISA) level will certainly be considered in the continuation of this project. Unfortunately, the general awareness of the ongoing standardization processes seems quite low [6].

The suitability of the presented architecture in the context of JTRS has been discussed [7].

Bibliography

[1] *Software Defined Radio Forum*. http://www.sdrforum.org.

[2] Eric Tell, Anders Nilsson, and Dake Liu, "A low area and low power programmable baseband processor architecture," in *Proceedings of the 5th International Workshop on System-on-Chip for Real-Time Applications (IWSOC)*, July 2005.

[3] *Sirific Wireless Corporation*. http://www.sirific.com.

[4] *Joint Tactical Radio System*. http://www.jtrs.army.mil.

[5] *Object Management Group*. http://www.omg.org.

[6] C. F. Leanderson, "Business potential of software defined radio technology," in *Proceedings of Radiovetenskap och Kommunikation*, June 2005.

[7] Dake Liu, Anders Nilsson, and Eric Tell, "Fully flexible baseband DSP processors for future SDR/JTRS," in *Proceedings of Wester European Armaments Organization (WEAO) CEPA2 Workshop*, Mar. 2005.

Part II

Design of Programmable Baseband Processors

Chapter 3
Application Specific Processors

3.1 Introduction

While the market for general DSP processors is dominated by a few large players, the market for embedded DSP solutions is shared between more than 100 different chip vendors providing a wide range of application specific architectures [1]. Embedded DSP solutions constitute more than two thirds of the entire market for DSP-centric circuits and this share is increasing.

This chapter will give an introduction to application specific programmable processors or ASIPs (Application Specific Instruction set Processors). These are programmable processors which are optimized for a specific application or application domain.

An ASIP can be seen as an attempt to find the best trade-off between flexibility and performance. Using programmable components such as general processors and DSPs in an embedded system has the advantage of flexibility throughout the product lifetime, faster development (software instead of hardware), a certain degree of error tolerance (software workarounds for hardware bugs), and the possibility of using the same hardware component for a large set of different applications and func-

tions. However, in many situations general processors can not reach the necessary performance in terms of MIPS/mW or MIPS/mm^2. To reach the highest possible performance or the lowest possible power consumption or cost, ASICs[1] are the only solution.

ASIPs can be a way to find the golden middle path between these two extremes. The art of ASIP design consists of reaching just the right flexibility, function coverage, and performance for the application or application domain while reaching significantly better area and power consumption figures than a general processor or DSP. For some applications, an ASIP can reach MIPS/mW in the same order of magnitude as a direct-mapped ASIC and silicon area may be even smaller due to the difficulties in reusing hardware between different functions in an ASIC. The means to reach these goals are specialized non-traditional architectures, optimized instruction sets, and hardware acceleration. Figure 3.1 illustrates the performance vs. flexibility trade-off.

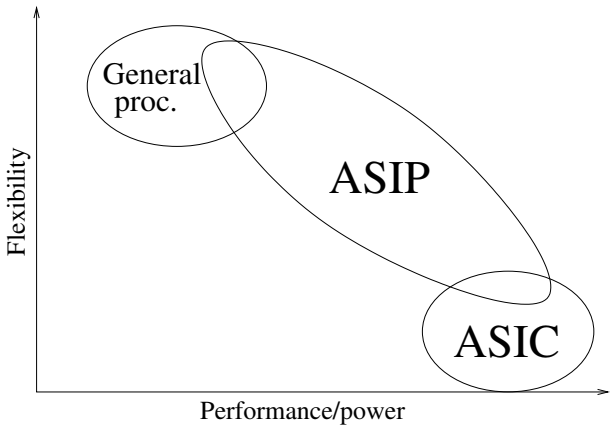

Figure 3.1: Flexibility vs. performance/power

Many ASIP architectures have features which are also common in general DSP processors, while others such as network processors may use completely unique solutions [2]. In order to design an efficient architec-

[1] It is not always clear what an ASIC is exactly, and maybe it would be better to avoid the word entirely. However, in this thesis ASIC refers to any non-programmable circuit.

ture it is important to not only study what type of operations are used in the application, but also issues like memory usage, memory access patterns, and the presence of real-time processing requirements.

This chapter is intended to give an introduction to design of application specific processors. A more thorough treatment of this subject is the book by Liu [3].

3.2 Acceleration Techniques

The means available to enhance the performance of a programmable processor can be summarized as follows, ordered according to increasing design cost:

1. Instruction level acceleration
2. Function level acceleration
3. Architecture enhancement (SIMD/VLIW/superscalar)

If this is not sufficient one may resort to a multi-processor solution.

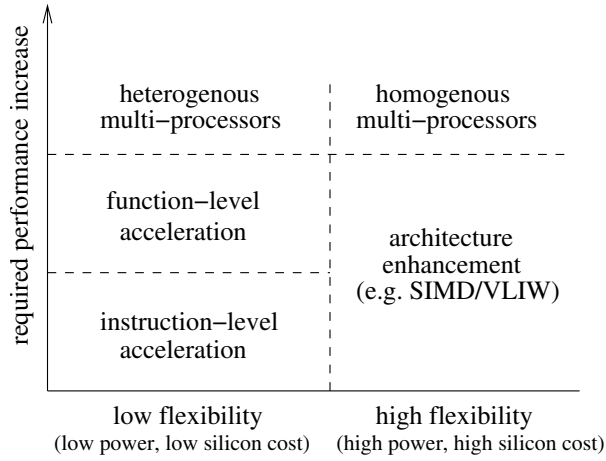

Figure 3.2: Methods for increasing performance

The existence of acceleration on instruction or function level is essentially what separates an application specific processor from a general purpose processor.

Architectural enhancements, i.e. adding more general execution units and/or issuing multiple instructions per clock cycle, can of course be useful in ASIPs as well as in general processors to increase general computing capacity. However, if the performance requirements can instead be reached by acceleration of application specific functions, this will usually lead to a more efficient solution.

3.2.1 Instruction Level Acceleration

Instruction level acceleration means that sequences of operations that are common in an application or application domain are replaced by specialized instructions. The most common example of instruction level acceleration is probably the MAC (multiply-accumulate) instruction, which is used by essentially every existing DSP instruction set. Other examples are the ACS (add-compare-select) instruction which can be used to accelerate the Viterbi algorithm [4] and instructions for efficient implementation of FFT butterflies (either a complete butterfly or an ADDSUB instruction computing a+b and a-b in parallel). Specialized instructions may also imply special data memory addressing modes.

In some cases, new instructions can be added without adding any new data path hardware but just extending the instruction decoder. In other cases execution units must be modified or even new ones added. In the worst case more extensive architectural modifications such as new data buses and memory and register file ports are needed. This may imply considerable extra verification cost in addition to the extra hardware cost. The extent of the modifications needed must of course be weighted against the number of clock cycles saved by adding a new instruction.

Implementation of instruction level acceleration is simplified by good instruction set orthogonality and a well structured instruction decoder. To facilitate future updates of an architecture it is also a good idea to do the

complete instruction decoding in the instruction decoder, instead of decentralizing parts of it to the respective execution units. Doing the latter may reduce the number of wires between instruction decoder and execution unit, but may also result in more modifications needed in order to add new instructions.

3.2.2 Special Addressing

Special addressing modes such as modulo addressing for implementation of circular buffers and bit-reversed addressing for FFTs are common features in DSP processors. For ASIPs other special addressing modes may be considered (e.g. for Viterbi decoding, rake receivers, or fast table look-ups). Whether this should be considered to be instruction level acceleration or function level acceleration or none of these is a philosophical question which will not be discussed further here.

3.2.3 Function Level Acceleration

Function level acceleration means that an entire algorithm or subroutine is replaced by fixed-function hardware. Examples are an FFT engine or a complete Viterbi decoder. The process of deciding what function level accelerators should be used is often (in this thesis and otherwise) referred to as *SW-HW partitioning*.

Addition of function accelerator blocks typically implies a larger hardware cost than instruction level acceleration but will on the other hand result in a larger performance increase. A possible additional benefit is that accelerators may run in the background while the processor continues program execution. This increases the degree of parallelism in the system.

A risk when using function level acceleration is that the accelerator block will be very application specific, i.e. that it only can be used for one very specific variant of an algorithm that is only used for one specific task or standard.

Unless all future use of the processor is known at design time it is important to not only consider the physical size of the accelerator and the number of clock cycles saved. It should also be taken into account to which extent the accelerator can be reused for different purposes or standards. Thus it is suitable to make a configurable accelerator in many cases. An FFT accelerator could handle different FFT sizes and a convolutional channel coder could handle different polynomials and code rates.

3.3 Design Flow

It is not easy to describe the ASIP design flow in a simple illustration, but one attempt at this can be found in figure 3.3. The following discussion is based on the flow used and experiences gained during this research project. The discussion is mainly applicable to embedded application specific DSP processors.

3.3.1 Requirement Specification

The assumption here is that we start from some kind of high level requirement specification. This includes a description of the function coverage and peripheral environment (interfaces) together with any constraints on performance, silicon area, power consumption, clock frequency, etc. In a typical case, a specific functional coverage and computing performance should be reached while minimizing power consumption, silicon cost, and design/verification time.

The requirement specification in this research project was essentially a set of baseband standards that should be covered, together with system requirements such as maximum frequency and power consumption based on what would be reasonable values in a handheld multi-standard radio device.

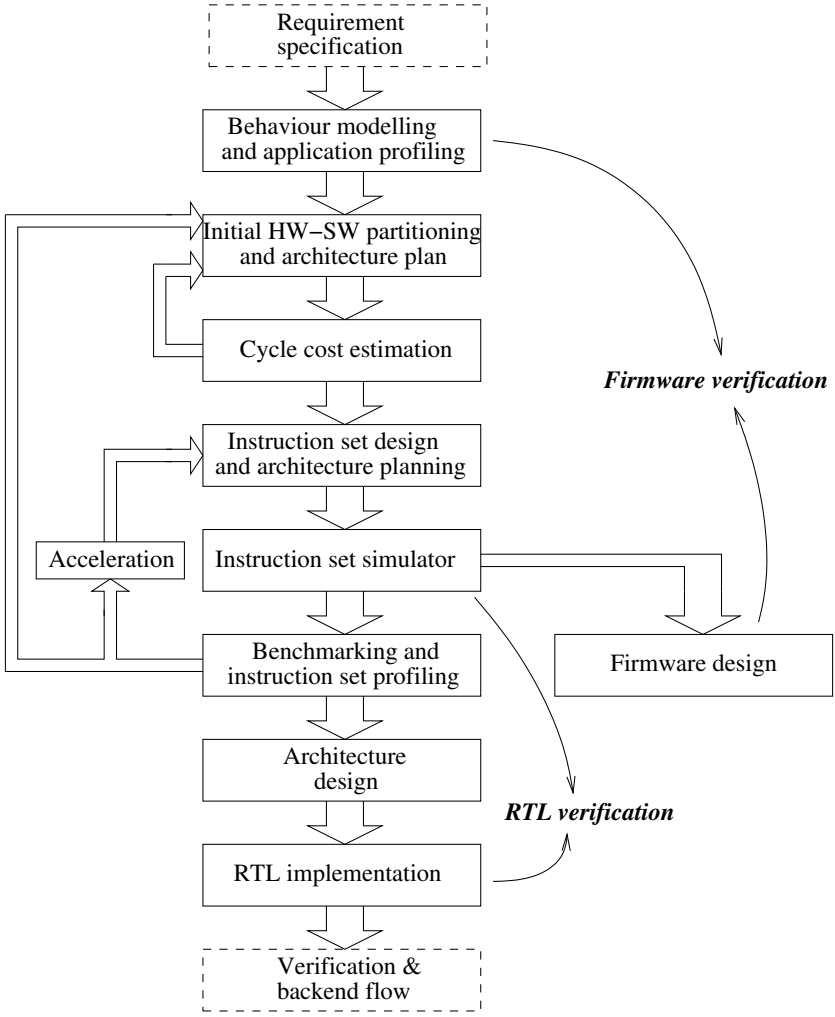

Figure 3.3: A simplified view of the ASIP design flow

3.3.2 Behavior Modeling

The first phase of the work consists of understanding the standards and exploring algorithms. Based on this, *behavioral models* are created. The first models are typically floating point models, for example Matlab models. For the baseband processor case, both transmitter and receiver is modeled

for each standard. In addition, realistic channel models are required as well as models of hardware imperfections such as frequency offset and radio non-linearities.

Channel models are often found in the standard documents. In other cases, standard channel models such as the JTC models [5] for WLAN indoor environments are used. It should also be possible to feed real recorded air data to the receiver model. This setup allows realistic bit error rate simulations and testing of different baseband processing algorithms (to the extent which the algorithms are not specified in the standard).

The next step is to create fixed point models. This phase determines the required precision needed at different stages of processing. In the radio baseband case this includes specifying the number of bits in ADC and DAC, unless this is given beforehand. Another issue to consider at this stage is the implementation of Automatic Gain Control (AGC) and Automatic Frequency Control (AFC) which is connected to the number of bits needed (see section 4.3.1).

It is important to build the behavioral model with software implementation in mind, i.e. an effort should be made to find algorithms that are suitable for software implementation and easily mapped to assembly instructions.

The output of the behavioral modeling stage is a bit-true model of the application. This means that all algorithms and parameters have been fixed as well as data widths and scaling factors. The behavioral models have three main purposes:

1. Learning the standards and determining what algorithms should be used.

2. Profiling the application, i.e. determining what types of operations and what computing performance is needed. This information is directly used in hardware design to decide the necessary computing capacity and to guide the initial SW-HW partitioning and architecture decisions.

3. Specifying the firmware. Firmware is written by direct translation of the behavioral model into hardware specific code (typically assembly code). It is also used as an executable specification against which the firmware can be verified.

3.3.3 Initial Architecture Plan and MIPS Estimation

Based on the behavioral models and profiling, an initial architecture proposal is made. This includes decisions on architecture type (super scalar/VLIW/SIMD, single or dual MAC unit, etc.), supported data types and widths, memory organization, and possibly instruction or function level acceleration of some very common and MIPS consuming functions.

With the architecture proposal as a starting point the behavioral model is mapped to instructions in order to get an estimation of the required number of clock cycles. It is important to not only consider the computing cost in this process but also cost for memory accesses. This is necessary both in order to get an accurate cycle cost estimation and to decide the required memory bandwidth.

These steps will typically go through a few quick iterations with successively more accurate cycle cost calculations. The result should be a suitable architecture that meets the cycle cost requirements with some extra head room for control flow instructions.

It is essential (although easier said than done) to get an accurate cycle cost estimation in order to avoid architecture redesign at later stages or additional iterations in the more time consuming instruction set design and benchmarking loop of the design flow.

3.3.4 Instruction Set Design and Architecture Planning

The goal of this stage is to specify the instruction set of the processor. Note that what is designed here is the hardware/software interface (i.e. the programming model), not the hardware itself. However, there are of course some properties of the hardware that will be fixed here, such as the register set and types of execution units.

Designing an instruction set architecture (ISA) is a non-trivial task which will typically go through several iterations. One fundamental trade-off to make is between *orthogonality* and instruction word length.

Orthogonality is a somewhat vague concept in this case. It refers to the completeness and regularity of the instruction set. An orthogonal instruction set has few special cases and all instructions use the same instruction modes and the same registers in the same way. This makes the job easier for programmers and compiler designers. An orthogonal instruction encoding means that all instructions are encoded according to the same pattern; A certain bit in the instruction word should as far as possible always have the same meaning. This reduces instruction decoding complexity but also simplifies addition of new instructions and implementation of operand stopping techniques in order to reduce power consumption (see section 3.5).

The goal of good orthogonality is in direct conflict with the goal of small code size. Orthogonal instruction encoding inevitably introduces redundancy in the instruction word, leaving many codes unused. Means towards shorter instructions such as implied addressing or restricting register use also directly reduces orthogonality. So does the introduction of complex application specific instructions introduced to improve performance and reduce memory usage.

3.3.5 Instruction Set Simulator

The instruction set simulator (ISS) is a piece of software which simulates the behavior of the processor in a bit-true and cycle-true way. This means that it should be possible to input an assembly program and run it and the ISS should produce exactly the same result as the hardware would, at exactly the right clock cycle.

The ISS has multiple roles in the design flow:

Benchmarking tool: The ISS is used during the ISA design iteration to benchmark the performance and profile application code.

Hardware specification: The ISS specifies the exact behavior of the hard-

ware. During RTL verification, the HDL code is verified against the ISS behavior.

Firmware design tool: The ISS is the central component of the firmware design environment. It provides firmware designers with a way of designing and testing firmware without available hardware.

In order to be useful, the ISS provides debugging features such as breakpoints, cycle-by-cycle execution, hardware observability, and controllability. Statistics collection and profiling features are also useful both for instruction set and firmware optimization.

Note: The requirements on the simulator for hardware development and firmware development are slightly different. For hardware verification it is essential that the simulator is bit and cycle accurate, while short simulation time on the host machine is not as important since it will be much faster than RTL simulation anyway. To reach reasonable execution times for very large applications during firmware development, faster simulation speed may be more important than to have cycle-exact simulations in all parts. One option could be to develop a separate version of the simulator for firmware development once the ISA is fixed.

3.3.6 Benchmarking and Profiling

Benchmarking generally consists of running a specified piece of code on a processor and collecting statistics such as execution time, cycle cost, or memory usage. The purpose is typically to evaluate the performance of a processor for a certain application or application domain and compare it to other solutions. An introduction to benchmarking of DSP processors can be found in [6].

In ASIP design however, the purpose is rather to verify that the performance is sufficient for the application at hand, and if not to determine what parts need improvement.

It is obviously essential that the code used for benchmarking and profiling is representative of the entire application or at least the most timing critical parts of the application. The ultimate benchmark would of course

be the entire application itself. In reality though, this is typically not available until later stages of the design flow. Instead, *kernel benchmarking* is applied. This means that the processor is benchmarked for the most common functions (i.e the kernel functions) of the application. Examples of kernel benchmarks could be a filter, an FFT, or a finite state machine.

Notice that a useful kernel benchmark should include the overhead for initialization and wrapping up as well as rounding and saturation operations when appropriate. It is also important to benchmark control oriented code since it is a significant part of most applications.

