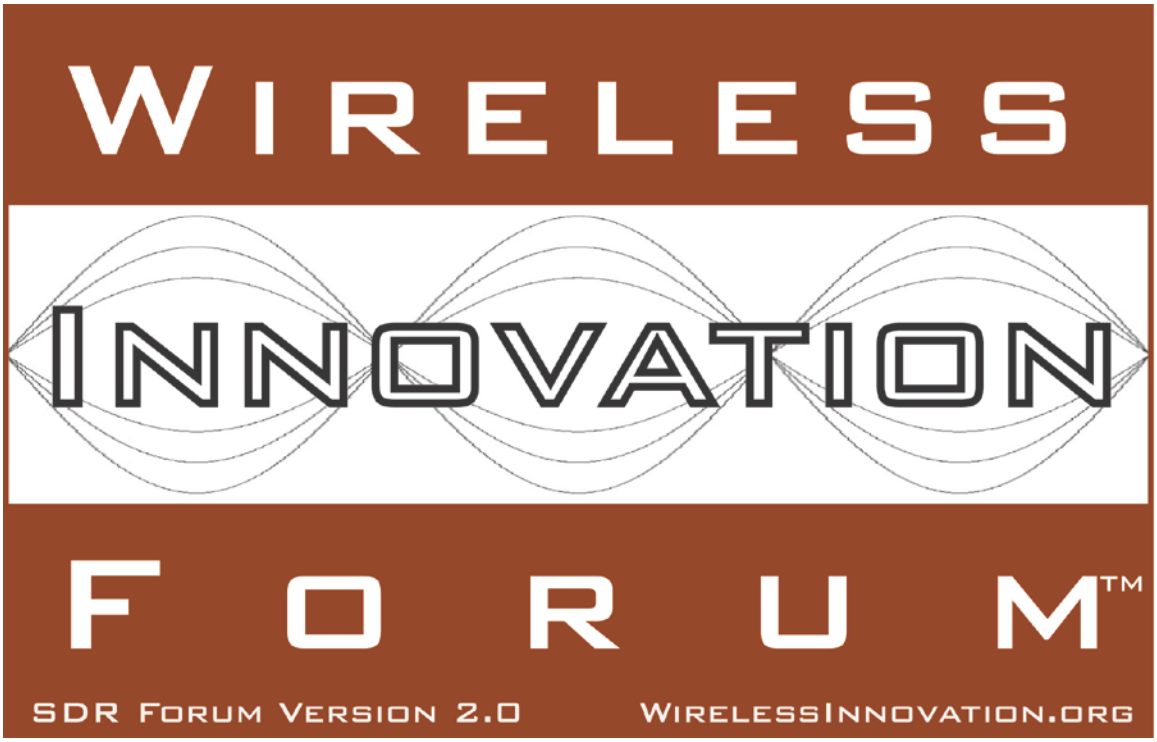

Commercial Baseband Technology Overview:

The Current State of Technology Development and Future Directions

Document WINNF-09-P-0009-V1.0.0

Version 1.0.0

5 February 2010

TERMS, CONDITIONS & NOTICES

This document has been prepared by the Commercial Baseband Processing Technologies Work Group to assist The Software Defined Radio Forum Inc. (or its successors or assigns, hereafter "the Forum"). It may be amended or withdrawn at a later time and it is not binding on any member of the Forum or of the Commercial Baseband Processing Technologies Work Group.

Contributors to this document that have submitted copyrighted materials (the Submission) to the Forum for use in this document retain copyright ownership of their original work, while at the same time granting the Forum a non-exclusive, irrevocable, worldwide, perpetual, royalty-free license under the Submitter's copyrights in the Submission to reproduce, distribute, publish, display, perform, and create derivative works of the Submission based on that original work for the purpose of developing this document under the Forum's own copyright.

Permission is granted to the Forum's participants to copy any portion of this document for legitimate purposes of the Forum. Copying for monetary gain or for other non-Forum related purposes is prohibited.

THIS DOCUMENT IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NON-INFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY USE OF THIS SPECIFICATION SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE FORUM, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER, DIRECTLY OR INDIRECTLY, ARISING FROM THE USE OF THIS DOCUMENT.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the specification set forth in this document, and to provide supporting documentation.

Wireless Innovation Forum ™ and SDR Forum ™ are trademarks of the Software Defined Radio Forum Inc.

Table of Contents

List of Figures

List of Tables

CONTRIBUTORS

Etherstack

Anna Squires
Etherstack Inc.
145 W 27th St
New York, NY 10001 USA
917-661-4110
Anna.squires@etherstack.com

IMEC

Bart Van Poucke
IMEC
Kapeldreef 75, Leuven, Belgium
+32 16 288 351
bart.vanpoucke@imec.be

Sandbridge

John Glossner
Sandbridge Technologies, Inc.
59 Composite Way, Lowell, MA 01851 USA
+1 914 715 9622
jglossner@sandbridgetech.com

ST-Ericsson

Kees Moerman
ST-Ericsson
High Tech Campus HTC41, 5656 AE Eindhoven, The Netherlands
+31 40 292 2535
E-mail: kees.moerman@stericsson.com

Kees van Berkel
ST-Ericsson
High Tech Campus HTC41, 5656 AE Eindhoven, The Netherlands
+31 40 292 2553
E-mail: kees.van.berkel@stericsson.com

Xilinx

Manuel Uhm
Xilinx
2100 Logic Drive
San Jose, CA 95124 USA
+1 408-626-6325
manuel.uhm@xilinx.com

COMMERCIAL BASEBAND TECHNOLOGY OVERVIEW: CURRENT STATE OF TECHNOLOGY DEVELOPMENT AND FUTURE DIRECTIONS

1 Executive Summary

This document discusses both programming environments and hardware platforms for Software Defined Radio (SDR). It is not intended to provide a generic description of programmable baseband technologies. Rather it focuses on a few specific technologies as representative examples.

SDR technology has developed rapidly in recent years. Semiconductor process improvements following Moore's Law have made low power, low cost devices available that are suitable for commercial deployments. With such advancement, designs have been introduced for a range of both infrastructure and handset user equipment that use SDR techniques. As technologies continue to converge, SDR designs offer elegant and reusable methods of implementing worldwide communications coverage.

In the opening section Etherstack describes a programming methodology used on commercial projects that maximizes software portability, maintainability and performance, and minimizes cost.

IMEC describes an SDR platform implemented in TSMC 90G multi-VT technology which targets 3 antennas, 200Mbps platforms with under 500mW power consumption. This is accomplished on a VLIW + Array Processor with SIMD capability platform. A retargetable C compiler transparently maps data-flow dominated loops to the array and the remaining code to the VLIW processors. Intrinsics are also provided for complex multiplication and vector operations.

Sandbridge next describes the SB3500 MPSoC implemented in 65nmprocess technology. The 4-way multithreaded 16-wide vector processors each run at 600MHz. There are 3 processors per chip. The MPSoC is fully programmed in ANSI C with a compiler capable of vectorizing data parallelism, determining non-associative DSP datatypes and generating threads across multiple processors. Complete systems for handset designs including UMTS, 1xEVDO, WiMax, WiFi, DVB, GPS, and recently LTE have been implemented on the platform.