Third party kernel benchmarks for general DSP processing are available, e.g. from BDTi [6]. EEMBC [7] provides benchmarks targeted for different application domains such as automotive/industrial, consumer, networking, and telecommunications. Even if these are not representative of your exact application they may still be useful for initial benchmarking. Since they are quite well known, these benchmarks may also be important for making comparisons to other processors for marketing purposes.

Profiling consists of collecting statistics on the occurrence of different instructions and the execution time spent on different types of operations, from the benchmarking. This includes initialization, addressing, and loop overhead as well as actual useful arithmetic operations. Together with statistics on program and data memory usage, this provides information to guide instruction set improvements and further instruction or function level acceleration.

Some details on benchmarking and profiling related to this research project can be found in appendix A.

3.3.7 Acceleration

If the benchmarking shows that the performance requirements are not met, additional acceleration must be applied. The profiling information is used to guide the decision. Instruction level acceleration is typically applied if the design is close to meeting the requirements, while function level acceleration is applied if a larger performance increase is needed.

After acceleration has been applied, benchmarking is reiterated to verify if the speedup is sufficient or if additional acceleration is needed.

3.3.8 Architecture Design

Data Path

The data path is the part of the processor that does all the "useful" work, e.g. execution units, register files, data memories, and buses connecting these components. The data path design consists of mapping the instruction set to hardware. Execution units, registers, and interconnects (buses) are added to the architecture until there is hardware to support all instructions. One goal is to reach the highest possible degree of multiplexing, i.e. hardware components should be reused between as many instructions as possible in order to minimize the total amount of hardware.

Another important issue is the pipeline design. The number of pipeline steps must be decided and the pipeline balanced to avoid unwanted critical paths.

Control Path

The control path is all hardware needed to generate the appropriate control signals for the data path. This includes program memory, instruction fetching, instruction decoding, and program flow control (logic handling jumps, hardware loops, interrupts, pipeline hazards, etc).

While the data path constitutes the major part of the hardware, the control path is usually the part requiring the most care in both design and verification.

Memory Subsystem

Memory is a major performance bottleneck in many systems and the gap between computing performance and memory performance keeps growing. The access time for memory is typically longer than the critical path of computing logic except for small on-chip memories. DSP applications

often require a large amount of memory and a very high memory bandwidth, making design of the memory subsystem a major challenge.

In systems requiring large amounts of memory, much design effort is put into features such as cache control, MMUs, and advanced DMA features. These are complex functions and a large set of possible execution cases must be carefully verified to avoid deadlocks or data corruption.

This thesis is however mostly concerned with processors using relatively small on-chip memories. The issue of memory subsystem design is thereby limited to number, types, widths, and sizes of memories, bus organization, and addressing circuitry.

3.3.9 RTL Implementation and Backend Flow

The RTL implementation starts with stepwise refinement of the architecture and results in the final implementation, e.g. in a hardware description language (HDL). The process is similar to that of any ASIC and will not be discussed here. The same goes for the backend design flow.

3.3.10 Verification

As mentioned earlier the instruction set simulator is an important tool for RTL verification since it is in fact an executable specification of the hardware behavior. Apart from this the issues are the same as in verification of HDL based hardware in general.

The applied flow is bottom-up, starting with testbenches for individual subblocks of the design. At top level, the verification is based on running actual programs on the processor and comparing to the result from the ISS. First the general behavior of each instruction is verified, followed by verification of identified corner cases. This implies verification of highest/lowest data values, overflow cases, etc. as well as potentially problematic control flow cases such as interrupts during (nested) hardware loops/subroutine calls etc. The last phase is random testing which is an attempt to find any corner cases not previously discovered. The random testing consists of running a program with randomly generated

input data. It is also possible to generate random assembly instructions, although with some restrictions. The random testing should in principle run as long as possible (the process of generating random data, running it in the simulator and on the RTL code, and comparing the outputs must of course be automated).

Backend verification (layout vs. schematic and physical verification) will not be discussed here. See instead e.g. [8].

3.3.11 Concluding Remarks

The flow may be a lot more complex in a practical case than what is depicted in figure 3.3. If it is found during architecture design or RTL implementation that some features are not practical for implementation, the instruction set design may have to be iterated. If it is found in the backend flow that timing constraints are not met, architecture design and/or RTL implementation must be iterated, e.g. for pipeline repartitioning. If extra pipeline steps are needed the process may even have to be reiterated from instruction set design.

Furthermore, for some tasks it is not clear to which design step they belong. One such task is pipeline design. Since a changed pipeline depth typically will be visible to the programmer one could argue that it is part of the instruction set design. However, the details can typically not be fixed until RTL design. For designs operating at high frequencies using deep pipelines, feedback may even be needed from the backend flow.

It is also important to notice that many of the activities will be going on in parallel. This is necessary not only in order to reduce the total design time. If architecture design and RTL implementation is started early it can give valuable feedback to the instruction set design. It is also useful for the backend team to get an early version of the RTL code in order to plan layout, create necessary scripts, get an early estimation of the chip area, and trim the flow. Of course it is also important to discover any physical design problems as early as possible. Finally, firmware design should also start early, not only because it is a resource demanding task,

but also because there are no better benchmarking code than the actual application firmware that will ultimately run on the processor.

3.4 Processor Tool Chain

In addition to the ISS, the assembler is obviously a necessary tool for programming. Special architectures may also need specialized tools for tasks like configuration, scheduling, and hardware allocation.

It may also be very valuable to be able to program the processor in a higher level language such a C. However, it is often difficult to build a compiler that can produce efficient code for irregular application specific architectures. Typically a modified version of C is used, often with restrictions on data types, and using compiler known functions added for access to architecture specific features. In fact, most code will often still be architecture specific and often practically as low level as the assembly code. Kernel functions will probably still be written in hand-optimized assembly code. However, constructs such as for-loops and function calls may still be very useful, making a compiler valuable for writing non-timing-critical code.

The tools are usually put together in an integrated design environment (IDE), typically consisting of a graphical user interface (GUI) used by the programmer for writing code as well as for accessing the compiler/assembler, for simulation, and for debugging.

3.5 Low Power Design

Many generally available low-power design techniques can obviously be used in processor design as well. However, this section will discuss low-power issues that may be of particular interest in programmable processors.

3.5.1 Operand Stopping

The purpose of operand stopping is to avoid toggling in registers and logic blocks that are not producing any useful result. The input to an execution unit could for example be forced to zero or keep its previous value whenever the unit is not used. This is of particular interest in processors since they typically use multiplexed buses with high activity that are connected to several execution units. Without operand stopping any toggling on the bus would ripple through all execution units, producing a lot of unnecessary switching.

An instruction set with good orthogonality and a well structured instruction decoder will give a cost of implementing operand stopping in a processor that is typically quite low both in terms of design time and silicon area.

3.5.2 Memory

Memory normally contributes to a very significant part of the area and power used by a processor. Minimizing the memory size and memory accesses is therefore a very important issue. Memory power consumption has been discussed widely in the context of memory hierarchies and cache organization [9]. However, this project is restricted to systems with only relatively small on-chip memories and without caches but typically requiring a rather high memory bandwidth (in the order of hundreds of bits per clock cycle). The design of the memory architecture in such systems still requires significant effort. E.g. increased memory bandwidth can be reached either by using wider memories, multiple memory banks, or multi-port memories. These variants offer different trade-offs between flexibility, silicon area, and power consumption. The best choice depends on the memory sizes and data access patterns of the application at hand.

3.5.3 Control Overhead

One reason why a processor loses power in comparison to an ASIC is the additional control overhead for program memory, instruction fetching, and decoding. To reach power consumption close to an ASIC, the control path must have a low complexity. The obtainable result depends on the complexity of application.

One way of reducing the control power is by introducing idle or sleep modes to avoid the need for busy-wait loops. One solution is to use an idle instruction which halts instruction fetching until an interrupt event occurs.

3.5.4 Leakage

With ever decreasing feature sizes, the issue of leakage in transistors becomes increasingly important [10]. Ways to deal with leakage, such as shutting of the power supply or substrate biasing would typically be handled by system level power management located outside the processor, in either case it will not be further discussed here. However, one simple and efficient way of reducing the leakage is to reduce the number of gates, i.e. to make an area efficient design. There are cases where an ASIP can actually reach a lower area than fixed function hardware, due to a higher degree of hardware multiplexing. A multi-standard baseband processor may be such a case, as shown in [11].

3.5.5 Acceleration and Parallelism

It is clear that power may be saved in a programmable processor by running a function unsuitable for software in a fixed function hardware accelerator. However, this is not only a result of the higher efficency in the hardware implementation. Increased parallelism can in itself also lead to lower power consumption.

According to the well known formula, the dynamic power consumption can be written $P = \alpha V^2 f C$, where V is the supply voltage, f is the

clock frequency, C is the total capacitance, and α is the switching activity.

Consider a situation where we have a function running in a hardware block. If we simply copy this block and instead perform the same operation using two identical blocks but running at half the frequency, the task will finish in the same time and the dynamic power consumption will be the same (C is doubled and f is halved). However, since the hardware is now running at a lower clock frequency it will be possible to use devices with lower drive strength and hence lower power consumption. It may also be possible to lower the supply voltage or increase the threshold voltage to reduce leakage in the entire design. The end result is that power consumption has been traded against silicon area.

To conclude, in addition to reduced control overhead, acceleration leads to reduced clock frequency which further reduces power consumption. One assumption made here is that the increased power due to parallelization overhead, leakage in the added circuitry, and longer wires is not large enough to cancel out the positive effects. This assumption may not be true in all cases, especially not for future feature sizes with increasing impact of both wires and leakage.

3.5.6 Data Width Masking

In fixed function hardware, the computing precision at every stage of an algorithm is assigned to a specific computing or storage unit which can be optimized to the minimum number of required bits in order to save hardware and power. In a programmable processor on the other hand, all computations use the same hardware. This means in principle that the hardware must be designed for the largest precision required in the application.

One way of reducing switching, mainly in arithmetic units, is by zeroing out the least significant bits of data whenever the native computing precision is not needed. This can be done by adding hardware which can be configured to mask out a selected number of least significant bits of the data depending on the required precision. The masking can be performed

Table 3.1: Relative dynamic power consumption using data masking and reduced hardware precision.

Masked 16-bit MAC		Different MAC unit precisions	
Masked Precision	Power	Data path precision	Power
16-bit (no masking)	1.00	16-bit	1.00
12-bit (mask 4-LSB)	0.47	12-bit	0.41
10-bit (mask 6-LSB)	0.31	10-bit	0.26
8-bit (mask 8-LSB)	0.18	8-bit	0.12
6-bit (mask 10-LSB)	0.09	6-bit	0.06
4-bit (mask 12-LSB)	0.04	4-bit	0.02

either at the input of arithmetic units or at the memory interfaces.

Table 3.1, originally presented in [12], shows results from an investigation of the effects of masking in a multiply-accumulate unit. The dynamic power consumption in MAC units with different data widths was compared to that of a MAC unit of full width but with data masking at the inputs. The figures were obtained from toggling statistics generated by simulations in Mentor Graphics ModelSim and physical parameters extracted by Cadence PKS synthesis, using the tool which is described in the master thesis by J. Nilsson [13]. As can be seen, the switching power reduction is more than 50% already at a reduction from 16 to 12 bits and the difference between masking and using reduced hardware is only six percentage points.

A more thorough investigation of the effects of adding variable data width features in the scope of this project has not yet been concluded.

Bibliography

[1] *Forward Concepts*. http://www.fwdconcepts.com.

[2] Thomas Henriksson, Ulf Nordqvist, and Dake Liu, "Specification of a configurable general-purpose protocol processor," *IEE proceedings of Circuits, Devices, and Systems*, no. 3, pp. 198–202, 2002.

[3] Dake Liu, *Design of Embedded DSP Processors, 2nd ed.* Linköping University, 2004.

[4] Jeong Hoo Lee, Weaon Heum Park, Jong Ha Moon, and Myung H. Sunwoo, "Efficient DSP architecture for Viterbi decoding with small trace back latency," in *Proceeding of the IEEE Asia-Pacific Conference on Circuits and Systems*, pp. 129–132, Dec. 2004.

[5] Karen Hallford and Mark Webster, *Multipath Measurement in Wireless LANs.* Intersil corporation, application note, 2001.

[6] Berkeley Design Technology, Inc. (BDTI), *Evaluating DSP Processor Performance (white paper)*. http://www.bdti.com.

[7] *Embedded Microprocessor Benchmark Consortium*. http://www.eembc.com.

[8] Michel J. S. Smith, *Application Specific Integrated Circuits*. Addison-Wesley, 1997.

[9] Vasily G. Moshnyaga and Koji Inoue, *Low Power Electronics Design*, ch. 25 - Low Power Cache Design. Ed. Christian Piquet, CRC Press, 2004.

[10] Antonio Ferré and Joan Figueras, *Low Power Electronics Design*, ch. 3 - Leakage in CMOS Nanometric Technologies. Ed. Christian Piquet, CRC Press, 2004.

[11] Eric Tell, Anders Nilsson, and Dake Liu, "A low area and low power programmable baseband processor architecture," in *Proceedings of the 5th International Workshop on System-on-Chip for Real-Time Applications (IWSOC)*, July 2005.

[12] Dake Liu and Eric Tell, *Low Power Electronics Design*, ch. 23 - Low Power Programmable Baseband Processors. Ed. Christian Piquet, CRC Press, 2004.

[13] Jesper Nilsson, *Mixed RTL and gate-level power estimation with low power design iteration (Master thesis)*. Linköping University, 2003.

Chapter 4
Introduction to Baseband Processing

The purpose of this chapter is to give an overview of the DSP tasks involved in radio baseband processing, to define the function coverage of the baseband processor, and to introduce some of the concepts and terminology used later on.

4.1 System Overview

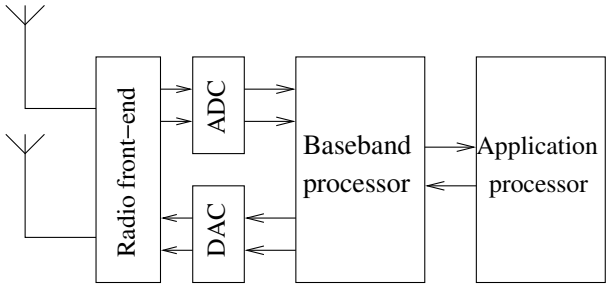

Figure 4.1: Overview of a radio transceiver

Figure 4.1 gives an overview of a radio transceiver. The application processor runs the application that generates or consumes the transmit-

ted data. It may be a voice codec in a mobile phone, a video decoder in a digital TV receiver, or a general processor running an operating system and a web browser. Here it is assumed that the application processor also handles media access control (MAC) and higher protocol layers. In reality the application processor is typically a micro controller or a general purpose processor, often combined with a DSP or application specific hardware.

The task of the baseband processor is to handle the digital parts of the physical layer (PHY) processing (i.e. the "modem"). Figure 4.2 illustrates the tasks involved. This chapter will give an introduction to the different tasks. A more comprehensive discussion of many of the issues is e.g. [1].

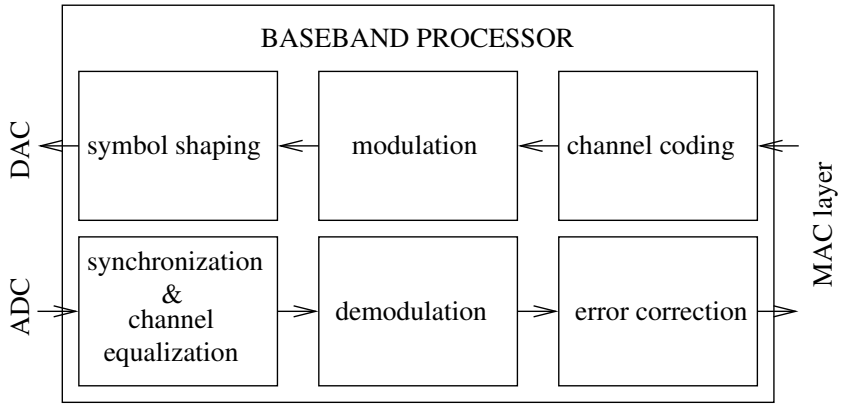

Figure 4.2: Overview of baseband processing

4.2 The Transmitter

The transmitter flow in the baseband processor consists of three main functions: Channel coding, digital modulation, and symbol shaping.

Channel coding includes different methods for error correction (i.e. redundant coding such as Reed-Solomon or convolutional codes) and error checking (e.g. cyclic redundancy check [CRC]). *Interleaving* is applied after redundant coding in many systems to make sure that consecutive

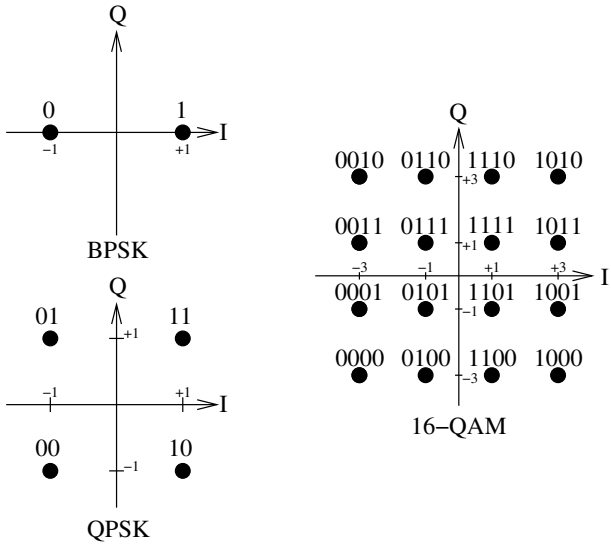

Figure 4.3: Examples of signal constellations

bits in the bit stream are not transmitted consecutively in time (or on the same frequency in the OFDM case). Thereby the robustness against burst errors is improved. *Scrambling* is often used to turn the bit stream into a pseudo-noise sequence without long runs of ones or zeros.

Digital modulation is the process of mapping a bit stream to a stream of *symbols*, each consisting of a number of complex samples. The first (and sometimes the only) step of the modulation is to map groups of bits to complex values according to a specific signal constellation as shown in figure 4.3. In most cases, a second step called domain translation is used. In an Orthogonal Frequency Division Multiplexing (OFDM) system, an inverse fast Fourier transform (IFFT) is used for this step. In direct sequence spread spectrum (DSSS) modulation the complex value is multiplied by a so called spreading sequence of ones and minus ones, see further sections 4.4 and 4.5.

The last step is symbol shaping which consist of transforming the square wave into a band-limited signal, typically using a finite impulse response (FIR) filter. This is necessary to make sure no part of the trans-

mitted signal lies outside the permitted frequency band.

The real and imaginary part of the complex valued signal, also known as the in-phase (I) and quadrature-phase (Q) signals, are converted into analog signals and sent to the radio front-end. The I and Q signals are mixed with carrier signals with a 90 degrees phase difference and added. The result is a sinusoid signal at the carrier frequency with an amplitude corresponding to the absolute value of the complex number and a phase corresponding to the argument of the complex number. This signal is amplified, filtered, and transmitted through the antenna.

4.3 The Receiver

The receiver flow is essentially the opposite of the transmitter flow: The baseband processor receives the digitized I and Q signals and has to re-generate the transmitted bit stream.

Reception is significantly more challenging than transmission since the transmitted signal is affected by several kinds of distortions before reaching the baseband processor.

Some challenges that must be handled are:

- Noise
- Multi-path channel fading
- Mobility
- Large dynamic range
- Frequency offset between transmitter and receiver
- Non-linearities and other distortion in RF and analog circuits

4.3.1 Dynamic Range

The strength of the received signal can vary dramatically depending on the distance to the transmitter and depending on channel conditions. It

is not unusual to have to handle a dynamic range of 60-100 dB [2]. Since it is not practical to design systems with so large dynamic range, several levels of automatic gain control (AGC) is normally used. The received energy must continuously be measured and the gain of front-end components adjusted to normalize the received energy at the ADC. With a good gain control algorithm fewer bits are needed in the ADC and baseband processor.

4.3.2 Synchronization

Synchronization can be divided into two steps. The first step is to detect an incoming signal or frame, this is known as energy detection. Functions such as AGC, frequency offset estimation, and antenna selection may also be carried out at this time. The next step is symbol synchronization which aims to determine the exact timing of incoming symbols. These operations are typically based on complex auto or cross correlations. In many systems, each frame contains a known preamble or pilot sequence used for this purpose.

4.3.3 Channel Estimation and Equalization

In essentially all radio systems the transmitted signal is subject to multi-path propagation, i.e. the received signal will contain multiple copies of the tranmitted signal due to reflections of objects in the environment. The different signal paths will have different strength and timing and will add constructively or destructively at different frequencies. The effects of multi-path propagation is known as fading. One such effect is overlap between consecutive symbols, so called inter-symbol interference (ISI).

The fading can be characterized by the impulse response of the channel. One way of measuring the severity of the fading is the RMS delay spread which describes the root-mean-square distance in time between the different signal paths.

If the symbol time is significantly longer than the delay spread (or in other words if the signal bandwidth is small enough), the fading can be

considered constant over the signal spectrum, so called *flat fading*. When flat fading can not be assumed the receiver must compensate for the fading by channel equalization.

In outdoor environments the difference between signal paths can be in the order of several micro seconds. As an example a delay spread of 1 μs corresponds to a 300 m difference between the propagation paths. Under such conditions the minimum symbol time without advanced channel equalization would be approximately 10 μs (ten times the delay spread), corresponding to a symbol rate of 100k symbols/s.

In most cases correlation with known data is used to estimate the channel and some kind of filtering is used for equalization.

4.3.4 Frequency and Timing Offset

If the oscillator frequencies in the transmitter and receiver are not exactly the same this will cause a small rotation in the complex number plane between each consecutive sample. In many systems, this error must be estimated and compensated. This is done either by modifying the oscillator frequency via some adaptive algorithm, or by rotating each sample slightly relative to its predecessor.

One way of avoiding part of the problem is by using *differential modulation* where the information is represented by the phase difference between two consecutive symbols instead of the absolute phase.

However, unless the oscillator frequency is controlled one may still have to compensate for timing offset, i.e. the difference between sample time in ADC and DAC. For example samples may have to be inserted or removed from the data stream at regular intervals.

4.3.5 Mobility

In a situation where the transmitter or receiver is not stationary the channel conditions will change continously. This means that an estimated set of channel parameters will become obsolete after a certain period of time, known as the channel coherence time. In systems with high mobility, the

channel coherence time may be in the order of one or a few symbol times and the channel must therefore be continously tracked. This may consume a very significant part of the processing power in the receiver. In the WLAN applications discussed later in this thesis, the channel can be assumed to be stationary for the duration of a frame.

Mobility also causes Doppler shift which results in effects similar to those caused by frequency offset.

4.3.6 Demodulation and Channel Decoding

Demodulation is the opposite operation of modulation. It involves an FFT in OFDM systems and correlation with the spreading sequence in DSSS systems. The last step of demodulation is *demapping*, i.e. finding the closest point in the constellation diagram and converting the complex value into one or more bits.

Channel decoding consists of deinterleaving, error correction, descrambling, and possibly error checking. Error correction (e.g. Viterbi or turbo decoding) tends to be very demanding functions.

4.4 OFDM Modulation

OFDM is a modulation method where the data is sent over a large number of adjacent *sub-carrier* frequencies simultaneously. The symbols to be transmitted on each of the sub-carriers are collected in the frequency domain and are all simultaneously transformed to a time domain signal using an IFFT.

The idea is that each sub-carrier signal should have a very small bandwidth and hence only be subject to flat fading. Thereby a complex channel equalizer can be avoided. Instead channel equalization only consists of multiplying each sub-carrier with a complex valued constant in order to scale and rotate the constellation point to its correct position. To further increase the robustness against ISI a *cyclic prefix* (CP) is often added to each symbol after the IFFT. The CP acts as a guard interval: as long as

the delay spread is shorter than the CP, the ISI is not a problem.

OFDM presently seems to be gaining in popularity. It is used in e.g. the IEEE 802.11a and IEEE 802.11g Wireless LAN standards, digital audio and video broadcasting (DAB/DVB), and the recent WiMax standard. It is being considered for several future standards including fourth generation mobile telephony. Figure 4.4 illustrates typical OFDM processing flows.

4.5 Spread Spectrum Modulation

The purpose of spread spectrum modulation is to spread the radio signal over a larger spectrum than necessary in order to make it less sensitive to interference. Two basic types of spread spectrum modulation exists.

In frequency hopping spread spectrum, the transmitter regularly alternates between different carrier frequencies according to a predetermined pattern.

In direct sequence spread spectrum (DSSS), each symbol is multiplied with a pseudo-noise sequence of length N to create N so called *chips*. This operation is known as spreading and N is called the spreading factor. The chips are then transmitted at N times the symbol rate. In the receiver the chips are correlated with the spreading sequence to recover the symbol. This is known as despreading. DSSS is used in the IEEE 802.11b WLAN standard (a.k.a. WiFi).

A variant of DSSS is code division multiple access (CDMA) which is used in the third generation mobile telephony systems. CDMA uses the fact that if each user uses its own spreading sequence (or code), which is orthogonal to all other used spreading codes, several users can transmit on the same frequency in the same time slot. After despreading, the interference from other users will just look like white noise.

In DSSS and CDMA systems a structure known as a *rake receiver* is often used for channel equalization. In the rake receiver, different signal paths are despread separately (usually the 3-6 strongest paths are selected) and the contributions from the different paths are then added constructively.

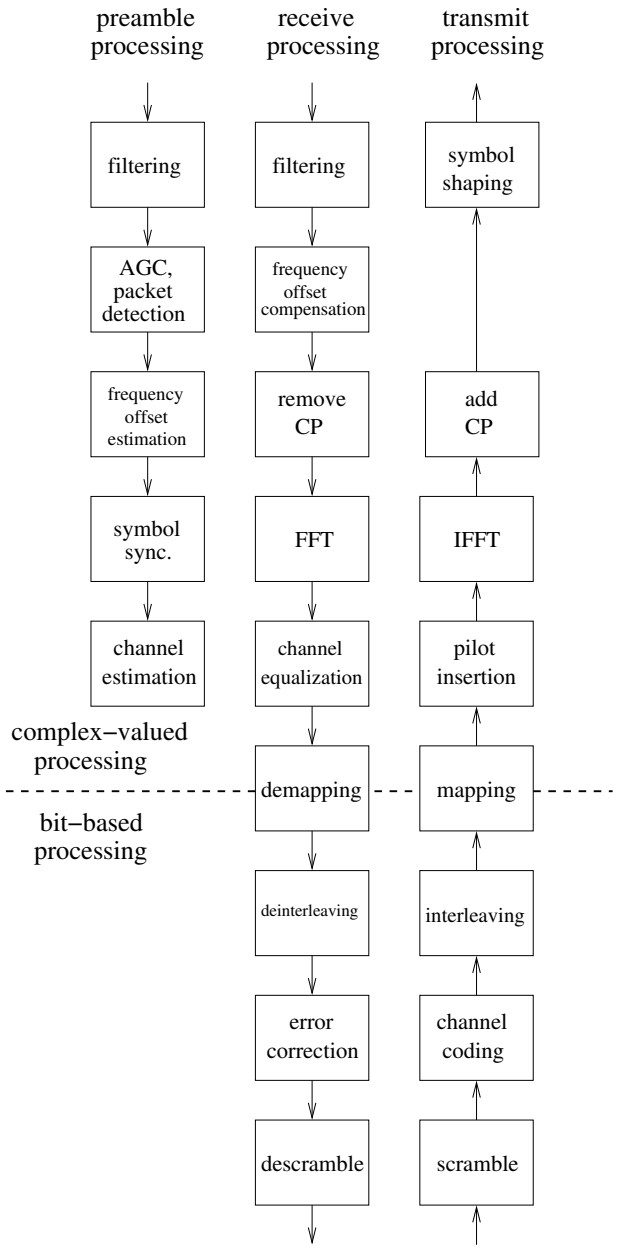

Figure 4.4: OFDM processing flows

In e.g. 3G systems which may have a large delay spread (several microseconds) and high mobility (terminals may travel at up to 250 km/h), the rake processing is the single most computing intensive part of the receiver. Figure 4.5 illustrates the DSSS/CDMA processing flows.

4.6 MIMO Systems

MIMO, which stands for multiple input multiple output, is a method used to reach higher data rate with a given bandwidth and SNR by using multiple antennas in both transmitter and receiver. MIMO has been widely discussed over the last few years and is proposed e.g. for the upcoming IEEE 802.11n WLAN standard.

MIMO will cause considerable processing challenges for future baseband processors, e.g. in the form of matrix inversion operations necessary in the receiver [3].

4.7 Computational Complexity

The computing capacity needed in the baseband processor depends mainly on the data rate and the mobility. The requirements increase superlinearly with the data rate. For many parts of the flow, the number of operations per data bit is constant. Other parts, e.g. channel equalization, will require more processing per bit when the symbol rate increases.

Mobility mainly determines how often channel estimation must be repeated. The processing requirements for channel estimation and tracking will be approximately inversely proportional to the channel coherence time or proportional to the velocity of the transmitter or receiver.

Data rate and mobility for some standards is illustrated in figure 4.6. One main motivation for using a programmable processor is that with enough flexibility and a certain level of computing capacity a range of systems with different types of modulation and different trade-offs between data rate and mobility can be covered.

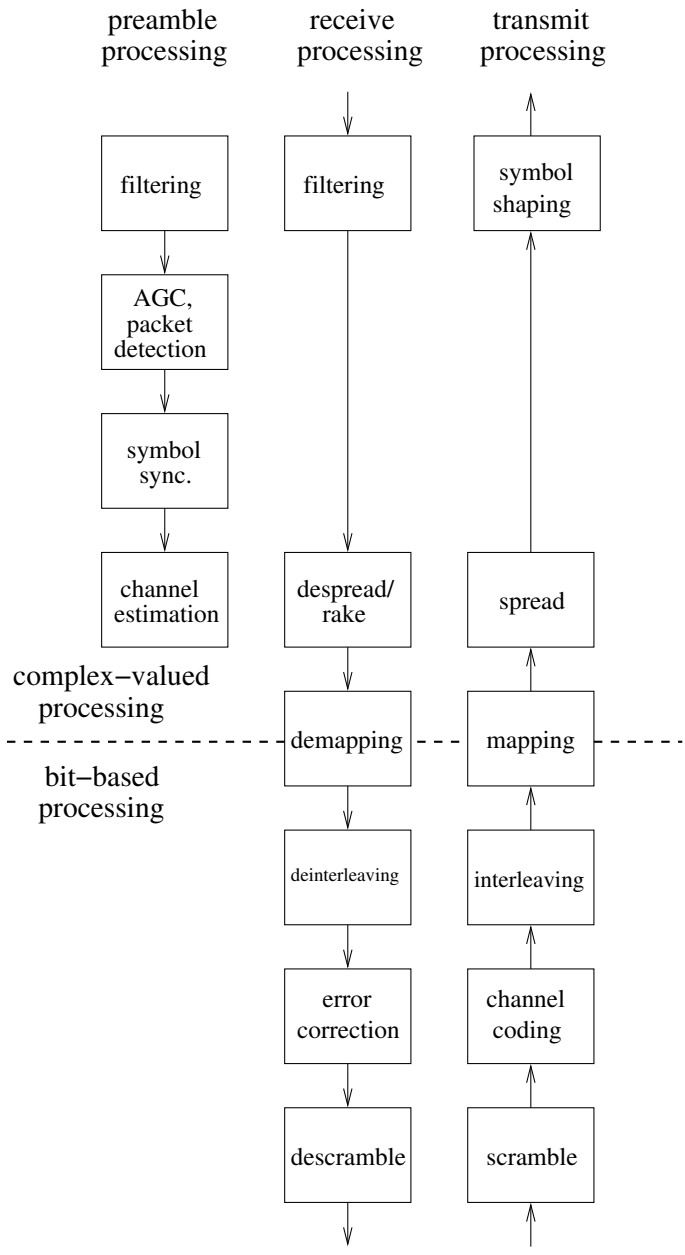

Figure 4.5: DSSS/CDMA processing flows

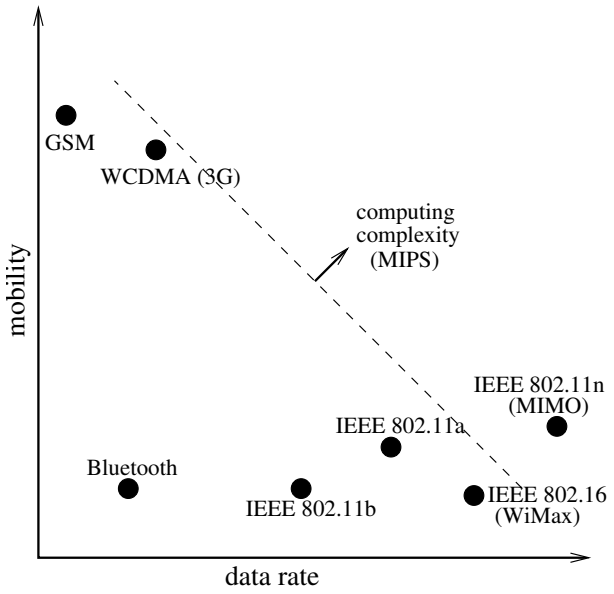

Figure 4.6: Data rate and mobility for different standards

The performance of conventional DSP processors is typically only sufficient to manage low-rate, low-mobility systems. Even for moderate data rates (more than a few hundred kbit/s) or mobility requirements (more than walking speed), high-end processors using VLIW/SIMD technologies and/or operating at several GHz will be required. These processors are often too expensive and power consuming. The rest of this thesis discusses how more cost- and power-efficient programmable solutions can be obtained.

Bibliography

[1] John B. Anderson, *Digital Transmission Engineering*. IEEE Press, Prentice Hall, 1999. ISBN 0-7803-3457-4.

[2] Bosco Leung and Behzad Razavi, *RF Microelectronics*. Prentice Hall, Pearson Education, Inc., 2004. ISBN 0-13-861998-0.

[3] Haiyan Jiao, Anders Nilsson, Eric Tell, and Dake Liu, "MIPS cost estimation for OFDM-VBLAST systems," in *paper submitted for review*, 2005.

Chapter 5

Programmable Baseband Processors

5.1 Introduction

This chapter will discuss what implications the properties of typical baseband applications have on the design of ASIPs for baseband processing. The focus is on terminals for mobile telephony and computer networking. The requirements are slightly different for unidirectional or broadcasting standards such as DAB and DVB since the latter need more data memory but have less requirements on latency.

5.2 Processing Requirements

Going back to figures 4.4 and 4.5 it is clear that the operations needed in a baseband processor can be divided into to two main classes: Calculations based on complex valued data and bit manipulation operations. This section will discuss the implementation of these two types of operations.

5.2.1 Convolution-Based Complex-Valued Processing

The data between the ADC/DAC and the mapping/demapping consists of I/Q-pairs, which are treated as complex values. The functions to be carried out are modulation/demodulation, synchronization, channel equalization, etc. The FFT function is heavily used in OFDM systems. In DSSS systems, modulation/demodulation is based on multiplication/correlation with a spreading sequence. Other common operations are correlation and filtering. These are all very common DSP algorithms, most of them convolution based. Traditional DSP processors are able to carry out these kinds of algorithms rather efficiently thanks to MAC units and optimized memory and bus architectures.

The following areas where optimizations can be made have been identified:

Complex-valued computations: Since almost all computations are based on complex-valued numbers, it is beneficial to optimize the ISA for complex calculations. This includes using complex data paths and data types, and instructions for operations such as conjugate, multiply by i, multiplication/MAC with conjugate, and complex absolute value.

Instruction level acceleration: In addition to the above mentioned operations on complex values, a few operations common in baseband processing were found to be suitable for instruction level acceleration. This includes finding the position of the maximum complex absolute value in a vector, accumulation of square absolute values, FFT butterfly, and combination of the last layer of FFT butterflies with frequency domain filtering. All of these are certainly operations that are common in other applications too. In a processor that should handle the IEEE 802.11b standard it may also be suitable to accelerate the modified Walsh transform butterfly (essentially a radix-4 FFT butterfly without multiplications). However, because of the high symbol rate in IEEE 802.11b it may be more suitable to use function level acceleration for the modified Walsh transform.

Data width optimization: The requirements on data precision is generally low in baseband processing. As discussed in section 3.5.6, significant reduction of hardware size and power consumption can be achieved by keeping down the data width. The BBP1 baseband processor, presented in chapter 7, uses 16 bit native data width (16 bits for real and 16 bits for imaginary part) and 12x12 bit precision for multiplications. This was proven to be sufficient by behavioral simulations of IEEE 802.11a and b systems.