St-Ericsson next describes the EVP, earlier developed by NXP: a 16-wide vector processor with support for handset (WLAN, HSPA, TD-SCDMA, DVB, LTE, etc.). A 45nm LP CMOS implementation of the EVP runs at 320MHz (worst case) with 0.5mW/MHz power dissipation (including data and program memory). They describe an EVP-C compiler which accepts ANSI-C plus function intrinsics to describe vector operations. The EVP is already in mass production as part of a TD-SCDMA baseband chip.

Xilinx then provides a description of their infrastructure FPGA platform. A comprehensive range of devices are capable of implementing a wide range of interface standards such as LTE, TD-SCDMA, WiMax, UMTS, and CDMA.

As future revisions of this document are released, companies with examples beyond the contributors to this document are encouraged to participate.

2 Introduction

Traditional communications systems have typically been implemented using custom hardware solutions. Chip rate, symbol rate, and bit rate co-processors are often coordinated by programmable DSPs but the DSP processor does not typically participate in computationally intensive tasks. Even with a single communication system the hardware development cycle is difficult often requiring multiple chip redesigns late into the certification process. When multiple communication systems' requirements are considered, both silicon area and design validation are major inhibitors to commercial success. A software-based platform capable of dynamically reconfiguring communications systems enables elegant reuse of silicon area and dramatically reduces time to market through software modifications instead of time consuming hardware redesigns.

The advantages of reconfigurable SDR solutions versus hardware solutions are significant.

1. Reconfigurable solutions are more flexible allowing multiple communication protocols to dynamically execute on the same transistors thereby reducing hardware costs. Specific functions such as filters, modulation schemes, encoders/decoders etc., can be reconfigured adaptively at run time.

2. Several communication protocols can be efficiently stored in memory and coexist or execute concurrently. This significantly reduces the cost of the system for both the end user and the service provider.

3. Remotely reconfigurable protocols provide simple and inexpensive software version control and feature upgrades. This allows service providers to differentiate products after the product is deployed.

4. The development time of new and existing communications protocols is significantly reduced providing an accelerated time to market. Development cycles are not limited by long and laborious hardware design cycles. With SDR, new protocols are quickly added as soon as the software is available for deployment.

5. SDR provides an attractive method of dealing with new standards releases while assuring backward compatibility with existing standards.

3 Description of Contents

As hardware platforms have bifurcated into processor-based SDRs and FPGA-based SDRs, software techniques to program both homogeneous and heterogeneous systems have emerged. Rather than provide a generic description of programmable baseband technologies, this document is intended to give a few representative examples of platforms available and programming techniques. In the first section, Etherstack presents their programming methodology for SDR development. Next, three processor-based SDRs and their programming environments are presented. In section two, IMEC discusses the ADRES processor. In section three, Sandbridge discusses the SB3500 MPSoC. In section four, St-Ericsson (formerly NXP) discusses the EVP processor. In the final section Xilinx discusses FPGA-based SDRs.

CHAPTER 1 – TECHNIQUES FOR COMMERCIAL SDR WAVEFORM DEVELOPMENT (ETHERSTACK)

Abstract

Software Defined Radio (SDR) hardware platforms have been used in commercial defence and Land Mobile Radio (LMR) communications for several years. However the real value of SDR - software reusability, upgradeability and portability – is still often not achieved because the software itself fails to exploit the full potential of these platforms.

This paper introduces SDR software development practices successfully used in commercial radio projects to maximise software Portability, Maintainability and Performance (PMP), and to minimise software cost. These involve the application of specialist expertise, tools and procedures at each of the specification, design, implementation, integration and maintenance phases.

The practices described have been developed to address real problems found in industry, such as managing the deployment of a waveform across multiple disparate hardware platforms that may include heterogeneous devices or use the Software Communications Architecture (SCA).

1 Introduction

Wireless voice communications can be broadly divided into three industries: cellular, defence and Land Mobile Radio (LMR). The emergence of large-scale commercial cellular communications set a precedent in all three industries for bigger networks with more users and features, which coincided with a need for improved spectrum efficiency and utilization. These factors demand more intelligent radios, and intelligent radios require more software. Accordingly, all three industries began to shift from traditional analogue to digital technology.

In the commercial cellular industry, this shift from analogue (AMPS/ETACS/NMT450) to digital happened swiftly, helped by high sales volumes that meant the newly essential software intelligence was available to manufacturers already incorporated into dedicated function integrated chipsets. Unfortunately such a solution was unavailable in the lower-volume defence and LMR industries, which instead built software capable platforms out of affordable technology developed for the first generation of digital cellular devices: low-cost, low-power and high-peripheral count generic solid-state processors such as GPPs, DSPs and FPGAs.

As this shows, the commercial use of SDR is not new. Rather SDR platforms – that is, those that use generic processing nodes such as GPPs, DSPs and FPGAs to execute radio function in software – have been commercially available since the first digital Land Mobile Radio (LMR) mobile and base stations went to market approximately eight years ago.

Out of the first SDR hardware platforms an entire field has grown. However research in SDR software – or SDR waveform – development has lagged behind progress in SDR hardware and platform technology. Most waveform developments still involve cobbling together software design techniques

and tools from other fields and various different vendors, rather than offering an integrated approach geared for SDR.

As more radio function is implemented within the waveform and the value of the waveform therefore grows, it is worth considering how the wireless communications industries might improve the execution of waveform development in order to protect this value. The dedicated SDR tools and techniques introduced in this paper have been developed by a commercial SDR waveform company in order to optimise waveform portability, maintainability and performance and thus ensure best return on investment in waveform development.

2 What is a Commercial SDR Waveform?

According the Wireless Innovation Forum's* approved definitions [1] a waveform is:

1. The set of transformations applied to information to be transmitted and the corresponding set of transformations to convert received signals back to their information content.
2. Representation of a signal in space
3. The representation of transmitted RF signal plus optional additional radio functions up to and including all network layers.

In practice, the term 'waveform' is applied to a range of different software implementations, from a single signal processing algorithm or group of algorithms to a commercial SDR waveform product. However in terms of structure and complexity, these are entirely different.

Firstly, the signal-in-space component itself can vary widely in intricacy. A production-level commercial SDR waveform deployed on a radio used within a real communications network is substantially more complicated than, for example, the operation of just the Physical Layer of that waveform in demonstration in a controlled environment. It requires a great deal of additional complexity at the Network Layer, and must also co-ordinate the behaviour of many algorithms and higher control functions across all layers of the waveform.

Secondly, and more significantly, in addition to meeting complex signal-in-space and co-ordination requirements, a commercial SDR waveform must **manage the relationship between hardware and software** in order to optimise.

* In 2009 the Software Defined Radio Forum Inc. underwent a major strategy review, led by representatives from multiple member organizations. The result was a revised strategic plan that included rebranding the organization as the Wireless Innovation Forum (SDR Forum Version 2.0). This change in name reflects the fact that member organizations within the Forum are driving innovation in areas beyond "Software Defined Radio" to include system of systems, ad-hoc networks, cognitive radio, dynamic spectrum access, etc. The strategy update, including the name change, was put before the members by the Board of Directors at the Annual Meeting of the members on 3 December 2009 where it was approved. The legal name of the organization remains "The Software Defined Radio Forum Incorporated".

The press release announcing the name change can be found here:
http://www.businesswire.com/portal/site/home/permalink/?ndmViewId=news_view&newsId=20091203006419&newsLang=en

2.1 Portability

The waveform should be developed in a manner that ensures it will be compatible with and therefore reusable across as many current and future hardware platforms as possible. It should also be developed in a manner that minimises the work required to integrate it to any one potential hardware platform.

An important aspect of portability is ease of deployment: the ability to divide the waveform into arbitrary processor and tasking entities so that during deployment it can be split in different ways over a combination of platform processing resources. If you design correctly for optimal portability you also get reusability – the ability to reuse modules of waveform code in other developments – for free.

Optimal portability allows the maximum value to be derived from the waveform relative to the cost of its development, as the waveform can be reused on multiple platforms and repeat development due to redesign of a hardware platform is avoided. Also, an optimally portable waveform prevents waveform considerations from constraining hardware design decisions.

2.2 Maintainability

The waveform should be developed in a manner that allows it to be maintained. This ensures the waveform can be upgraded with fixes and new features, and therefore has a far longer lifespan. Portability and testability are important pre-requisites for maintainability.

2.3 Real-time Embedded Performance

The waveform should be designed and implemented in a manner that will allow best real-time embedded performance to be achieved across a range of potential platforms unknown at the time of development.

Note that the aim of SDR waveform development is to *optimise* these three factors. Some trade-off between them may be required (although not necessarily; often they reinforce one another) and as long as innovations in hardware continue, waveform design will always be chasing a moving target. Best-practice development is about accommodating these practicalities to derive maximum value from the waveform relative to the cost of its development.

3 Industry Status of SDR Waveform Development

Designing and implementing a commercial SDR Waveform that both meets the signal-in-space requirements of the corresponding communications standard or specification and correctly manages the relationship between the hardware and the software is extremely complex.

Despite this, most focus in SDR research and industry is on SDR hardware and on standardisation of the interfaces between hardware and software. Of roughly 120 papers that were submitted to the Forum's SDR'07 Technical Conference and Product Exposition, only a handful considered aspects of SDR waveform design and none looked at the problem in entirety.

Many initiatives – including the JTRS program and the resultant Software Communications Architecture (SCA) – have the outcome, if not the stated intent, of providing guidance on waveform design. However

again, the benefits the SCA can offer in this regard are limited to the interfaces between the waveform and the SDR/SCA platform, whereas the majority of gains in Portability, Maintainability and Performance (PMP) are achieved during design and implementation of the waveform itself. Building an SCA-compliant waveform does not therefore guarantee these have been optimised.

The most likely explanation for this 'hardware-up' approach lies in wireless communications' heritage in analogue technology. As this has been the norm for many decades, more industry expertise is found in hardware design.

As a result, the complexity of SDR waveform software is often poorly understood and SDR waveform development poorly executed, leading to spiralling development times, spiralling costs, and inferior products. For example, the commercial development of non-SDR, fixed platform complex radio software such as GSM, TETRA or APCO P25 is generally measured in tens of millions of US dollars. This figure is for software that can only be deployed on a single platform and does not factor in the added complexity required of a useful, generic SDR waveform implementation. To see more widespread adoption of SDR waveform development best-practice, and an attendant improvement in waveform products and reduction in waveform cost, the wireless communications industries must mature on two fronts.

Firstly, SDR Waveform specialisation must be cultivated. Secondly, the true and complete separation of SDR hardware and SDR waveform development must occur.

3.1 SDR Waveform Specialisation

Just as hardware design is executed by hardware experts, so should commercial SDR waveform design be executed by waveform experts.

SDR Waveform Engineers are familiar with a wide variety of different wireless communications standards and specifications. They also specialise in wireless technology and embedded software design best-practice for SDR hardware platforms.

A well defined wireless communications standard or specification is the starting point of any waveform. This is fundamental for interoperability; without it, no two independently developed waveforms or radios will communicate correctly or completely. Many problems stem from poorly defined standards, and thus attention during development of the standard or specification should be paid to exhaustively describing the air interface, rather than detailing how the waveform should be implemented.

That job belongs to the SDR Waveform Engineer. Due to their expertise, waveform engineers are efficient at taking a communications standard and:

a) Interpreting it.

b) Identifying deficiencies in the standard and compensating for these.

c) Combining the standard and additional proprietary customer features into a unified waveform requirements specification.

d) Deriving from this a design that is modular and optimises PMP.

e) Ensuring that the waveform will suit embedded platforms and not compromise embedded performance or resource use.

f) Designing and using appropriate SDR test and other tools to develop, verify and maintain the waveform.

g) Writing documentation to accompany the waveform.

These are each specialist skills that an SDR Waveform Engineer refines over many years and multiple waveform developments and deliveries.

3.2 Separation of Waveform and Platform Development

Optimal waveform design is achieved in complete isolation from a target or reference hardware platform.

In most waveform programs, the target or reference hardware platform is known during development. This can only degrade the quality of the waveform, because any design decisions made in order to accommodate the platform will immediately compromise the waveform's portability. Once this occurs, the waveform is married to the hardware and will not have a life beyond it.

Optimal waveform design is achieved when the waveform is developed in complete isolation from the platform, by planning from the specification or standard down rather than from the platform up.

This does not mean that the hardware is disregarded – an SDR Waveform Engineer must consider the particular characteristics of embedded platforms at every design decision. Rather, it means that embedded best performance for SDR platforms is planned into the design from the outset, in conjunction with portability and maintainability.

4 Commercial SDR Waveform Design at Etherstack

Etherstack is an independent, specialist commercial SDR waveform company that has been developing waveforms for radio manufacturers and defence clients internationally since the outset of commercial SDR.

As an independent waveform supplier, Etherstack is driven by commercial imperative to design SDR waveforms that are as flexible, portable, reusable and maintainable as possible – and that can also be optimised for best performance on small form factor embedded radio platforms. Etherstack also needs to be able to execute waveform development efficiently in order to minimise waveform cost.

Over many waveform deliveries, Etherstack has independently built and then refined the tools and development methodologies necessary to successfully optimise PMP. These are introduced below.

4.1 Specification and Design

As mentioned earlier, it is not the role of a communications standard or specification to describe how a waveform should be implemented. Usually too a customer will have proprietary features that require incorporation into the waveform design. Developing a new waveform therefore involves deriving specific waveform requirements from the combined requirements of the standard and the client, and then interpreting these in a design.

At Etherstack, the first step in this process of interpretation is to represent the design as functional layers that accord with those in the OSI model, if and where practical. These layers are then divided into many communicating modules by applying a disciplined and structured process of decomposition into consistent design entity types and collaborating patterns of entities. This process continues iteratively until the functional complexity of each entity is minimised. At this stage radio functions that are potentially susceptible to platform dependency have already been identified and isolated.

From there, the information flow between each communicating entity in the waveform is represented as a group of thoroughly defined interfaces, in order to implement the scenarios or use cases identified in the original waveform requirements. This process is illustrated in Figure 1.