5.2.2 Bit-Based Processing

Many of the processing steps between the interface to the MAC-layer and the mapping/demapping are operations which are based on manipulation of individual data bits (e.g. channel coding, scrambling, interleaving, and CRC checksum calculation). This tends to make them rather inefficient for software implementation. On the other hand, many of these operations can be implemented in very small hardware blocks. For example, both scrambling, convolutional coding, and CRC can be implemented as simple linear feedback shift registers with XOR-gates. This fact often makes these operations suitable for function level acceleration. These operations are also rather similar between different standards, which makes it beneficial to build configurable accelerators.

The required computing cost is on the other hand not extremely high, implying that software implementation may be suitable in some cases with low or moderate data rates.

5.2.3 Error Correction

Error correction (e.g. Reed-Solomon, Viterbi, and turbo decoders) is generally very demanding. Instruction level acceleration has been suggested, e.g. Galois field operations for Reed-Solomon [1] and add-compare-select instructions for Viterbi [2]. However, even with such instructions the MIPS cost is still high and in most cases these operations should be accelerated on function level. Again, good possibility of reusing such accel-

erators in several standards makes function level acceleration even more attractive.

5.3 Real-Time Requirements

Essentially all radio systems we are interested in are hard real time systems. Transmitted samples must be sent to the ADC at exactly the right clock cycle and incoming data must be taken care of.

In many cases the latency requirements are also very though. One example is the latency requirement for acknowledging a received packet in a WLAN system. As mentioned earlier, broadcasting systems such as DAB/DVB where latency is not a big issue are not considered here.

Implications of this is for example that regular cache memories or branch prediction cannot be used because of the unpredictability in execution time it introduces, and that interrupt latencies need to be kept short.

During the course of this project it was found that neither of these factors were limiting. The required memory sizes are quite small since the latency requirements does not allow a lot of data to be buffered in the baseband processor. For example, in the BBP1 processor the total amount of memory is 65 kbit for program and 100 kbit for data. The use of caches is of no interest with such small memory sizes. Furthermore, since the processing flow is to a large extent known, only rather simple control features and interrupt handling are sufficient, which simplifies reaching low interrupt latencies.

The challenge was rather to find an architecture that allowed efficient scheduling and communication between the DSP core and accelerators. Once sufficient computing power and low communication overhead was reached remaining problems were minor.

It should be noted though that detailed scheduling is necessary to determine the real hardware requirements. Simply using the average MIPS cost is not sufficient. This is due to e.g. dependencies within received data. In many cases, heavy processing is required during the preamble

of a frame in order to perform synchronization and channel estimation. These operations must be completed before processing of the data symbols can start.

5.4 Memory Issues

As mentioned above, the required amount of data memory tends to be rather small in baseband processing. On the other hand the memory access rate tends to be quite high. This means a significant total memory bandwidth is required if a moderate clock frequency is to be used.

Example 1: IEEE 802.11a is an OFDM system using 64-point FFTs. One FFT+channel equalization requires a total of 1152 memory access. Each access reads or writes a complex value of 16+16 bits. The symbol time is 4μs resulting in $1152 * (16 + 16)/4 = 9212$ bits accessed per microsecond. If the processor runs at 160 MHz this means an average of $9212/160 = 58$ bits per clock cycle just for the FFT and channel equalization during payload processing. Memory access rates for various systems has been further studied in [3].

Example 2: The BBP1 processor has a dual complex MAC unit. To keep the MAC fully occupied, up to 4 complex values needs to be read and two written every clock cycle, resulting in a required memory bandwidth of $(16 + 16) * (4 + 2) = 192$ bits per clock cycle for the MAC unit only. At the same time incoming data from the ADC have to be buffered and memory may also be accessed by accelerators. On the other hand, the total amount of data memory needed for 802.11a reception (including FFT coefficients and housekeeping variables etc.) is only 18.7 kbit, or equivalent to 585 samples.

The conclusion of these examples is that a typical programmable baseband processor will have multiple memory blocks of relatively small size. Thanks to the regular access patterns and well known scheduling in baseband processing, dual port memories with their extra area and power cost can typically be avoided. Though dual port memories may simplify design, solutions using standard memories with smaller power and area,

and without increased cycle cost can usually be found. The memory issue is further discussed in section 7.5.

Bibliography

[1] H. Michel Ji, "An optimized processor for fast Reed-Solomon encoding and decoding," in *Proceeding of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. III–3097–III–3100, May 2002.

[2] Jeong Hoo Lee, Weaon Heum Park, Jong Ha Moon, and Myung H. Sunwoo, "Efficient DSP architecture for Viterbi decoding with small trace back latency," in *Proceeding of the IEEE Asia-Pacific Conference on Circuits and Systems*, pp. 129–132, Dec. 2004.

[3] Anders Nilsson, Eric Tell, and Dake Liu, "Design methodology for memory-efficient multi-standard baseband processors," in *Proceedings of Asia-Pacific Communications Conference (APCC)*, 2005.

Chapter 6
Related Work

6.1 Introduction

Not many academic publications are available in the area of programmable baseband processors. Most of the existing publications focus on small parts of the system or on high level issues and very few on new architectures.

For many of the commercial solutions it is difficult to obtain detailed information without a none-disclosure agreement. Nevertheless, this chapter briefly presents some other, mainly commercial, solutions.

6.2 Rice University - Imagine

Rajagopal et al. have published one of the few existing academic papers on programmable baseband processor architectures [1]. The architecture is a modification of a media processor architecture from Stanford called "Imagine" [2]. It has eight VLIW-based computational clusters organized in a SIMD fashion. Each cluster has three adders and three multipliers. The memory subsystem is stream based and optimized for media applications.

At 500 MHz it can run WCDMA multi-user-detection [3] (i.e. the process used in a base station for simultaneous channel estimation and rake

reception for several users) for 32 users under low or medium mobility.

6.3 Morpho Technologies - M-rDSP

Morpho Technologies provides the M-rDSP architecture[4]. It is based on an array of between 8 and 64 reconfigurable cells (RCs) controlled by a 32-bit RISC unit. Each RC contains an ALU, a MAC unit, and special functional units for wireless applications. The configuration of all RCs and interconnects is specified in a "context". A context memory that can store up to 512 different contexts allows the function of the RC array to be changed every clock cycle.

The MS1-16 IP core has 16 reconfigurable cells and WCDMA-specific blocks for sequence generation and interleaving. Running at 500 MHz it supports WCDMA, GSM/GPRS, and IEEE 802.11a/b/g. The silicon area is claimed to be 65% less and estimated power consumption 50% less than other fully programmable DSP solutions.

6.4 Sandbridge Technologies - Sandblaster

Sandbridge Technologies Inc. has a processor architecture called "Sandblaster" [5]. The processor has a RISC based integer unit and a SIMD unit. The Sandblaster uses a form of multithreading. The core holds contexts for up to 8 threads simultaneously. The treads take turns to issue instructions in a round-robin fashion. Every clock cycle one of the treads can issue one instruction. When a thread only issues one instruction every eight clock cycles, data dependency problems due to the deep pipelines and memory latency are avoided. The Sandblaster architecture is intended to handle both media and baseband processing.

The SB9600 chip contains four Sandblaster processors and an ARM micro controller. Each Sandblaster core has 64 kByte of data memory divided into 8 memory banks and a 64 kByte instruction cache. There is also a 256 kByte level two cache for each core. The SB9600 supports 2Mbit/s WCDMA, GPS/GPRS and IEEE 802.11b running at 600 MHz.

Low-power circuit techniques and fine-grained clock gating is used to achieve a power consumption of less than 500 mW.

6.5 SystemOnIC - Hipersonic

SystemOnIC, now acquired by Philips, developed a programmable baseband processor called "Hipersonic 1" [6]. Hipersonic 1 is designed for the OFDM standards IEEE 802.11a and HIPERLAN/2. It is based on a flexible DSP core called OnDSP. It has one general data path and 8 SIMD data paths, each including a 40-bit ALU, a barrel shifter, and MAC unit with a 40-bit accumulator. The data paths are connected to a 128-bit-wide data memory via an interconnect unit. The general data path, memory, interconnect unit, and SIMD data paths are controlled using VLIW instructions. A dynamic code-width reduction technique called tagged VLIW [7] is used to reduce program size. A small instruction cache (loop cache) is also used.

Transmit and receive filters, channel coding (convolutional encoding, scrambling, CRC and Viterbi), and DES (Data Encryption Standard) encryption/decryption is handled by a hardwired coprocessor. A DMA unit handles data transfers between the DSP core and the coprocessor.

A chip including the Hipersonic core and ADC/DAC has a transistor count of 7.6 million. The chip has 32 kByte of program memory and 15 kByte of data memory. Extensive clock gating is used and the maximum power consumption is 300-800 mW running at 120 MHz in a 0.18 μm CMOS technology.

6.6 Other Solutions

A common solution for reaching some flexibility is to use a general processor or DSP together with ASIC blocks. Usually the ASIC block has low flexibility and most parts are standard specific. The programmable processor will mostly execute the control flow and very little of the data processing.

FPGA based solutions are discussed occasionally. These are to expensive and power consuming for most applications. Reconfiguration time is also a problem.

Another alternative is configurable architectures [8]. These typically consist of a configurable fabric of computing elements which are controlled by a programmable processor. The advantage compared to FPGAs is that they can be reconfigured dynamically at run-time. However, the reconfiguration time is still typically in the order of micro seconds which makes it difficult to reuse computing units for more than one task in a transmitter or receiver flow. So far the power consumption also tends to be high.

6.7 Discussion

The solution from Rice university is just a minor modification of a processor designed for media applications. The performance for bit oriented tasks, e.g. Viterbi decoding, is low. It seems more focused on base station processing and area and power efficiency is expected to be low.

The Sandbridge and Hipersonic solutions are in principle based on well known VLIW and SIMD solutions. The solution from Morpho technologies seems more innovative. Unfortunately, no detailed information about e.g. the interconnect and memory organization has been found.

None of the presented solutions use natively complex data paths, although it may be possible to execute a complex operation in one clock cycle by combining VLIW/SIMD execution units.

The Hipersonic processor has a SW-HW partitioning similar to the one proposed in this thesis while the other architectures rely more on wide SIMD/VLIW data paths even for bit oriented processing. None of these projects have any focus on flexible and efficient integration of hardware accelerators.

It is difficult to make quantitative comparisons between architectures since the function coverage varies. Nevertheless, it seems very likely that the BBP1 processor presented in chapters 7 and 8 is more area and power

efficient than all of the architectures described in this chapter, with the possible exception of the solution from Morpho technologies. However, the other architectures may have more flexibility outside the baseband processing domain. The more regular architectures may also simplify firmware design and design tools. Sandblaster is programmed in standard C while Hipersonic uses "Systemonic-C" which is described as a "C++ based high level assembler".

Bibliography

[1] Sridhar Rajagopal, Scott Rixner, and Joseph R. Cavallaro, "A programmable baseband processor design for software defined radios," in *Proceedings of the 45th Midwest Symposium on Circuits and Systems*, pp. 413–416, 2002.

[2] B. Khailany, W. J. Dally, U. J. Kapasi, P. Mattson, J. Namkoong, J. D. Owens, B. Towles, A. Chang, and S. Rixner, "Imagine: Media processing with streams," *IEEE Micro*, no. 2, pp. 140–147, 2001.

[3] S. Rajagopal, S. Bhashyam, J. R. Cavallaro, and B. Aazhang, "Real-time algorithms and architectures for multiuser channel estimation and detection in wireless base-station receivers," No. 3, pp. 468–479, 2002.

[4] *Morpho Technologies*. http://www.morphotech.com/.

[5] John Glossner, Daniael Iancu, Jin Lu, Erdem Hokenek, and Mayan Moudgill, "A software-defined communications baseband design," *IEEE Communications Magazine*, no. 1, pp. 120–128, 2003.

[6] Johannes Kneip, Matthias Weiss, Wolfram Drwscher, Volker Aue, Jürgen Strobel, Thoms Oberthür, Michael Bolle, and Gerhard Fettweis, "Single chip programmable baseband ASSP for 5 GHz wireless LAN applications," *IEICE Transactions on Electron.*, pp. 359–367, Feb. 2002.

[7] Matthias Weiss and Gerhard Fettweis, "Dynamic codewidth reduction for VLIW instruction set architectures in digital signal processors," in *Proceedings of IWISP*, 1996.

[8] Srikathyayani Srikanteswara, Ramesh Chembil Palat, Jeffrey H. Reed, and Peter Athanas, "An overview of configurable computing machines for software radio handsets," *IEEE Communications Magazine*, no. 7, pp. 134–141, 2003.

Part III

The BBP1 Architecture

Chapter 7
The BBP1 Baseband Processor Architecture

7.1 Introduction

This chapter will describe the main contribution of the presented work; the baseband processor architecture. The main goals set up at the start of the project can be summarized as follows:

- Find an efficient instruction set for baseband processing.
- Find which functions in a baseband processor are suitable for hardware acceleration and especially if it is possible to build flexible hardware accelerators that can be reused between standards and functions.
- Find an efficient baseband processor architecture focusing on
    1. Low computing latency and predictable hard real-time performance.
    2. Minimized memory size and memory access overhead.

Meeting these goals would contribute greatly in reaching the ultimate goal of finding power and area efficient architectures for programmable multi-standard baseband processors.

The main achievements resulting from this project are:

1. The DSP core and its instruction set, including a novel type of vector instructions.
2. The overall architecture, using a configurable network to connect the accelerators and memories to the processor.
3. HW-SW codesign for IEEE 802.11a/b. Mapping and scheduling of these standards on the architecture.

The first two points are intimately connected since the efficient execution of vector instructions relies on the network design and especially the memory subsystem with a number of small data memories with decentralized address generation.

This chapter will give an overview of the architecture followed by a more detailed description of the novelties of the instruction set, the memory subsystem and the integration of accelerator units in the baseband processor. Sections 7.2 - 7.6 refers primarily to the first implementation of the architecture which was a baseband processor focused on WLAN applications. Much of these details have previously been published [1, 2]. Section 7.7 will describe some possible variations of the architecture as well as some scalability issues.

Chapter 8 discusses the demonstrator chip which was manufactured as well as the developed design tools (assembler and simulator) and firmware for IEEE 802.11a and IEEE 802.11b[1].

Appendix A contains a summary of the algorithms used in the WLAN applications and a quantitative motivation of ISA design choices based on the MIPS costs and timing requirements of the standards.

7.2 Architecture Overview

Figure 7.1 gives an overview of the architecture. The core of the processor is an application specific DSP processor with a simple ALSU (arithmetic-

[1] In the rest of this thesis the IEEE 802.11a and IEEE 802.11b standards will sometimes be refered to as just 11a and 11b

logic-shift unit) and a specialized complex-MAC unit. The DSP core is connected to a number of memories and accelerators via a configurable network.

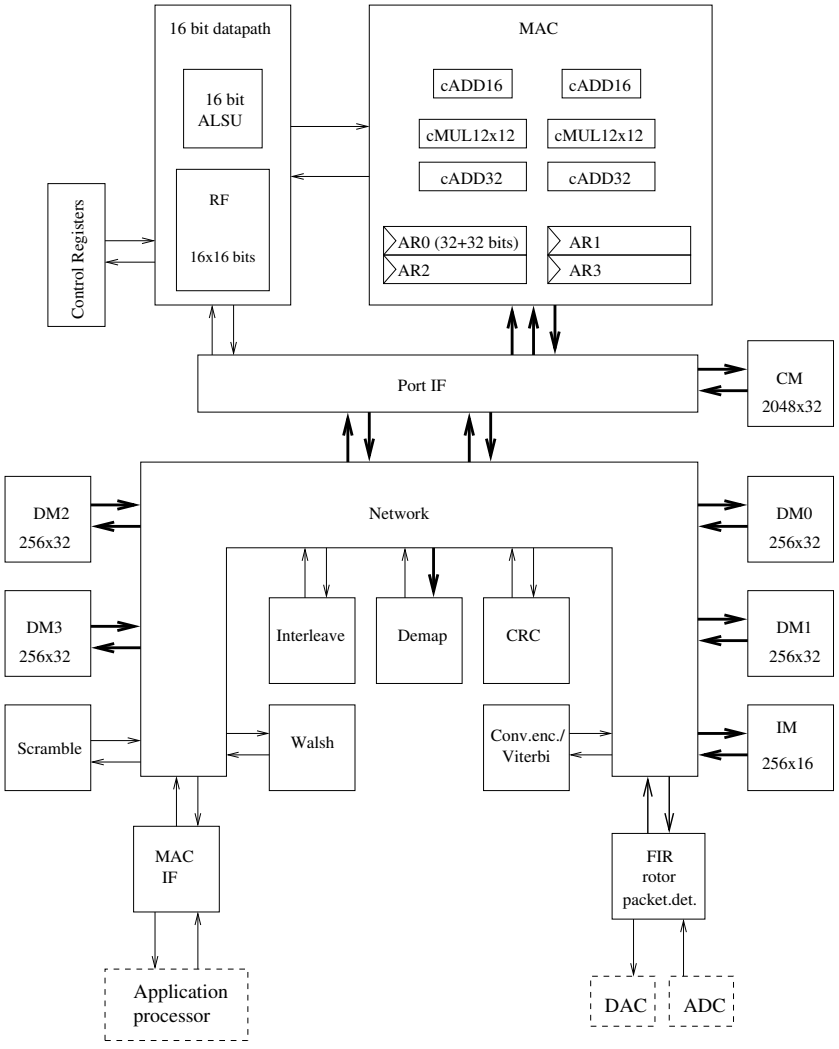

Figure 7.1: Overview of the BBP1 architecture

7.2.1 The Network

The network design allows a large number of concurrent communications, allowing a high degree of parallelism in the system. It also minimizes the number of memory accesses by letting accelerators pass data between each other without intermediate memory storage and by enabling "swapping" of entire memories between different units by reconnection of the network. The network is a simple crossbar, configured entirely by the processor core. This eliminates the need for an arbiter and addressing logic and makes the network interfaces simple.

7.2.2 The DSP Core

The DSP core has two separate data paths. The first one is a rather simple data path consisting of a 16-bit ALSU and sixteen 16-bit general purpose registers. This data path executes RISC-style instructions mostly used for control flow, integer arithmetics, and configuration tasks. The main feature of the DSP core is the dual complex MAC unit which is optimized for execution of the operations on vectors of complex numbers (I/Q-pairs) common in baseband processing. The MAC unit executes a novel type of vector instructions, further described in section 7.3.