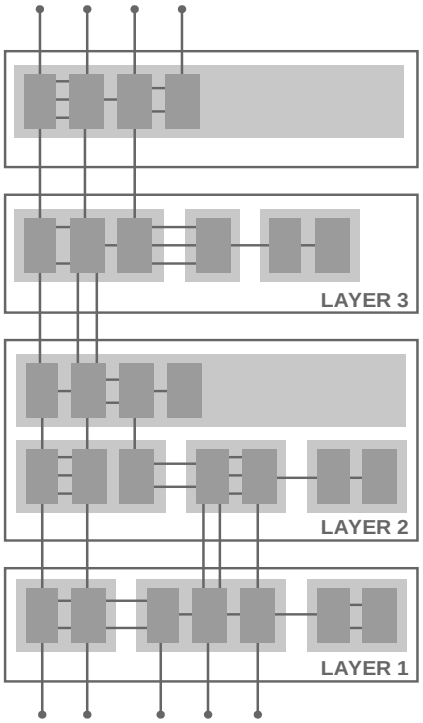

Figure 1: Functional Waveform Decomposition

4.2 Implementation

Once the design is complete, it is implemented as a Base Waveform.

The Base Waveform is the golden code from which each subsequent Target Waveform is derived during deployment on a new hardware platform. This approach is key for maximising the lifetime of the waveform: if a client updates or redesigns their hardware, the Base Waveform is deployed afresh on the new platform, rather than trying to move software designed solely for a legacy platform across to it. The Base Waveform also acts as a central repository for all waveform maintenance, and provides a reference for regression testing and Target Waveform integration verification.

The Base Waveform is built by mapping the design entities into well defined implementation entity types provided by Etherstack's Core Services. These Core Services are a set of sophisticated structural blocks and mechanisms, and combine to allow almost complete abstraction of the waveform from the underlying operating system and hardware platform. They also facilitate the isolation of each functional entity in the waveform to maximise deployment flexibility, and introduce to the waveform intrinsic diagnostic and test support that is compatible with Etherstack's Development, Simulation and Automated Test Environments.

The Base Waveform is implemented entirely in a general purpose programming language such as ANSI C to allow for efficient test and development cycles. As illustrated in Figure 2, it executes within Etherstack's Development and Simulation Environment on a laptop or desktop computer in concurrent operation with Etherstack's Automated Test Environment.

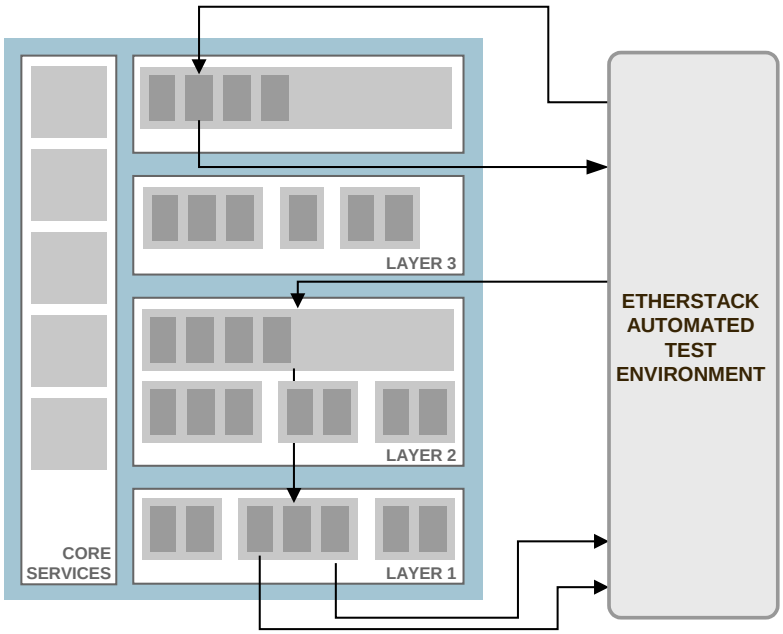

Figure 2: Testing of Implemented Base Waveform

4.3 Integration

It is at the integration stage that the advantages of the PMP design methodologies applied during the design and implementation phases are reaped.

The correct division of the waveform into isolated communicating entities of minimum functional complexity allows the waveform engineer complete flexibility to position different waveform entities over the different processing nodes on a heterogeneous platform in a manner dictated by the requirements of the platform itself. This is illustrated in Figures 3 to 6.

Integrating the Base Waveform to a target platform (and thus deriving the Target Waveform) therefore involves firstly deciding which waveform entities will execute on which radio devices according to resource availability and compatibility between particular entity functions and available processing nodes. The next step is to ensure the Transceiver, Audio, Data, Security, Database and Application waveform-to-platform interfaces are compatible. Lastly, the waveform entities are integrated to each relevant processing node on the platform, and optimised for best performance on that node if necessary.

The integrated Target Waveform is verified on the hardware platform via automated testing, using Etherstack's Automated Test Environment and identical test scripts to those applied to the Base Waveform. The thorough definition of interfaces between waveform entities during the design and implementation phases allows each to be tested in isolation, as well as for the waveform to be tested as a whole. This is illustrated in Figure 7, and testing is described further in Section 4.4.

It is worth recalling that due to Etherstack's Core Services, the operating system used on each processing node is immaterial; the waveform is almost entirely operating system and hardware platform independent.

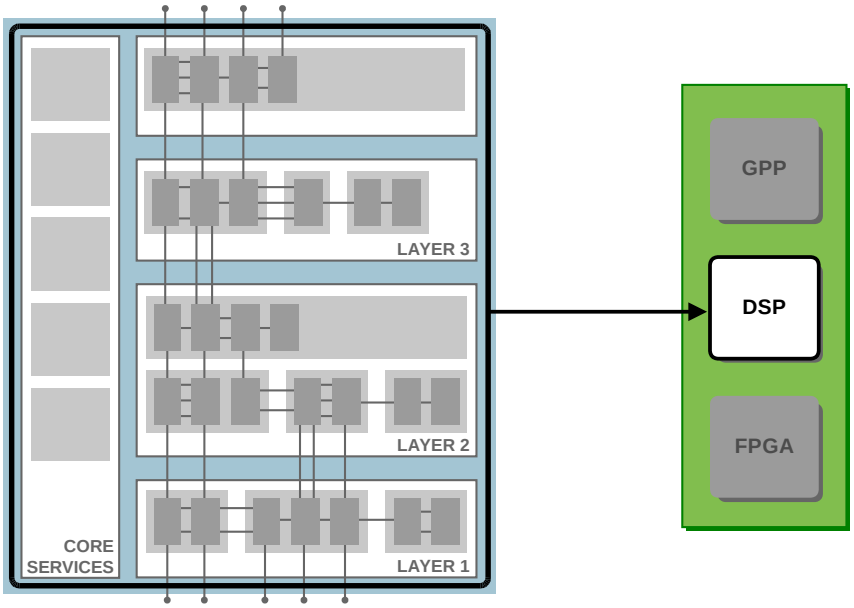

Figure 3: Integration of Entire Waveform to DSP

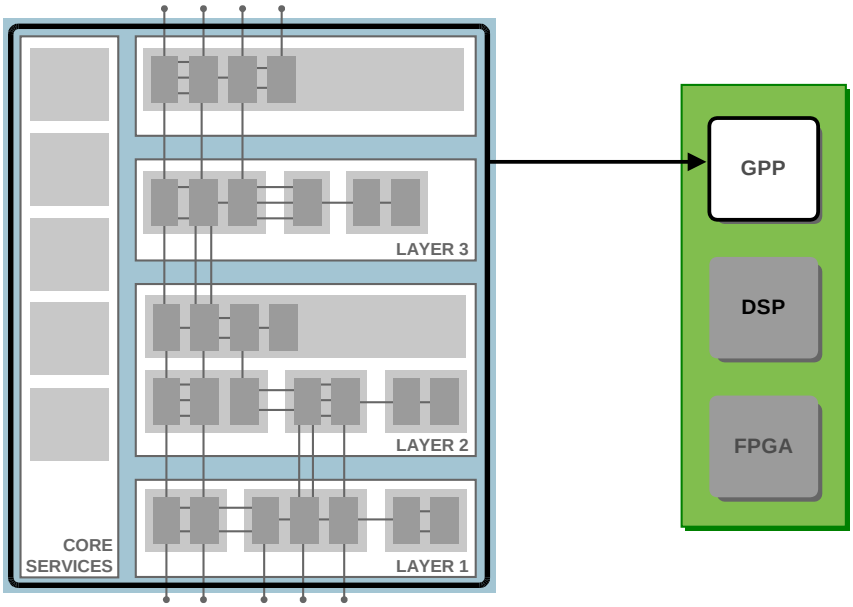

Figure 4: Integration of Entire Waveform to GPP

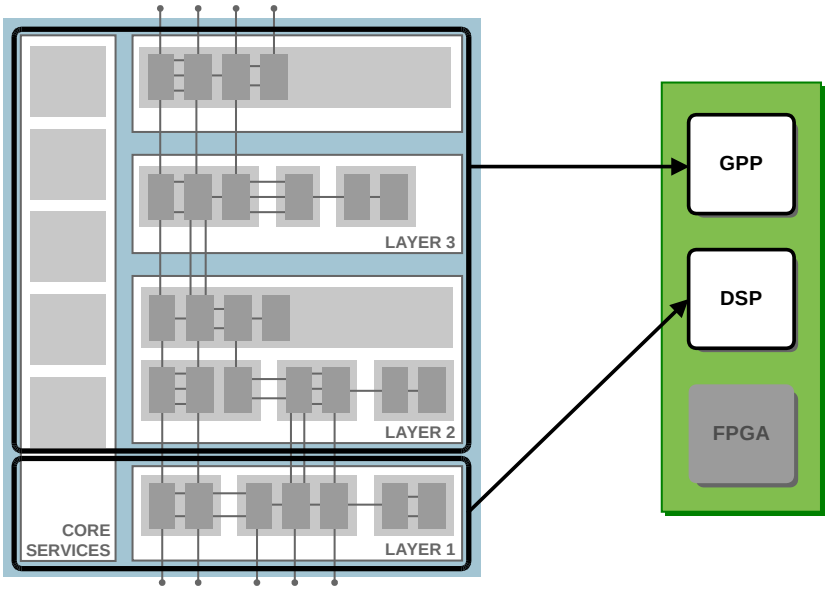

Figure 5: Integration of Waveform to GPP and DSP

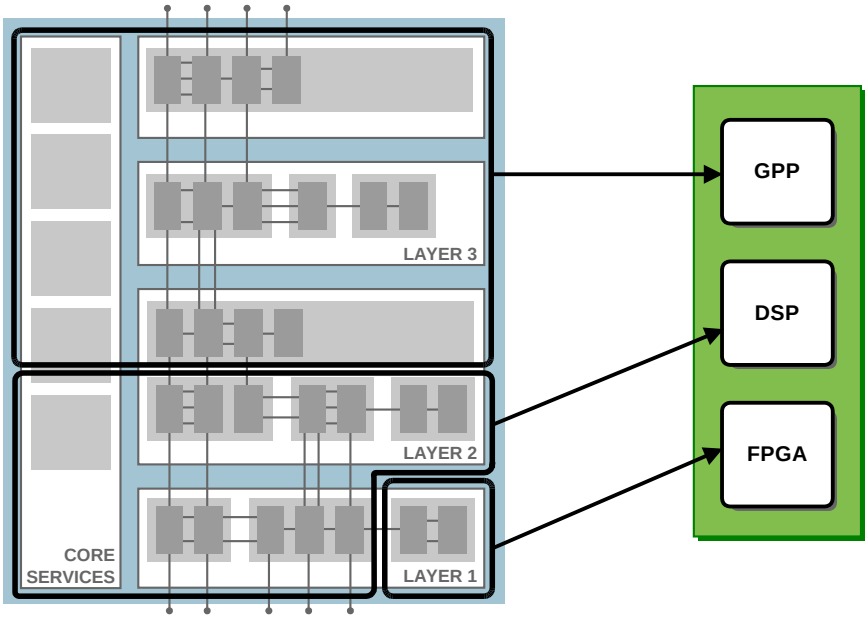

Figure 6: Integration of Waveform to GPP, DSP and FPGA

4.4 Maintenance

Any benefits gained due to the application of correct SDR waveform development methodology rely on a test environment built on the same principles. This must be compatible with both the simulation environment and all potential target platforms in order to verify the waveform during development, prove its correct implementation as a Base Waveform, prove the correct deployment of Target Waveforms and provide regression testing for ongoing maintenance of Base and Target Waveforms.

As with Etherstack's Core Services, Etherstack's Automated Test Environment took tens of engineer years to develop. During its development, the following features were identified as essential in order to support optimal waveform PMP:

a) A library of human-readable test scripts for each waveform that comprehensively covers the behaviour captured in the corresponding waveform requirements specifications. Each of Etherstack's test scripts contains many hundreds of test vectors that are applied to both internal and external interfaces on the waveform under test, in the process deriving human-readable test outputs.

b) Automation. To be exhaustive, waveform testing should be automated just as hardware verification is automated. This involves the automatic execution of each test script in a library in sequence, and of each test vector in a test script – accompanied by automated verification of the results.

c) Identical automated testing of the Base Waveform in simulation and a Target Waveform deployed on an embedded hardware platform. Etherstack builds intrinsic support for hardware agnostic testing into every waveform via the Core Services. Support for testing on the target

hardware is similarly built into the Automated Test Environment. This allows the same test scripts to be executed via the Automated Test Environment over both the Base Waveform in simulation on a PC, and over each Target Waveform on each target hardware platform. This is key for verifying not only that the Base Waveform meets the specifications, but that the Target Waveform does also – and that the behaviour of the two is entirely consistent.

d) The ability to replicate a problem scenario discovered during a Target Waveform's operation on an embedded platform in the Base Waveform, by executing a live log derived from the target hardware platform in the Base Waveform simulation environment.

e) Every internal interface must be testable. This ensures that the resolution of the software under test can be varied from the entire waveform down to a single entity – key if entities or groups of entities may be deployed on different processing nodes in a platform.

f) Graphic tools for visual representation of all signalling and state activity output during testing. These are valuable for diagnostics and as an aid to help clients understand the operation of the waveform. In addition to interface activity, Etherstack's test environment outputs diagnostic information about the waveform such as internal entity state, registered variable state, timer expiry and so on – all of which can be viewed graphically.