Instruction Set

Minimization of the program size was one of the main design goals and an efficient instruction set is obviously essential in order to reach this goal.

The instruction set can be divided into three classes of instructions:

- RISC-style instructions for both real and complex data.
- Instructions for network and accelerator configuration.
- Vector instructions.

All instructions are 16 bits wide, resulting in very efficient use of program memory. Figure 7.2 shows the instruction encoding. The numbers within parentheses are the number of bits in each field. Together with the

vector instructions and acceleration of selected functions this results in very small program memory requirements (see table 8.1).

Normal instruction (4 different formats):

0	subtype (2–5)	instr. (2–6)	arguments (4–11)

Vector instruction:

10	instruction (4)	ports (3)	vector size (7)

Accelerator instruction:

11	accelerator ID(4)	control vector (10)

Figure 7.2: Instruction encoding formats

Use of dedicated instructions for configuration of the network and accelerators will reduce the overhead for such tasks compared to using memory mapped registers or similar for configuration. Considering the extent to which the architecture relies on efficient integration of accelerators and how a baseband application typically would make use of the possibility of swapping memories between units, the small extra complexity of the control path due to these special instructions can certainly be justified.

Much of the configuration can also be carried out via a control register file. The control register space is also used for e.g. data path and interrupt configuration, and for communicating status information from accelerators and other hardware components.

Pipelining

All non-vector instructions use three pipeline stages (fetch, decode, and execute), except instructions reading from the network which use two execution stages. Vector instructions have 5-8 pipeline stages.

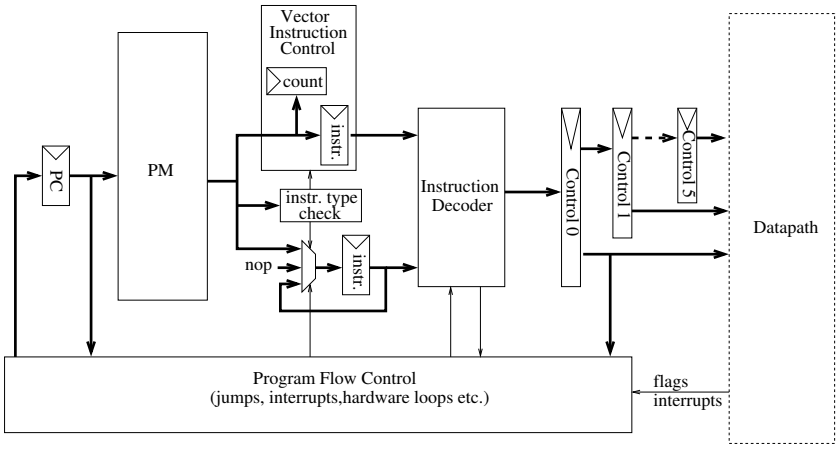

Figure 7.3: The control path

Control Path Features

An overview of the control path of the processor can be found in figure 7.3. The control path complexity is similar to what can be found in simple DSP processors or micro controllers but with the addition of the vector instruction control unit which handles the multi cycle vector instructions. The control path has been kept very simple to eliminate control overhead which is typically quite substantial in enhanced processor types such as VLIW and superscalar processors. The vector instruction control unit adds only little extra hardware. Much of the control path originates from a previously developed extensible DSP processor [3].

Features of the control path include a zero overhead hardware loop instruction and hardware PC and loop stacks, allowing fast interrupts, subroutine jumps, and nested hardware loops. Operand stopping is used to reduce power consumption.

The processor uses fast interrupts, meaning that only PC and status registers are saved on interrupts. Data registers will have to be saved manually. This simple approach was chosen because it allows very short interrupt latency (five clock cycles in the worst case) and small hardware cost. It was found that most interrupt handling routines in the applica-

tions of interest are simple tasks such as reconfiguration of the network or some of the accelerators, which does not require context switching.

The processor uses programmable interrupt vectors (i.e. the interrupt address for each interrupt source is programmable) for up to 8 separate interrupt sources. This includes two external interrupt pins, timers, interrupts from accelerators, and a special interrupt signaling the completion of a vector instruction. Accelerators may generate interrupts after completing a task, after processing a certain number of samples or on events such as packet detection (see section 7.6.7).

Pipeline conflict checks are not done by the hardware. This means that undefined execution results may occur if illegal instruction sequences are used (e.g. executing an instruction using the MAC unit when a vector instruction is being executed). All such checks are instead done automatically by the instruction set simulator during software development.

Although some dependency checking was implemented in the previous processor [3] it was not implemented in BBP1, mainly to save implementation and verification time. Such features are typically not as important in embedded processors since the applications are usually well known and extensively simulated before deployment.

Register forwarding features are are of little interest in BBP1 since (1) RISC instructions only have a single execution cycle, so the result is available for the next instruction anyway and (2) for network accesses it would increase the critical path as long as the network is not further pipelined. The only situation where it may be applicable is when a vector instruction is followed by a data move from accumulator to general register.

7.2.3 The MAC Unit

The MAC unit is the most substantial part of the DSP core. It is optimized for operations on vectors of complex numbers. Since such a significant fraction of the convolution/multiply-accumulate based operations in baseband processing is complex valued it was found beneficial to optimize the MAC unit for such computations even at the expense of

slightly larger overhead for real-valued computations. Figure 7.4 depicts the MAC unit.

The unit contains two complex 12x12 bit multipliers (i.e. eight real multipliers), two 32-bit complex adders (for accumulation 8 guard bits and 8 precision bits are added to the 16 native bits), and two 16-bit complex adders. It has four 32+32 bit accumulator registers. It can carry out for example two complex multiply-accumulate operations or one radix-2 FFT butterfly each clock cycle. It also supports scaling, rounding, and saturation of the result.

7.3 Vector Instructions

The introduction of vector instructions is the single most important novelty in the design of the processor core. Through their introduction we reach a higher efficency, smaller program size, and increased parallelism during execution of complex vectors operations, such as vector addition, scalar products, correlation, FIR-filtering, FFT, vector absolute maximum search, etc.

The vector instructions have the following properties:

- They operate on vectors of complex numbers. The vector size can be any number between 1 and 128 and is given explicitly in the instruction.

- The operand vectors are normally read from memory but may also come directly from an accelerator or external interface (see further section 7.4). The result is stored in memory or sent to an accelerator or external interface, unless the result is scalar (i.e. the result of a scalar product or max search) in which case it is stored in one of the four accumulator registers.

- Vector instructions require multiple clock cycles to complete, depending on the vector size. However other instructions not using the MAC unit may execute in parallel with the vector instruction.

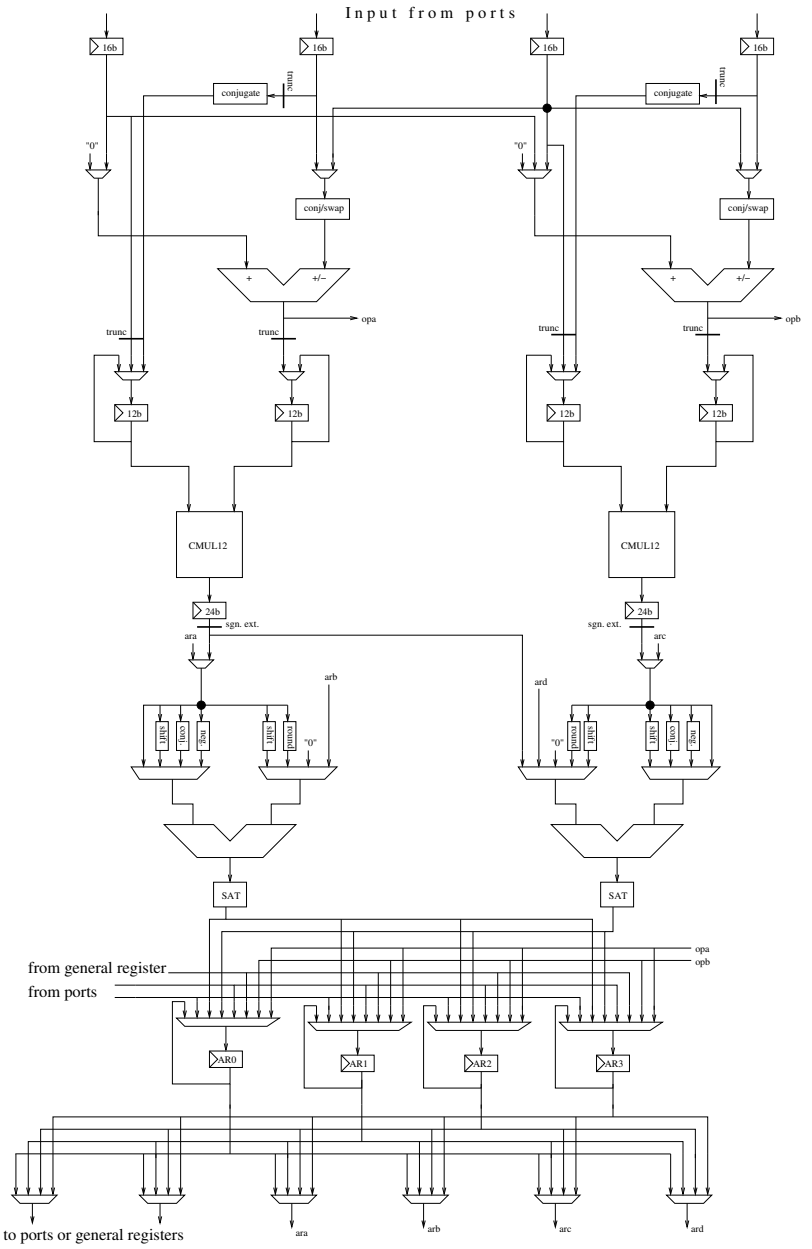

Figure 7.4: The MAC unit

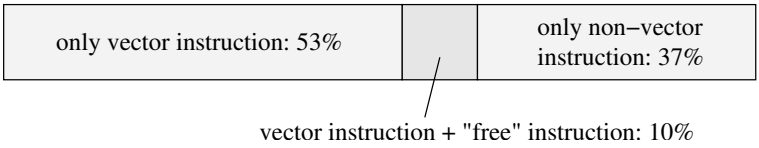

Figure 7.5: Fraction of the execution clock cycles spent on vector instructions and other instructions

```
FFT.64          port1,port0     ; last layer of 64-point FFT
ADD             R0,R1           ; “free” control flow instructions
MVR2CR          R1,CR25
ACL             dm0,read|0x00   ; setup addressing for...
ACL             dm0,rstep|2     ; ...square abs max search
IDLE            mac             ; wait for FFT vector instruction to finish
SQRABSMAX.64    port2,AR0       ; find square absolute max value
```

Figure 7.6: Illustration of instruction parallelism. FFT.64 and SQRABSMAX.64 are vector instructions

The fact that other instructions, such as integer arithmetics and accelerator and network configuration, can execute in parallel often allows part of the cycle cost for control flow code to be hidden behind vector instructions. This is illustrated in figure 7.6. Note that although vector instructions and RISC instructions execute in parallel, they are not *issued* in parallel. In other words, during the clock cycle the vector instruction is *started*, no other instruction can execute. The first RISC instruction is executed in the *next* clock cycle. This is illustrated in figure 7.7. It is this fact that allows us to keep a very simple control path, instead of resorting to a VLIW or superscalar solution.

Statistics collected from the execution of the implemented firmware showed that approximately 20% of all non-vector instructions were hidden and executed without any extra cycle cost. Figure 7.5 shows the fraction of the execution cycles spent executing vector instructions and non-vector instructions. Table 7.1 gives some examples of vector instructions and their use.

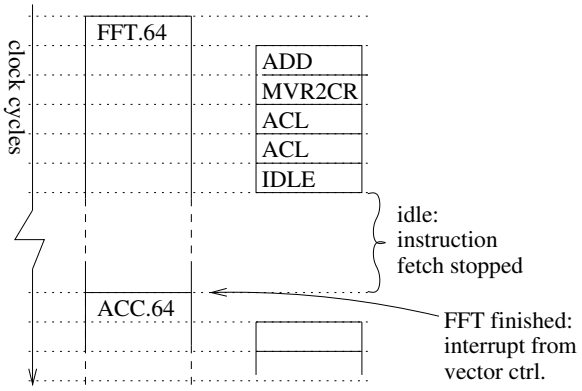

Figure 7.7: Timing for execution of the instructions in the code example from figure 7.6

Vector Instructions vs. Hardware Loops

At a first glance the vector instructions may seem similar to single instruction hardware loops which are available in many general DSP processors. However, there are a few important differences.

The most obvious difference is perhaps that no dedicated hardware loop or repeat instruction is needed (saving one line of program code and one clock cycle). More significant is however the possibility to execute other instructions in parallel as described above. As a natural extension to this concept it is also possible to add additional execution units with their own vector instruction control units, thereby allowing several vector instructions to execute in parallel on different execution units.

Another point worth mentioning is the fact that the vector instruction control unit may implicitly handle any prolog/epilog processing, i.e. any special conditions occurring at the start or end of a loop. One example of this is the last step of a vector instruction of odd size, in which only one operation is executed instead of two parallel operations which is the normal case. This feature becomes even more important if the execution units are made wider in order to further increase the computing capacity. The two latter of these points are further discussed in section 7.7.

Table 7.1: Examples of vector instructions

MAC	Scalar product, correlation, FIR filtering
VADD	Vector addition
VMUL	Elementwise vector multiplication, multiply vector by scalar
FFT	One layer of fast Fourier transform butterflies
FFT2	Includes frequency domain filtering in last layer of and FFT, e.g. for OFDM channel equalization
SQRABS	Vector elementwise square absolute value
SQRABSACC	Sum of square absolute values, vector energy
SQRABSMAX	Find value and position of maximum square absolute value

7.4 The Accelerator Network

Most components are connected to the DSP core via the network. This includes data memories, accelerators, and external interfaces (ADC, DAC and MAC-layer interfaces). All components have essentially the same interfaces (although data width may vary) and look the same to the processor core.

The network behaves like a crossbar switch which is configured entirely by the core by way of dedicated instructions. This eliminates the need for arbitration and addressing logic thereby reducing the complexity of the network and the accelerator interfaces while still allowing many concurrent communications. Since some units will never need to communicate directly, the complexity of the network can be further reduced by applying restrictions to what connections are possible.

Each network port consists of a pair of one read port and one write port. A connection is set up by connecting one read port to one write port. The reading unit requests one word of data by asserting a *ReadRequest* signal during one clock cycle and the transmitting unit uses a *DataAvailable* signal to indicate that new data is available on the port. The requesting unit may have up to two outstanding read requests, but must then halt

if no data available signal is received. This simple protocol allows a new data item to be communicated every clock cycle, but still provides sufficient flow control. Figure 7.8 shows examples of network transactions.

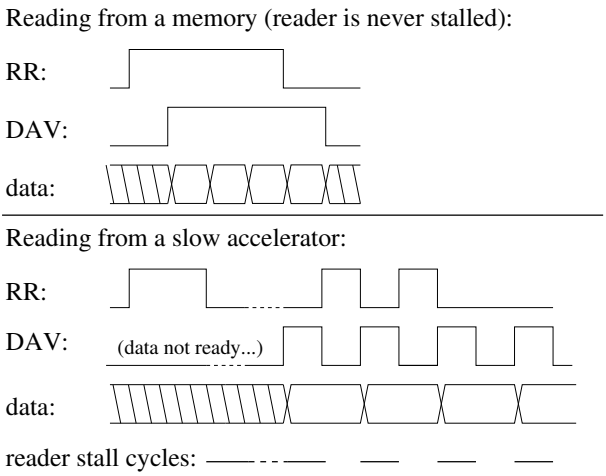

Figure 7.8: Network transaction examples

The network in BBP1 has 16 ports, two of which are connected to the core, five to data memories, and the remaining to accelerators or external interfaces.

7.4.1 Accelerator Chaining and Function Level Pipelining

A chain of network components connected together will automatically synchronize and communicate without any interaction from the processor core. This is a very important feature which allows truly concurrent operation of the core and any number of accelerators. It also reduces the number of memory accesses since no intermediate memory storage is needed when sending data between accelerators.

Accelerator chaining also enables a type of function level pipelining. This is illustrated for an OFDM receiver in figure 7.9. At the same time the core is computing the FFT for one symbol, the previous symbol could be processed by a chain of accelerators for demapping, interleaving, and

channel decoding, while the next symbol is received, decimated, and derotated by the front end accelerator (see 7.6) and stored in memory.

Synchronization between the pipeline steps can be achieved by interrupt signals from accelerators upon completion of tasks, by timers, or by a sample counter feature in the front-end accelerator.

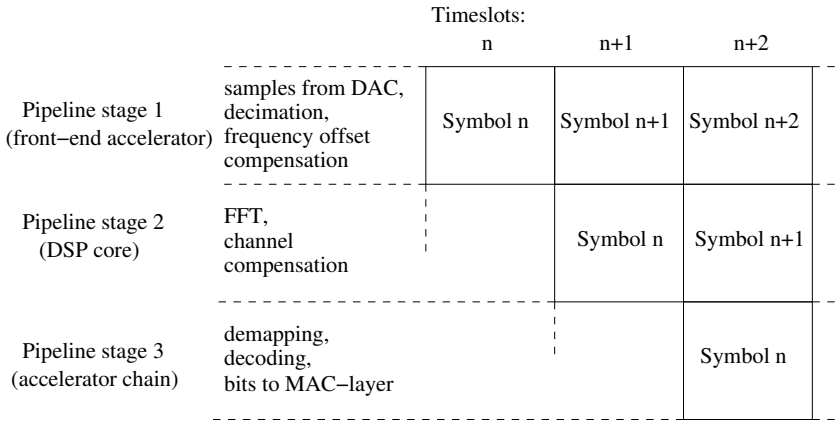

Figure 7.9: Function level pipelining

7.5 Data Memory Architecture

As mentioned earlier, reducing the amount of memory and memory accesses was a major goal in the design. Using a number of small data memories gives enough memory bandwidth to keep the core and MAC fully occupied while simultaneously buffering incoming data from the radio front-end interface and feeding accelerators or the MAC-layer interface.

Since the memories are so small and thereby fast, there is no need for cache memories. Thereby a major source of control overhead and unpredictability is eliminated. The network always gives a unit (core or accelerator) exclusive access to a memory. This eliminates stall cycles due to access conflicts and gives a highly predictable architecture.

Another important feature of the architecture is the possibility of switching memories between units. In other words, when e.g. the DSP core has

finished a computation on a block of data, the entire memory containing the result can be "handed over" to another unit by reconfiguration of the network. This eliminates data moves between memories and almost eliminates the communications overhead in terms of core clock cycles for sending data between core and accelerators. The only remaining overhead is typically 1-2 clock cycles for reconfiguration of the network and 1-4 clock cycles for configuration of the memory addressing.

7.5.1 Address Generation

Address generation is carried out by address generation logic in each memory block. This means that no addressing information has to be sent over the network. It also eliminates the need for address generators in the accelerators.

In the simplest case, the address generation unit in a memory block is initialized by setting a base address and an increment register. In most cases the increment register value will be one, e.g. for fetching consecutive elements of a vector one at a time, or two if two vector elements are fetched in parallel, see below.

All memories in BBP1 also support modulo addressing for implementation of circular buffers. The four complex data memories DM0-DM3 also support bit-reversed addressing for FFTs.

To increase the memory bandwidth all memories except IM consists of two interleaved memory banks. Thus all odd addresses are in one bank and even addresses in the other, allowing consecutive addresses (vector elements) to be accessed in parallel. The implementation of the interfaces for these units are essentially two network ports with shared control logic (i.e. both data, ReadRequest, and DataAvailable signals are doubled, but the control logic is not).

One argument against each memory having its own address generator is that accelerators may use special addressing schemes and then it would be more natural to have the address generation in the accelerator instead. Otherwise each memory block that could potentially commu-

nicate with the accelerator would need to support a special addressing mode. However, it turned out that in fact all accelerators in BBP1 only uses post-increment addressing with an increment of one, i.e. always reading/writing consecutive vector elements one at a time. Therefore the used solution is more beneficial.

One operation that typically is associated with irregular addressing is interleaving. However, since this is a bit level operation it is much more efficient to implement the interleaving with an entirely specialized memory structure, i.e. the interleaver accelerator will have its own memory, designed specifically for interleaving, and the input and output to the network will still use linear addressing. Such a block has been described earlier [4].

A similar case would be an accelerator for FFT which may be a good idea for OFDM based standards using very large FFTs, e.g. DVB. In that case it is also believed that a better idea would be to have dedicated customized memory blocks in the FFT accelerator, thereby eliminating the need for FFT addressing for other memories. One possible design of an accelerator for FFT, DCT, and Walsh transform (used at the higher data rates in 11b) has also been described [5].

One final example, which is of interest especially in CDMA systems, is addressing in a programmable solution for rake receivers. This was discussed in [6].

Another solution is to use special address generator components connected to the network [7]. This is however not suitable in the presented architecture due to increased latency, increased number of required network ports, and increased network traffic among other things.

7.6 Function Level Acceleration for BBP1

This section will briefly describe the hardware accelerators used in BBP1. Acceleration decisions are based on MIPS costs (see appendix A) and estimated hardware cost for each function in the IEEE 802.11a and b standards.

7.6.1 Channel Coding

IEEE 802.11a uses a convolutional encoder for channel coding and the Viterbi algorithm for decoding. Puncturing is used to achieve different code rates. Both these functions are good candidates for acceleration, as stated in section 5.2. Since convolutional encoding and Viterbi decoding never run simultaneously the two functions are built into the same accelerator unit in order to save network ports and configuration logic.

7.6.2 Scrambling

To cover 11a and 11b scrambling and descrambling, three different modes are needed (since for 11a scrambling and descrambling are identical). The three different functions are very similar. A multi-mode scrambler can easily be designed and uses very little hardware. Figure 7.10 shows the kernel logic of the scrambler.

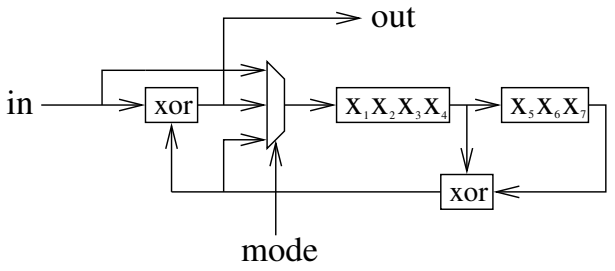

Figure 7.10: Combined IEEE 802.11a/b scrambler/descrambler

7.6.3 Interleaving

IEEE 802.11a uses a block interleaver. The block size is equal to one OFDM symbol, i.e. between 48 and 288 bits depending on the data rate. The multi-mode block interleaver described in [4] is used.

7.6.4 Demapping

Demapping is rather cycle consuming for large constellations such as 16-QAM and 64-QAM and acceleration is needed to reach enough throughput at the highest data rates for 11a. The demapper is a simple unit based on four small adders.

7.6.5 Walsh Transform

A modified Walsh transform plus absolute maximum value search are the kernel operations of the CCK demodulation used in 11b. Acceleration of these steps decreased the required clock frequency for 11b reception at the highest data rate by approximately 60 MHz. The hardware in the implemented accelerator contains 15 adders and and two multipliers making this the second largest accelerator after the Viterbi decoder.

7.6.6 CRC

CRC check sums are used for the 11b header and for the MAC frames (the MAC layer is the same for both standards). The cycle cost for software implementation would not be very high if only the 11b PHY checksum was implemented, but it does require a substantial amount of data memory for look-up table storage. The core of the CRC hardware is similar to the scrambler and consists of a feedback shift register and three XOR-gates.

7.6.7 Front-End Accelerator

The front-end accelerator contains the interfaces to ADC and DAC, a configurable FIR filter that can be used for symbol shaping or decimation, a rotor (essentially an NCO and a complex multiplier) used for frequency offset compensation, and a packet detector based on auto correlation used to wake the processor from sleep mode when an incoming packet is detected. These functions are used by many standards, runs much of the time, and especially the filtering can be rather demanding.

7.7 Design Variations and Scalability Issues

7.7.1 Scalability - Increasing Computing Capacity

As mentioned in section 7.3, a natural extension of the vector instruction concept would be to add more execution units with their own vector execution control. It would then be possible to execute multiple vector instructions in parallel. The cost of this, apart from the extra execution units themselves, would be the additional vector execution controllers and network ports. The cost of the vector execution control unit is very low and increasing the network size is also not believed to be a problem in a practical case, see section 7.7.3.

Another way of increasing computing capacity would be to increase the width of the execution units, e.g. using a 4-way complex MAC unit instead of a double complex MAC. The two main complications this would introduce is: (1) Increased memory bandwidth requirements to keep the unit fully occupied and (2) more complex prolog/epilog processing.

The first of these problems could be handled either by increasing the number of memory blocks, which would imply that the number of network ports would also be increased, or by making the memories wider. The latter solution seems more attractive, since it does not increase the logical depth of the network and gives lower control complexity. The slightly reduced flexibility with that solution is not believed to be an issue since vector elements are most often accessed sequentially.

The more complicated prolog/epilog conditions would lead to a slightly more complex vector execution control unit. Note however that this complexity otherwise would have to be handled explicitly in the program by extra assembly instructions (as in a VLIW or SIMD processor) and/or conditional execution.

A possible configuration for future implementations of this architecture was presented in [6]. Here the MAC unit has been extended to a 4-way unit and a 4-way complex ALU unit primarily optimized for despread and rake processing has been added. This configuration will support most current OFDM and CDMA standards.

7.7.2 Simultaneous Multi-Standard Processing

One issue that was not addressed in BBP1 is the possibility of handling standards using full duplex or even running multiple standards simultaneously. In a practical case it is not obvious that one actually will want to run multiple standards simultaneously on the same processor, mainly due to the scheduling and certification issues mentioned below. A multiprocessor solution in which all simultaneously active baseband applications run on separate processors would simplify things in both these aspects. Nevertheless, the following issues have been identified and may be considered in future work:

Fast Context Switching

As mentioned in section 7.2.2, the need for supporting context switching on interrupts was not found in the applications for which BBP1 was designed. However, in order to process several separate flows simultaneously, fast context switching on interrupts would be helpful. One possible solution is something similar to what is used for fast interrupts in the ARM architecture [8], i.e. to duplicate some or all of the general registers and swap to a new set of registers in a single clock cycle on interrupts.

Multiple Front-End Interfaces

The implemented radio front-end interface allows simultaneous reception and transmission and can therefore in principle handle the full duplex case. It is also possible to add several front-end interface blocks of the same type (or any other kind of interface for that matter) in order to handle simultaneous reception and transmission of multiple streams. In that case, more memory blocks would probably also be added for buffering in the different streams.

Scheduling Issues

One very important issue that has not been thoroughly investigated is scheduling. In the implemented firmware each standard has been individually statically scheduled by hand and the feasibility of the schedule has been proven in simulation. Scheduling of multiple simultaneously running standards is much more complex and dynamic scheduling may be necessary. Approaches to scheduling and the need for scheduling tools will be further investigated.

Certification

One issue that has come up in the context of simultaneous multi standard processing is that of certification. Any radio device must go through a certification process to ensure that it will behave correctly in the system and not interfere illegally with other devices. This is typically a rather time consuming and costly process. A multi-standard device must go through a certification process for each of the supported standards. One risk is that if the firmware of different standards are intertwined and running simultaneously on the same processor, a modification of the firmware for one standard would make it necessary to recertify the device for all implemented standards. The cost of this process may be prohibitively high.

7.7.3 The Network

In the implemented BBP1 demonstrator chip (see section 8.4) the network has 16 ports. This network constitutes approximately 10% of the logic area and has no critical signal paths. However, since the size of a full crossbar grows as the square of the number of ports, and the logic depth grows as the logarithm of the number of ports, a limit on the number of ports will eventually be reached with the current network architecture.

In practical cases this is not believed to be a problem since the number of ports will not be so much larger.

As previously discussed, the network complexity can be reduced by restricting the crossbar. One solution of interest would be to divide the

network into two separate networks: One handling complex valued data and one handling bit oriented data. The mapper/demapper accelerator and the core could act as bridges between the two networks.

Finally it should be mentioned that the next generation processors in this project will use fewer rather than more network ports. For example the possibility of building a general bit manipulation processor to replace both CRC, scrambling, channel encoding, and interleaving accelerators will be investigated.

Bibliography

[1] Eric Tell, Anders Nilsson, and Dake Liu, "A programmable DSP core for baseband processing," in *Proceedings of IEEE Northeast Workshop on Circuits and Systems (NEWCAS)*, June 2005.

[2] Eric Tell, Anders Nilsson, and Dake Liu, "A low area and low power programmable baseband processor architecture," in *Proceedings of the 5th International Workshop on System-on-Chip for Real-Time Applications (IWSOC)*, July 2005.

[3] Eric Tell, Mikael Olausson, and Dake Liu, "A general DSP processor at the cost of 23k gates and 1/2 a man-year design time," in *Proceedings of IEEE International Conference on Acoustics, Signal, and Speech Processing (ICASSP)*, pp. 657–660, Apr. 2003.

[4] Eric Tell and Dake Liu, "A hardware architecture for a multi mode block interleaver," in *Proceedings of the International Conference on Circuits and Systems for Communications (ICCSC)*, June 2004.

[5] Eric Tell, Olle Seger, and Dake Liu, "A converged hardware solution for FFT, DCT and Walsh transform," in *Proceedings of the International Symposium on Signal Processing and its Applications (ISSPA)*, pp. 609–612, July 2003.

[6] Anders Nilsson, Eric Tell, and Dake Liu, "A programmable SIMD-based multi-standard rake-receiver architecture," in *Proceedings of EUSIPCO*, Sept. 2005.

[7] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. M. Rabaey, "A 1-V heterogeneous reconfigurable DSP IC for wireless baseband digital signal processing," *IEEE Journal of Solid State circuits*, vol. 35, pp. 1697–1704, Nov. 2000.

[8] David Seal, ed., *ARM Architecture Reference Manual, 2nd ed.* Addison-Wesley, 2000. ISBN 1-201-73719-1.

Chapter 8
Implementation

8.1 Introduction

To prove the feasibility of the presented architecture, the programmable WLAN processor described in chapter 7 has been implemented together with design tools and firmware for IEEE 802.11a and IEEE 802.11b. This chapter will give a brief overview of these various parts.

8.2 Design Tools

An assembler and an instruction set simulator, as described in section 3.3, were developed as soon as the first draft of the instruction set was completed. These tools have later been updated according to the design iterations of the ISA as well as to add various features to increase their usability for RTL verification, firmware debugging, and benchmarking/profiling.

8.2.1 The Assembler

The assembler was implemented in C++ using the Flex lexical analyzer generator [1], and Bison compiler generator [2]. It is based on a generic assembler implemented by Dr. Daniel Wiklund.

8.2.2 The Instruction Set Simulator

The bit-true and cycle-true instruction set simulator was implemented in C++. The simulator features include:

- Standard debugging features for observability and controllability, breakpoints, etc.
- Data dependency checking (read-write, write-read, and write-write dependencies) and structural pipeline hazard checking. No such checks are performed in the hardware, as mentioned section in 7.2.2.
- Tracking of undefined data values.
- Possibility to load/save memory contents and input/output data in various file formats to simplify exchange of data with other applications, e.g. Matlab and RTL simulators.
- Possibility to run script files and use batch mode execution.

Organization of the Simulator

The organization of the simulator is illustrated in figure 8.1.

The bottom layer consists of completely hardware independent classes for bit true arithmetic operations on various data widths and tracking of undefined values. There are also classes for support of scheduling, dependency checking, and logging as well as base classes for defining operations and instructions.

The middle layer consists of one class that stores the complete state of all hardware except for accelerators, and classes for describing all operations that instructions can perform on the hardware. Each instruction is described by a class that defines which operations it performs in each pipeline stage. There are also classes describing different types of accelerators. Which accelerators should be instantiated and which network ports they are connected to can be defined at runtime.

The next layer is a simulator class with methods for controlling the hardware, execution of instructions, breakpoints, etc. Finally, a user in-

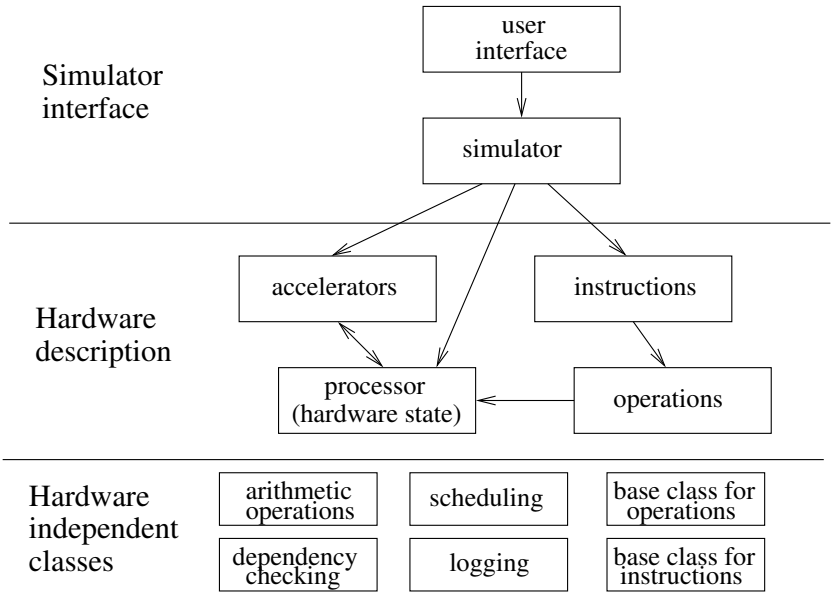

Figure 8.1: Organization of the instruction set simulator

terface layer is implemented to let the user interact with the system via the simulator class. The currently used implementation has a simple text based user interface, but a GUI is under development.

The simulator was designed to allow easy plug-in of accelerators and to be simple to reuse for future architectures.

Simulator Execution Flow

Figure 8.2 depicts the flow used in the simulator to achieve cycle-true execution. When an execution command is issued by the user the simulator will fetch the next instruction from program memory at the address given by the program counter (PC). The resource usage table of the instruction is first checked against already scheduled instructions. If any resource usage conflict is found an error message is printed and execution is stopped.

Otherwise data dependency checking is done. If a possible data dependency error is found, a warning is displayed (the most common case

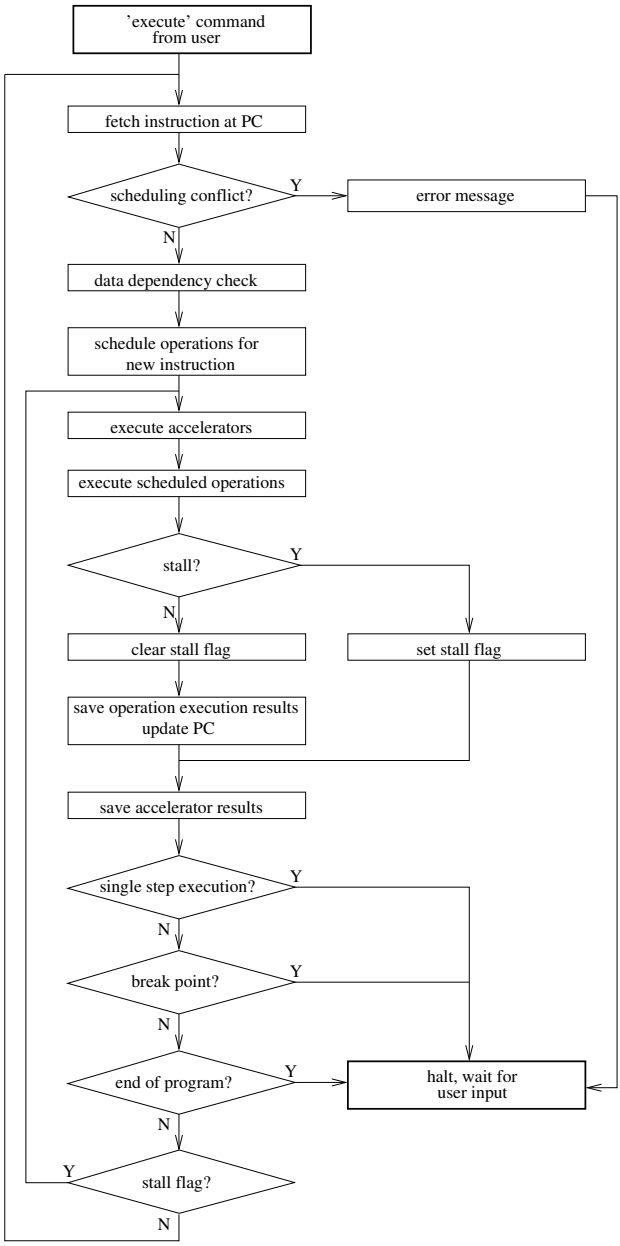

Figure 8.2: Simulator execution flow chart

is that the instruction tries to use a result of a previous calculation which is not yet completed). Next, all the sub-operations of the instruction is scheduled into their respective time slot (pipeline stage).