It is also worth noting that in addition to automated testing, successful maintenance of the Base Waveform and each of its Target Waveforms requires highly controlled source code management and versioning.

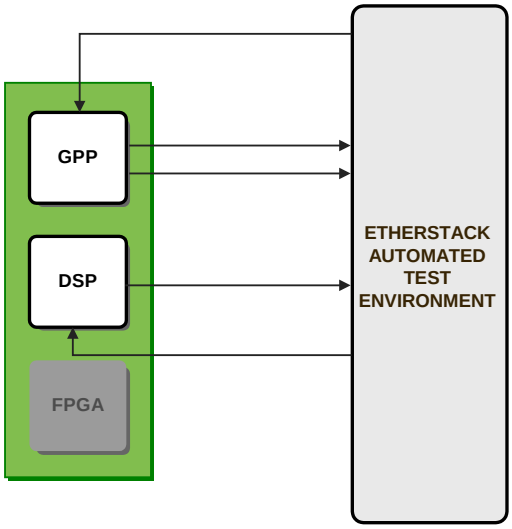

Figure 7: Testing of Waveform on Target Hardware

5 Porting to the Software Communications Architecture (SCA)

If a waveform has been designed correctly, porting it to the SCA in order to make it SCA compliant is uncomplicated. At Etherstack, this is achieved by merely identifying how the waveform will be divided into SCA Resources and Devices, and then applying "SCA Wrappers" that convert the identified

modules into the relevant Resource or Device. This is illustrated in Figure 8. Using this approach, manufacturers can deploy the save waveform, with exactly the same features, on both SCA and non-SCA radios.

Etherstack's Automated Test Environment is also capable of executing the test script suite over the SCA ports of the ensuing SCA waveform, so the full benefits of a non-SCA deployment are retained.

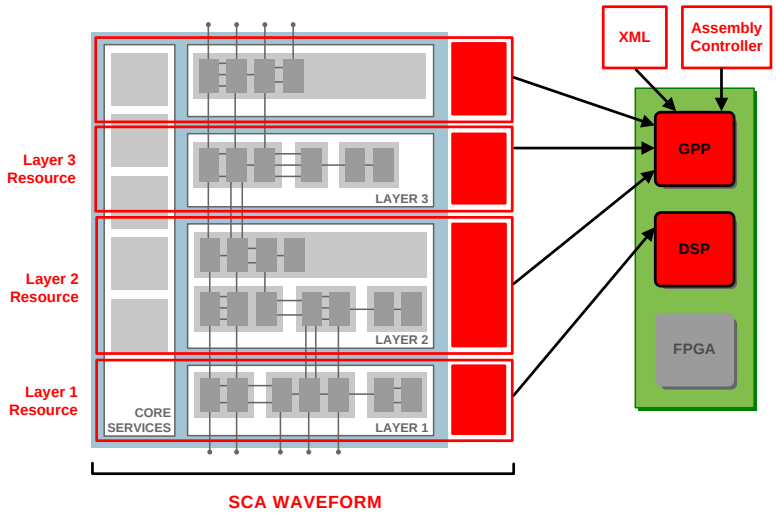

Figure 8: Porting an SDR Waveform to the SCA

6 Conclusion

Commercial SDR Waveforms involve complex signal-in-space and co-ordination functions, and a carefully managed relationship between hardware and software that aims to optimise waveform PMP - Portability, Maintainability and Performance.

To improve the quality and longevity of SDR waveforms in light of these requirements, this paper advocates the cultivation of SDR waveform development as an independent, specialist enterprise within the wireless communications industries.

It also recommends specific practices to apply at each phase of waveform development through specification, design, implementation, integration and maintenance, in order to optimise PMP and therefore capitalise on the value of SDR.

7 References

SDR Forum Cognitive Radio WG, *Cognitive Radio Definitions and Nomenclature*, SDRF-06-P-0009-V1.0.0, 30 May 2008 available at www.wirelessinnovation.org

CHAPTER 2 – IMEC'S LOW POWER SDR SOLUTION

1 Solution target

The multitude of existing and emerging high-throughput wireless standards coupled with stringent energy consumption constraints poses a great challenge on SDR baseband platform design targeted to handheld devices. This solution targets to support amongst others WLAN (IEEE802.11 family), WMAN (WiMAX), 3GPP-LTE, and digital broadcasting (which comes in different regional variations: DMB, ISDB-T, DVB-h). The high level targets are:

- Support up to 3 antennas
- Have 200Mbps throughput
- Maximum power consumption of 500mW
- Below 10mW idle power.

This makes the solution valid for today's high end mobile terminals.

2 Functional Partitioning

In the functional partitioning of the SDR platform flexibility is traded off versus energy efficiency. Indeed different functions of the platform have different flexibility requirements originating from different degrees of versatility of the algorithms. The duty cycle of a specific function determines the need for energy efficiency, while the amount of code to be mapped gives requirements for the programming efficiency. Table 1 gives an overview of the targeted functions and their design requirements.

Table 1: Fundamental design trade-offs for baseband platforms

Functionality	**Duty Cycle / Energy efficiency**	**Versatility / Flexibility**	**Programming Efficiency**
Automatic Gain Control	Very High	Very Low	Low
Synchronization	High	Low	Low
Modulation / Demodulation	Medium	High	Very High
Forward Error Correction	Medium	Medium	Medium
Platform control	Medium	High	Very High

Exploiting these trade-offs has resulted in multiple heterogeneous processors for mapping the different functionalities.

- Digital front-end (DFE) tiles achieve packet detection with limited need in flexibility and code mapping productivity (<5% of the total application code) but very high energy efficiency requirements. They are heavily exploiting sub-word parallelism and techniques to reduce the instruction and data memory overhead.

- Baseband engines handle the modulation and demodulation of the burst. The processing architecture which is based on the combination of a Coarse Grain Array (CGA) processor coupled with vector processing provides an appropriate trade-off between performance, energy efficiency and flexibility.

- Forward error correction (FEC) cores are typically implemented on reconfigurable hardware components. In a first generation mobile SDR baseband platform this design decision was valid based on the low need for flexibility, combined with a huge complexity if implemented on a generic DSP (see Figure 1). However, in recent and emerging wireless standards, an evolution towards more flexibility in the FEC as well has been observed. As a consequence, an application-specific architecture for advanced FEC algorithms (turbo- and LDPC-codes) is desired for future SDR baseband platforms. FEC decoding algorithms are indeed hard to map to typical baseband processors (mainly because of the extensive interleaving requirements). They also require an effective solution for the severe data access bottleneck.

- A platform controller is responsible for the MAC functionality and the PHY macro-pipeline scheduling. This typically runs on a RISC processor (ARM in represented case in Figure 9).

3 Baseband Platform

3.1 Description

IMEC's first generation SDR digital baseband platform is shown in Figure 9.

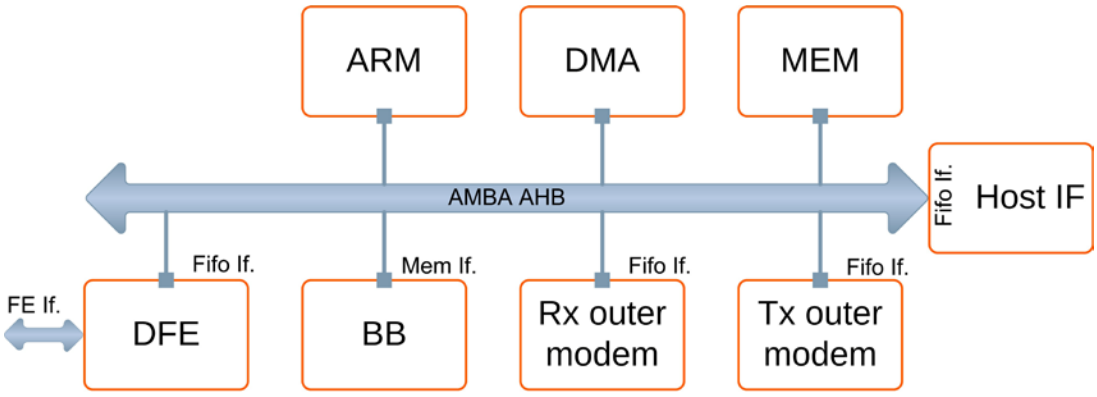

Figure 9: Heterogeneous MPSoC approach for mobile SDR baseband platform

The platform controller (ARM) is responsible for the MAC functionality and the PHY macro-pipeline scheduling. The Digital Front-End (DFE) consists of 3 identical tiles containing the digital transmit and receive logic to interface to a single radio front-end. The architecture of the two baseband engines is based on a CGA processor coupled with vector processing template

called ADRES, which provides an appropriate trade-off between performance, energy efficiency and flexibility. The outer modem blocks are implemented in a hardwired fashion. The different cores are connected by a segmented (AMBA) bus. This bus has been designed to achieve the required bandwidth while avoiding significant over-dimension.

In order to clarify the typical operation of a heterogeneous SDR platform in its different working modes, a short explanation is given below. More details on the specifics of the different components follow in the next sections.

- In idle mode, only the DFE Rx of the platform is operational. Upon detection of a signal at its input by the AGC, the 'SyncPro' (see section 3.2) will attempt initial synchronization, and if a valid signal is detected, the more heavily power consuming rest of the platform is then activated. The ARM will start controlling the communication on the platform, in order to receive data and communicate according to the MAC protocol.
- The ARM as a central controller sets up DMA transfers between the different components on the platform, receives interrupts when tasks are finalized and acts upon them.
- In transmit mode, the following sequence is set up: data is received through the host interface. The data is encoded, consequently modulated by the baseband engine, and finally transmitted to the DAC after up-sampling filtering in the DFE.
- In receive mode, the data received from the ADC by the DFE and down-sampled, consequently it is demodulated in the baseband engine, and decoded in the outer modem, from which it is transferred to the host interface.

3.2 Implementation facts

- TSMC 90G Multi-VT technology
- 32 sqmm core die area, 20 sqmm active area
- 6.7 Mbit memory (121 instances)
- 4 power domains, 8 clocks
- 270 I/O pins, 267 supply pins
- 2 ADRES processors, each with:
    - 33 memory macro @ 400MHz
        - 32KB instruction cache
        - 128-entries config mem
        - 64KB data scratchpad
    - 128KB IMEM @ 200Mhz
    - 400MHz Clock rate Worst Case Commercial operating conditions

3.3 Baseband Engine

The SDR-ADRES is a programmable solution for Inner Modem processing. It is designed as a platform component for flexible baseband processing.

The FLAI-ADRES is an instantiation for Inner Modem processing of a generic template. ADRES (Architecture for Dynamically Reconfigurable Embedded System) is a flexible architecture template that consists of a tightly coupled very long instruction word (VLIW) processor and a coarse-grained reconfigurable array. The integration of the VLIW and the reconfigurable array in a single architecture with two virtual functional views has advantages over state-of-the-art architectures. In most of the present solutions the coarse grain array operates next to a host processor, typically a reduced instruction set computer (RISC) that executes the

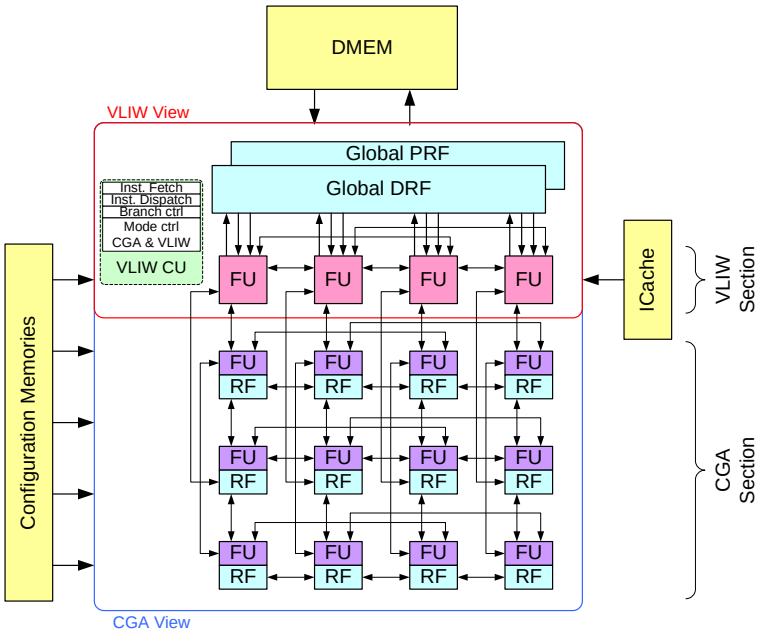

Figure 10: ADRES 4X4 instance extended with 4-way SIMD

remaining parts of the application. The communication between this processor and the configurable accelerator results in programming difficulties and communication overhead that can significantly reduce overall performance. Combining host processor and accelerator in one architecture leads to simplified programming and removes the communication bottleneck.

3.3.1 CGA architecture exploration framework

The CGA framework considered, namely ADRES (Mei B, 2003), comprises a parameterized array of interconnected Functional Units (FU) which have local Configuration Memories (CM) or Instruction Buffers (IB) and Register-Files (LRF). A limited subset of these units is instead connected to a shared multi-ported register-file (Global Register File, GRF), enabling their operation also as part of a standard VLIW (Very Long Instruction Word) processor. ADRES instances achieve low energy consumption relying on the small distributed configuration memories (instruction loop buffers) and maximizing the data forwarding between FUs (passing data from one FU to another without going through a RF) and the use of the LRFs and domain specific instructions. An essential element of this framework is the retargetable C compiler, namely DRESC (Mei B, 2002), which targets both the VLIW and CGA modes. DRESC transparently maps data-flow dominated loops onto the CGA and schedules the rest of the code

in the VLIW part. The data communication between the VLIW processor and the reconfigurable array, bottleneck in other CGA-based architectures, is implicitly performed through the GRF and shared memory. The compiler supports a broad range of different architectures which are described in an XML-based language. This includes arbitrary combinations of three types of basic components: functional units, storages resources such as register files and memory blocks, and routing resources that include wires, muxes and busses. DRESC can achieve high ILP by means of modulo scheduling (Rau R. B.,1995), a widely used software pipelining technique that executes multiple iterations of a loop in parallel. To hide the control-flow and enable further parallelization inside loops, the FU supports predicated operations for conditional execution.