Each of the accelerators reads any input data and executes one clock cycle. The resulting output is not yet made available to other units, since they may still need data from the previous clock cycle. This way synchronization is kept.

Next all operations scheduled for the current clock cycle (belonging to one or more previous instructions) are executed. If any operation can not execute due to network delays a stall flag is set and no result from any operation will be saved. Instead all instructions will be delayed one clock cycle and operations that was executed the current clock cycle will be executed again during the next clock cycle.

If the processor was not stalled, the results of instruction execution are stored and PC is updated. The next PC value is a function of interrupt conditions, control flow instructions, hardware loops, and the status of the idle flag. Delayed jumps are implemented by scheduling a PC-update operation in a later clock cycle. The idle flag is set by the IDLE instruction and cleared when an interrupt occurs.

After saving all results produced by instructions, the outputs from accelerators are also saved (regardless of the stall flag). Execution is now stopped if a break point or the end of the program is reached. Otherwise execution continues. If either of the stall or idle flags are set, no new instruction is fetched during the next clock cycle.

8.3 Firmware

11a and 11b transmitters and receivers have been scheduled onto the described architecture and firmware has been implemented. The performance of the processor was evaluated in terms of the minimum clock frequency necessary to meet all latency and throughput requirements at the highest data rate in each standard (54 Mbit/s in 11a and 11 Mbit/s in 11b). Table 8.1, shows the required frequencies and the program and data

Table 8.1: Firmware implementation results

Task	Required freq.	Program size	Data memory
11a Tx	155 MHz	1020 bytes	3456 bytes
11a Rx	160 MHz	1658 bytes	2340 bytes
11b Tx	120 MHz	476 bytes	484 bytes
11b Rx	110 MHz	1090 bytes	304 bytes

memory usage for each module.

The frequency limit for the 11b receiver is imposed by the throughput requirements at the highest data rate while the 11a receiver has throughput requirements and latency requirements for minimum size packets that are approximately equally restricting. For both transmitters throughput requirements are the limiting factors.

The required program and data memory sizes are small, proving the efficency of the instruction set and the memory architecture. A combined 11a and 11b firmware would require approximately half of the available program and data memory on the demonstrator chip. The requirements are lower than the sum of the figures in the table since a lot of the data memory and some subroutines are reused between modules.

A detailed description of the scheduling and hardware mapping is provided in appendix B. Since this is the first firmware version it is believed that further optimizations of the code should be possible. It is also possible to make other trade-offs between cycle cost and program size.

8.4 Prototype Chip

The design was implemented in VHDL using the HDL Designer tool from Mentor Graphics. Mentor Graphics' ModelSim was used for RTL simulation, verification, and post synthesis net list simulation. Output from the instruction set simulator was used as golden results.

Synthesis and backend flow was carried out using Synopsis backend tools. An Artisan standard cell library and memory generator was used.

The chip was manufactured in a 0.18 μm CMOS process from Chartered Semiconductor Manufacturing (CSM).

Power consumption is quoted at 160 MHz since that is the clock frequency that would be used for IEEE 802.11a or IEEE 802.11g processing. The quoted idle power is measured after execution of the IDLE instruction, meaning that instruction fetching is stopped but all clocks are still running at full speed.

Figure 8.3 shows a photo of the manufactured chip. All white boxes are memories. PM is the program memory and the remaining boxes are data memories. Table 8.2 summarizes the chip features.

The chip does not include the Viterbi decoder. It was skipped in order to meet the tapeout deadline. Hardware implementation of the Viterbi algorithm is a well explored area [3]. The silicon area of a Viterbi decoder for IEEE 802.11a is estimated to approximately 1 mm^2.

Table 8.3 shows the approximate area used by different parts of the processor. The estimations for individual logic blocks may be inexact since it is based only on the fraction of the logic cell area (and not the routing area) occupied by each block. The network in particular is expected to use a larger fraction of the logic area because of the large amount of interconnects in that block.

Table 8.2: Chip feature summary

Feature	**Value**
Technology	0.18 μm CMOS
Chip area	5 mm^2
Memory area	1.0 mm^2
Logic area	1.9 mm^2
Max frequency	240 MHz
Package	144 pin fpBGA
Power consumption @160 MHz:	
Idle	44 mW
Peak for 11a Reception	126 mW

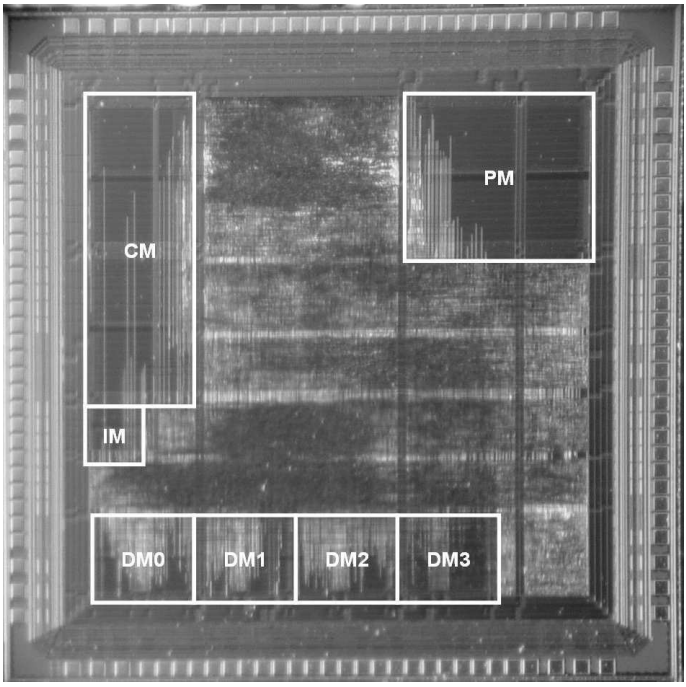

Figure 8.3: Die photo

Table 8.3: Chip area usage

Component	Area	Fraction of total area
Control path	0.18 mm^2	6.1%
MAC unit	0.46 mm^2	15.8%
Core data path except MAC	0.20 mm^2	7.0%
Memory addressing and control	0.22 mm^2	7.7%
Network	0.17 mm^2	5.9%
Accelerators	0.66 mm^2	22.9%
Logic total	*1.89 mm^2*	*65.4%*
Program memory	0.30 mm^2	10.4%
Data memories	0.70 mm^2	24.2%
Memory total	*1.00 mm^2*	*34.6%*

8.5 Test Board and Measurement Setup

A printed circuit board for test purposes was built. In addition to the BBP1 chip the board has an Atmel AVR microcontroller that is used for booting and controlling the BBP1 via a serial cable from a PC. The BBP1 can be clocked by an oscillator on the board, by an external clock, or by a slow clock generated by the microcontroller. For measurements, the board is connected to a Tektronix TLA721 pattern generator and logic analyzer system.

The chip was tested with the same test suits that were used for RTL verification and with the implemented 11a and 11b transmitter and receiver firmware. Receivers were tested with ADC data generated by behavioral models and with recorded air data. The chip functions correctly up to a clock frequency of approximately 240 MHz.

For power measurements, a special program was designed that continuously computes 64-point FFTs and simultaneously operates most accelerators as well as ADC and MAC interfaces. This is believed to correspond to the peak power conditions during 11a processing.

Bibliography

[1] Free Software Foundation, *GNU Flex 2.5 – Reference manual*. http://www.gnu.org/software/flex/manual/, 2005.

[2] Free Software Foundation, *GNU Bison 2.0 – Reference manual*. http://www.gnu.org/software/bison/manual/, 2005.

[3] S. Bitterlich and H. Meyr, "Efficient scalable architectures for Viterbi decoders," in *Proceedings of the International Conference on Application-Specific Array Processors*, pp. 89–100, Oct. 1993.

Part IV

Conclusions and Future Work

Chapter 9
Conclusions

This chapter summarizes some of the most important conclusions and results from the presented work. This includes general conclusions regarding programmable baseband processing, a unique programmable baseband processor architecture, and demonstration of the architecture in a manufactured demonstrator chip running application firmware at full speed.

9.1 Issues in Design of Programmable Baseband Processors

Design of programmable baseband processors has been discussed. Starting from general knowledge in design of application specific processors, the specific properties and requirements of baseband processing applications have been studied. The focus has been on low-cost and low-power architectures useful in handheld wireless terminals.

Some general conclusions regarding design of programmable baseband processors have been drawn:

Complex-valued computations: A large part of the computations in baseband processing is convolution-based operations on complex data. Convolution-based operations can in general be implemented efficiently in programmable processors (i.e. general DSP processors).

By introducing instructions and data paths for complex calculations, complex convolution-based functions can also be implemented efficiently.

Function level acceleration: Making the right choice of accelerators is important to reach a power- and area-efficient solution. Bit manipulation operations such as scrambling, CRC, and channel coding are usually inefficient in software but have very small hardware implementations. These operations are therefore often good candidates for acceleration although the MIPS may not be very high. Viterbi and Turbo decoding should be accelerated in most cases. These operations occur with small variations in many standards, making configurable accelerators a good choice. It is also a good idea to accelerate some operations close to the ADC/DAC interface, such as symbol shaping, decimation, and frequency offset compensation. Such functions are used in most standards, are running a large fraction of the time, and are rather computing intensive at high sample rates.

Since the data blocks in baseband processing are usually small, it is important to integrate the accelerators in the architecture in a way that provides low communication and control overhead.

Memory issues: The amount of memory required in a baseband processor is generally low but the required memory bandwidth is often relatively high. High memory bandwidth can be reached either by using wider memories, several memory banks, or dual port memories. The memory access patterns in baseband processing are known in advance and often very regular. Therefore dual port memories, being the most flexible but also the most costly solution, usually can and should be avoided.

9.2 An Architecture for Efficient Baseband Processing

A programmable baseband architecture has been presented. The architecture enables a good tradeoff between flexibility and performance and has proven to result in area- and power-efficient implementations of multi-standard programmable baseband processors.

Important novel features are:

- Vector instructions in which the vector size is given explicitly in the instruction. This saves program memory and allows instruction level parallelism without significant increase of control path complexity.
- An instruction set optimized for baseband processing, including vector instructions for operations on complex data.
- A scheme for interconnection of accelerators, memories, and peripherals to the programmable core via a configurable network. This scheme allows a high degree of parallelism and low overhead for communication and accelerator configuration. This is achieved by using a simple network which is statically scheduled and configured by the programmable core. The network is closely coupled to the instruction set architecture e.g. in terms of dedicated instructions for network and accelerator configuration.

9.3 Implementation Results

A demonstrator chip was manufactured to prove the architecture. The implemented processor was optimized for the WLAN standards IEEE 802.11a, b, and g. It includes accelerators for front-end operations, demapping, scrambling, CRC, interleaving, channel coding, and modified Walsh transform.

11a and 11b baseband PHY was mapped and scheduled on the architecture. Firmware was developed and tested on the manufactured chip.

The required silicon area is similar to what can be reached in a fixed function 11a/b baseband solution.

Measurements proved that the chip can operate at significantly higher speeds than those needed for the mentioned standards. Power consumption was also found to be competitive to fixed function solutions.

The ultimate conclusion is that the central ideas of the presented architecture are useful for implementation of low-cost, low-power programmable baseband processors, suitable for handheld multi-standard wireless terminals.

Chapter 10
Future Work

This chapter summarizes different directions the future work in this project may take. Much of this has already been touched upon in previous chapters. Some of the points are just small implementation issues or instruction set modifications and some could be entire PhD projects in themselves. Some things are already being taken care off and some probably never will.

10.1 ISA Improvements

The programmer and compiler friendliness of an instruction set may be an important issue which has not been deeply considered so far in this project. Based on the experience from firmware development and design of a C-compiler for BBP1, some areas for improvements have be identified and are currently being implemented. Some concrete examples of improvements are better support for software stacks and context switching.

10.2 Architecture Scaling

Ways of improving performance, such as using more and wider vector execution units, was discussed in section 7.7. Much of this is currently

being investigated for the next generation of the architecture.

Related issues which have not been studied so far are multiple core solutions and scheduling issues, especially for running multiple standards simultaneously.

10.3 Acceleration

The Viterbi accelerator that is needed by IEEE 802.11a/g and other standards is not yet available. A Viterbi or combined Viterbi/turbo decoder should be implemented. For example 3G standards and WiMax also use Reed-Solomon codes. Acceleration of Reed-Solomon decoding has so far not been studied at all in this project.

Another idea that has been considered is to design a more general bit manipulation processor that could handle many of the less complex bit manipulation tasks, such as channel coding, scrambling, CRC, and possibly interleaving.

Support for MIMO is certainly of great interest for the future. So far an initial study has been carried out. Some results from this can be found in [1].

10.4 Low-Power Features

Several generally available low power features may be considered for future implementations. Thanks to the modular architecture, clock gating for unused accelerators should be relatively straight forward to implement. Data-width masking may also be considered together with a more methodical way of determining data precision requirements.

10.5 Hardware

Having a hardware platform including ADC/DAC, radio, and MAC as well as the baseband processor would be invaluable for demonstration

purposes. Access to a radio front-end and ADC/DAC will also be necessary in order to gain a better understanding of what the real problems are and to look further into issues such as automatic gain and frequency control and the effects of radio impairments. These issues have not been sufficiently investigated so far in this project.

A hardware platform including radio, ADC/DAC, a high performance FPGA capable of running the baseband processor, and an ARM7 TDMI processor for control and MAC processing is currently being designed by Lic. Eng. Anders Nilsson.

10.6 Firmware Design Tools

More advanced design tools may be needed with a more complex architecture. If multiple standards should run simultaneously, tools to help scheduling are probably needed. If multi-processor solutions are used, a multi-processor simulation platform will be needed.

Bibliography

[1] Haiyan Jiao, Anders Nilsson, Eric Tell, and Dake Liu, "MIPS cost estimation for OFDM-VBLAST systems," in *paper submitted for review*, 2005.

Part V

Appendix

Appendix A
Application Profiling and Benchmarking

This appendix contains an overview of profiling and benchmarking figures that motivate the presented architecture and the choice of instruction set.

As a starting point, table A.1 gives an overview of the types of operations that are used in the WLAN applications for which the processor was designed (IEEE 802.11a [1] and IEEE 802.11b [2]). The focus is on the receiver flow since that is more demanding than the transmitter processing.

In the table, CMUL stands for operations based on complex multiplications and CMAC for operations based on complex multiply-accumulate operations (such as correlation, filtering, or scalar products).

Table A.2 shows the estimated MIPS capacity needed for the different functions in a general DSP processor in order to meet the timing requirements from the standards, assuming that a complex MAC unit exists. Most numbers were previously published in [3].

In order to reach a moderate clock frequency (no more than 200 MHz) without excessive hardware it is necessary to accelerate the receive/decimation filter and the Viterbi decoder. It is also suitable to accelerate demapping, interleaving, scrambling, and CRC, considering the negligible hardware size for these operations. Software implementation of CRC

Table A.1: Summary of IEEE 802.11a/b receiver processing

Function	**IEEE 802.11a**	**IEEE 802.11b**
Receive filtering	FIR filter **CMAC**	FIR filter **CMAC**
Packet detection	Auto correlation on short pilots **CMAC**	Auto correlation on preamble **CMAC**
Fine synchronization	Frequency-domain cross correlation on long pilot **FFT, CMUL, IFFT, absolute maximum search**	Cross correlation with barker sequence **CMAC, absolute maximum search**
Frequency offset estimation	Calculate argument of pilots autocorrelation. Tracking based on pilot tones **CMAC and cordic algorithm**	Not needed thanks to differential modulation
Frequency offset correction	Rotor **CMUL**	Not needed thanks to differential modulation
Channel estimation	Frequency-domain channel estimation based on long pilots **vector addition, FFT, CMUL**	Cross correlation with barker sequence, uses result from synchronization step
Channel equalization	Each sub-carrier multiplied by individual compensation factor **CMUL**	Channel-matched filter **CMAC**
Domain translation	**FFT**	barker despread **CMAC**
Demapping	BPSK, QPSK, 16-QAM, 64-QAM	DBPSK/DQPSK: **CMUL** + BPSK/QPSK CCK: **Modified Walsh transform** + **absolute maximum search** + DQPSK
Deinterleaving	**Block interleaver**	-
Error correction	**Viterbi algorithm**	-
Descrambling	8-bit LFSR	8-bit LFSR
Error checking	**CRC-32** for MAC frame	**CRC-16** for PLCP Header **CRC-32** for MAC frame

Table A.2: IEEE 802.11a/b MIPS costs at highest data rates

Function	**11a**	**11b**
Receive filter	600	660
Packet detection	80	44
Synchronization and channel estimation	108	12
Frequency offset compensation	80	-
Channel compensation	16	60
FFT/Walsh	108	160
Demapping	108	22
Deinterleaving	270	-
Viterbi	4000	-
Descrambling	162	33
CRC	37	5.5

also consumes significant data memory for look-up table storage, especially since 11b would need both a CRC-16 and a CRC-32 table.

Focusing on the remaining operations, the first observation that can be made is that they are almost entirely based on complex-valued data. It is therefore beneficial to go for an architecture that handles complex valued data natively.

Secondly it can be observed that for 11a (as for any OFDM standard), FFT calculations constitutes a very large part of the total computing load. Hence computation of FFT should be accelerated either on function or instruction level.

During continuous processing of data symbols, one 64-point FFT must be computed for every symbol, i.e. every 4 μs. Assuming that a radix-2 FFT algorithm is used, every 64-point FFT consists of 192 butterflies. This gives an average of $192/4 \cdot 10^{-6} = 48 \cdot 10^{6}$ butterflies per second. The available time for synchronization and channel estimation gives a similar value for the FFT computing requirements for that part of the flow. With somewhere between $1/4$ and $1/3$ of the computation time budgeted for

the FFTs, this means that an FFT instruction allowing one radix-2 butterfly to be calculated every clock cycle should be enough to reach a clock frequency below 200 MHz.

Next to FFT calculations, the biggest challenge in 11a is the heavy processing in the beginning of each frame which must be finished early enough to reach the latency requirements given by the short intra frame pacing (SIFS, see appendix B.1) for minimum size packets. This processing mostly consists of heavy multiplication/MAC based computation for synchronization and channel estimation. A data path including two complex multipliers was therefore deemed necessary to meet the latency requirements.

The SIFS requirement is less challenging for 11b. The largest problem is instead to reach enough throughput at the highest data rate (11 Mbit/s). The most cycle consuming function in 11b is the modified Walsh transform (see table A.2).

The first instruction set architecture proposal included dual complex MAC vector instructions, as described in section 7.3, and instruction level acceleration for FFT and Walsh transform.

During early benchmarking of the 11a receiver on the instruction set simulator it was found that the SIFS requirements would not be met at the target frequency of 160 MHz (four times the sampling frequency). One reason is that the absolute maximum search in the synchronization step requires a lot of clock cycles. Due to its inherit conditional operations, the max search does not benefit significantly from the dual MAC. The MAC unit was therefore extended to also allow instruction level acceleration for finding absolute maximum value and position in a complex vector.

During 11b benchmarking it was found that the throughput requirements were not reached for the highest data rate at the target frequency of 154 MHz (seven times the sampling frequency). For this reason it was decided to apply function level acceleration for the modified Walsh transform. The extra hardware requirement is not excessive since the modified Walsh transform only has multiplications by +/-1 and +/-i.

Table A.3 illustrates the impact of the dual complex MAC and instruc-

Table A.3: Cycle cost for IEEE 802.11a kernel operations

Benchmark	A	B	C
size 64 complex vector add	138	38	38
size 64 complex scalar product	270	36	36
64-point FFT[a]	1900	650	228
64 sample abs max search	460	362	34
11a fine synchronization	4700	1990	540
11a FFT+channel compensation	2130	695	228

A: General single MAC DSP

B: DSP with dual complex MAC unit

C: DSP with dual complex MAC and instruction level acceleration for FFT and absolute max search

[a] addressing support for FFT is assumed to exist

tion level acceleration on the cycle cost for some 11a kernel operations. Column A shows cycle costs for a general single MAC unit DSP processor. The estimations are based on data from *The Buyer's Guide to DSP Processors* [4]. Column B are estimated cycle costs for the proposed architecture but without the hardware acceleration for FFT and absolute maximum search. Column C are values from simulations of the presented BBP1 baseband processor. Apart from kernel operations, the table also contains cycle cost for the fine time synchronization, which is composed of a vector addition, an FFT, a vector multiplication, an IFFT, and an absolute maximum search. FFT and channel equalization (e.g. a vector multiplication) which are the heavy parts of the data symbol processing is also included. The FFT acceleration instructions allow the vector multiplication to be integrated into the FFT at no extra cycle cost in both these cases.

Bibliography

[1] "IEEE 802.11a - 1999, wireless LAN medium access control (MAC) and physical layer (PHY) specifications: High-speed physical layer in

the 5 GHz band," 1999.

[2] "IEEE 802.11b - 1999, wireless LAN medium access control (MAC) and physical layer (PHY) specifications: Higher-speed physical layer in the 2.5 GHz band," 1999.

[3] Anders Nilsson, Eric Tell, and Dake Liu, "An accelerator architecture for programmable multi-standard baseband processors," in *Proceedings of the International Conference on Wireless Networks and Emerging Technologies (WNET)*, pp. 644–649, July 2004.

[4] Berkley Design Technology, Inc., *Buyer's guide to DSP processors, 2001 ed.* BDTi, 2001.

Appendix B
Scheduling and Hardware Allocation

This chapter contains diagrams showing the timing and and hardware allocation of the receiver flows for IEEE 802.11a [1] and IEEE 802.11b [2]. Since transmission is less challenging it is only discussed briefly and no diagrams are included.

B.1 The SIFS

The one timing requirement from the IEEE 802.11 standards that has the largest impact on the design of the baseband processor is the so called short intra-frame spacing (SIFS). Among other things, the SIFS specifies the time from reception of the last bit of a packet at the antenna of the receiver until the transmission of a response should start. This imposes strict latency requirements on the receiver.

The SIFS in 11a is 16 μs and in 11b it is 10 μs. Assuming 2 μs for switching the front-end from receive to transmit mode, 2 μs delay through the front-end interface (mainly the filter), and a total of 2 μs for delays through the radio front-end and the MAC layer, 10 μs remains for the delay through the baseband receiver in the 11a case and 4 μs in the 11b case. The delay through the baseband transmitter is essentially zero since

the preamble is precomputed and stored in memory.

B.2 IEEE 802.11a

IEEE 802.11a is an OFDM standard using 64 sub-carriers (48 data carriers, 4 pilot tones and the remaining are unused guard carriers). The sample rate is 20 MHz. A cyclic prefix of 16 samples is used which gives a total symbol time of 4 μs.

The 11a frame starts with ten identical short pilot symbols of 16 samples each. These are used for packet detection and frequency offset estimation. Once the frequency error has been estimated, the rotor in the front-end accelerator is set to compensate it.

Next comes two identical long pilot symbols, each 64 samples, preceeded by a 32 sample cyclic prefix. The long pilots are used for synchronization and channel estimation.

The long pilots are followed by a header symbol containing packet information, like data rate and packet size. These fields must be decoded, descrambled, and extracted before the complete first payload symbol has arrived. Otherwise additional buffer memory is needed and the control flow will be more complex resulting in difficulties in reaching the latency requirements

Figure B.1 shows the scheduling for 11a reception. The only difference between different data rates is the delay through the accelerator chain. The schedule shows a receiver delay of less than one symbol time (4 μs), clearly meeting the SIFS requirements.

The pilot symbols for transmission are stored in memory and transmitted directly. Transmit processing consists of scrambling, convolutional encoding, interleaving, and mapping which are all handled by a chain of accelerators. The core handles insertion of pilot carriers from values stored in a table in coefficient memory and IFFT calculation.

B.3 IEEE 802.11b

For the lower data rates IEEE 802.11b employs direct sequence spread spectrum (DSSS) using a Barker sequence of length 11 for spreading. At the higher data rates it uses complementary code keying (CCK) which encodes groups of four or eight bits into eight complex chips. CCK is based on a modified Walsh transform. A description of CCK can be found in [3]. The chip rate is 11 Mchip/s giving a symbol time of 1 μs for DSSS and 8/11=0.727 μs for CCK.

The 11b frame starts with a SYNC field consisting of 128 (or optionally 56) scrambled ones at the lowest data rate, used for symbol synchronization and channel estimation. Next is the 16 bit start frame delimiter (SFD) which is used to find the beginning of the header. The 48 bit header contains information on data rate and packet size and is protected by a CRC-16 check sum. Figure B.2 shows the schedule for the header processing.

Figure B.3 shows the payload processing in DSSS mode. The processor load is very low for this mode. The limiting part is CCK demodulation at 11 Mbit/s which is demanding due to the short symbol time and the complexity of the Walsh transform and the following absolute maximum search. The Walsh accelerator which handles both the actual modified Walsh transform and the absolute maximum search requires 70 clock cycles to complete. Figure B.4 shows the 11 Mbit/s schedule. The SIFS requirements are easily met as long as the CCK throughput is enough.

The transmitter is also limited by 11 Mbit/s CCK modulation. It requires some bit manipulation, a table look-up, a DQPSK mapping, and eight complex multiplications.

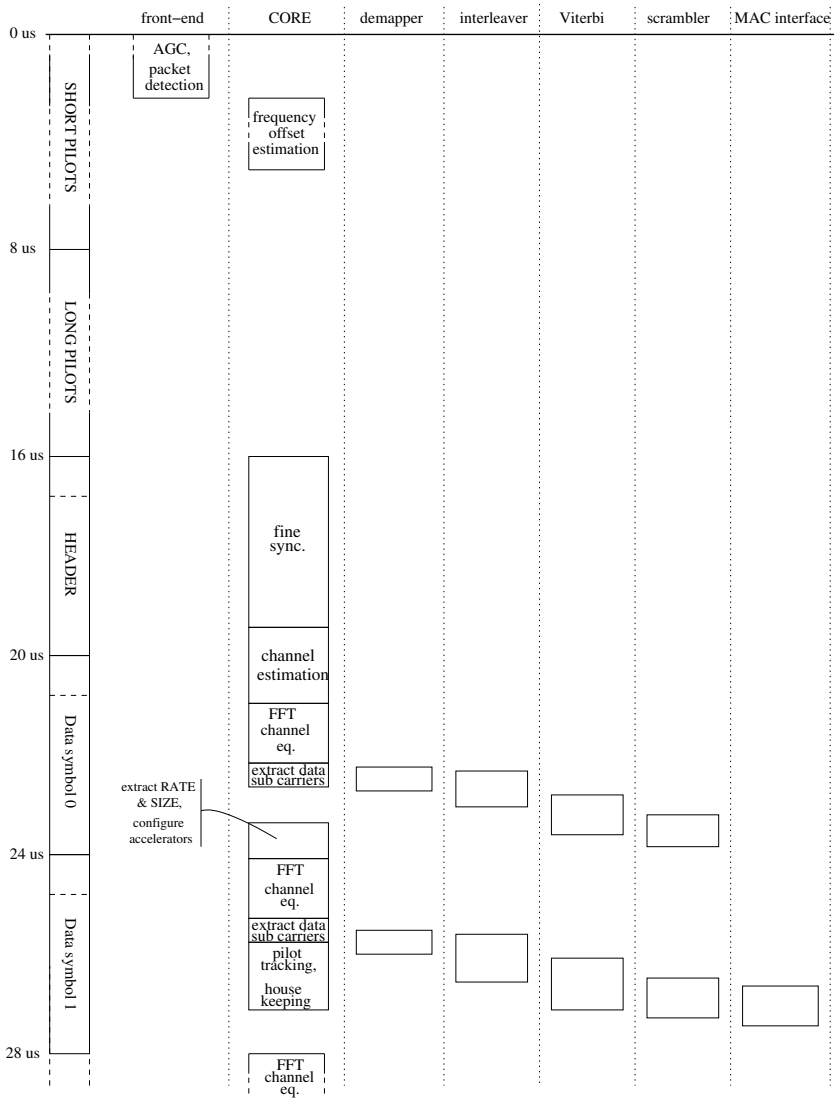

Figure B.1: Scheduling of IEEE 802.11a receiver for 160 MHz clock frequency

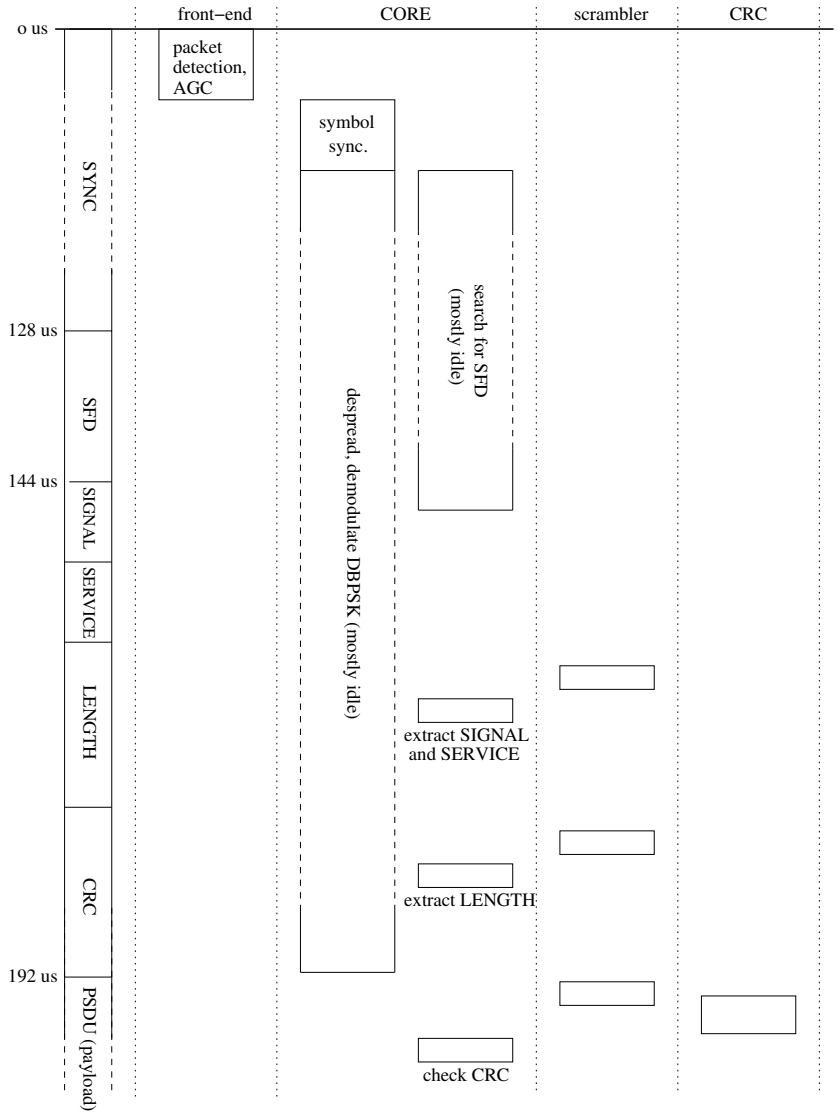

Figure B.2: Scheduling of IEEE 802.11b preamble and header processing for 154 MHz clock frequency

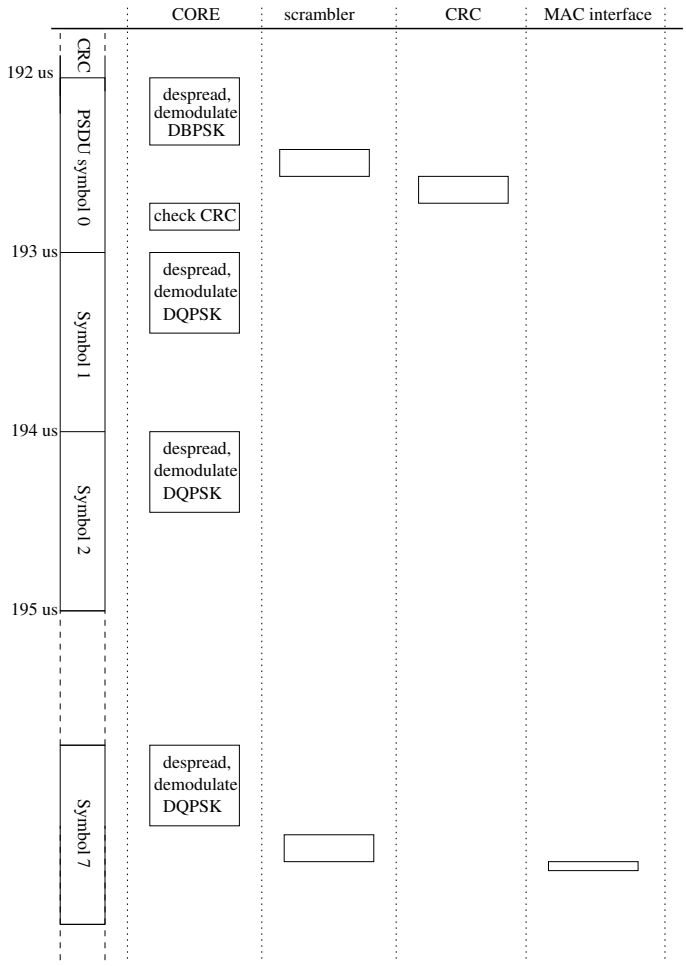

Figure B.3: Scheduling of IEEE 802.11b 2 Mbit/s reception

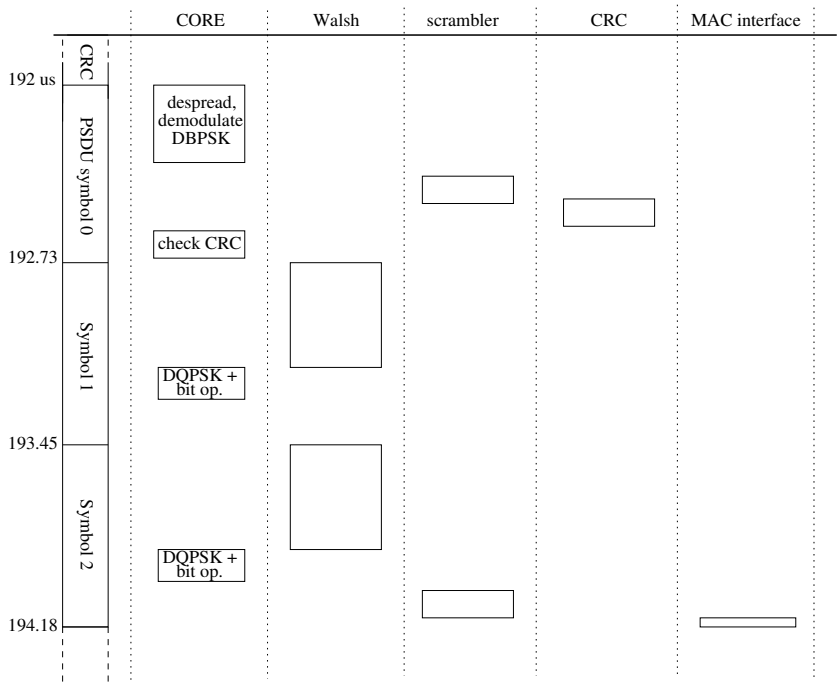

Figure B.4: Scheduling of IEEE 802.11b 11 Mbit/s reception

Bibliography

[1] "IEEE 802.11a - 1999, wireless LAN medium access control (MAC) and physical layer (PHY) specifications: High-speed physical layer in the 5 GHz band," 1999.

[2] "IEEE 802.11b - 1999, wireless LAN medium access control (MAC) and physical layer (PHY) specifications: Higher-speed physical layer in the 2.5 GHz band," 1999.

[3] B. Pearson, *Complementary Code Keying Made Simple*. Intersil corporation, white paper, 2001.

Index