Additionally, the programmer can provide specific instructions to the architecture through the use of intrinsics. These have the appearance of typical C function calls, which are replaced during the preprocessing step by programmer specified low-level instructions. This way the programmer can model the vector operations required to exploit DLP and other domain specific instructions such as a complex multiplier

3.3.2 Exploration methodology

With the ever increasing complexity of wireless platforms, most of the existing design challenges are related to the design methodology. The explosion of the design space together with the need of finding highly optimal solutions calls for new approaches that enable fast but still meaningful explorations. In the concrete case of processor architecture exploration, where the design space can become critically large, the accuracy of the energy or performance estimate can be relaxed in order to broaden the exploration scope. In (Novo D., 2008) an extensive energy-performance exploration of a CGA-based SDR processor is presented. This approach targets sufficient relative accuracy on the optimization metrics, which assures meaningful comparisons between different instances, while the absolute accuracy is relaxed and traded off against simulation time

3.3.3 SDR ADRES instance

The top level of the resulting optimized architecture is depicted in Figure 11. The main CPU in this processor is a 3-issue VLIW. The processor has an asynchronous reset, a single external system clock and a half-speed (AMBA) bus clock. Instruction and data flow are separated (Harvard architecture). A direct-mapped instruction cache is implemented for the VLIW CPU with a dedicated 128-bit wide instruction memory interface. The level-1 data memory contains 4Kx32-bit memory per bank. These banks can also be accessed externally through an AMBA2-compatible slave bus interface (which connects to the memory interface just like an additional load/store unit in the processor). The CGA configuration memories (128 contexts) and special registers are also mapped to the AMBA bus interface via a 32-bit internal bus. This way, the CGA configuration memories and the level-1 scratch-pad data memories can be accessed via DMA transfers.

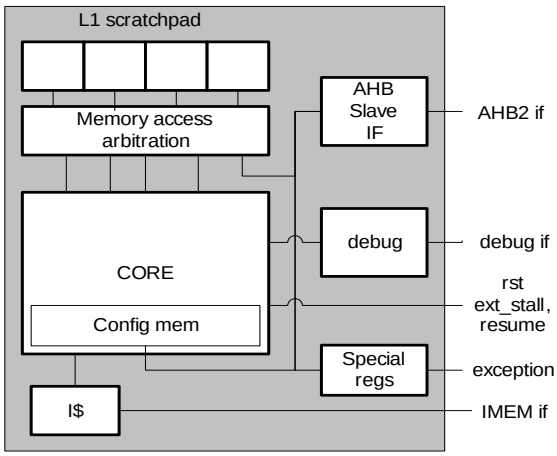

Figure 11: Processor top level architecture

3.3.4 Implementation facts

The architecture described above is implemented to reach a 400MHz clock rate in worst-case conditions when implemented in 90nm technology. Hence it delivers up to 16 units x 4 way SIMD x 400MHz = 25.6GOPS (16-bit) as foreseen to be sufficient to implement 2x2 20MHz MIMO-OFDM at 100Mbps+. To achieve such a clock frequency a general purpose option process has been selected. Although it is leakier, this process has a better power-delay product than the low power process usually considered for embedded applications. Leakage in operation mode can be tackled with a multi-VT design and, in standby, with third-party substrate biased standard cell library and memory macros.

3.3.5 Performance facts

The processor was dimensioned to enable the execution of next generation broadband cellular communication and wireless LAN standards. To evaluate its performance and power consumption, a set of benchmarks was selected corresponding to transmitter and receiver baseband processing in IEEE 802.11n and 3GPP-LTE. For each benchmark, the table below presents the portion of the execution time spent in accelerated mode as well as the Instruction per Cycle (IPC) ratio obtained in that mode.

Table 2: CGA performance

Benchmark	**Time in accelerator mode**	**IPC**
11n 64QAM TX	64%	10.75
11n 64QAM RX	56%	9.99
11g 64QAM TX	53%	11.05
11g 64QAM RX	56%	10.37
LTE TX	99%	8.76

The average IPC over the different benchmarks is 10.18. This number includes regular and SIMD operations, all counted as one operation. On average about 45% of the operations are SIMD operations. The resulting utilization of the accelerator's FUs is 10.18/16 = 63.65%. This figure is particularly high when considering that it is obtained with compiled ANSI C-code.

Furthermore, the prototype power consumption was estimated based on gate-level simulation. The netlist resulting from the physical design, back-annotated with capacitance and parasitics information extracted from the final layout was simulated with Mentor ModelsimTM. From such simulation, accurate gate level activity profiles were generated. Synopsys PrimePowerTM was used to evaluate the CPU and accelerator power consumption based on the activity profile and the layout information. Results are summarized in Table 3. Peak CPU and CGA powers are given for the typical design corner (V=1V, nominal process, T=25C). Leakage is extrapolated to typical leakage corner (V=1V, nominal process, T=65C). The data memory hierarchy power consumption is added in each case.

Table 3: Prototype processor power consumption

	Active(typical)	**Leakage (typical)**	**Leakage (T=65C)**
CPU+DMEM	75 mW	12.5 mW	25 mW
CGA+DMEM	310 mW	12.5 mW	25 mW

Figure 12 presents a further breakdown of the accelerator active power. A major fraction of 37% goes to the interconnect sub-system which include buffers, multiplexers and pipeline registers between the FUs. The FUs themselves, the configuration memories (CMEM) and the data memory (DMEM) consume 29%, 14% and 11% resp. The shared and distributed register files consume 7 and 2%.

In conclusion, the exploration of a hybrid CGA-SIMD ADRES processor has lead to a suitable

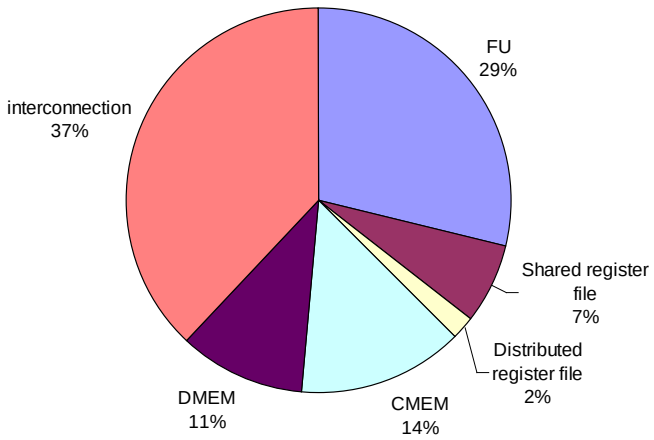

Figure 12: Power breakdown in CGA mode

architecture and implementation. The prototype design has shown to enable the real-time execution of several transmitter and receiver benchmarks out of next generation broadband cellular and wireless LAN communication standards with an average power consumption of 250mW.

3.3.6 Availability

The SDRE-ADRES is available as an open IP block with following parts:

- ADRES/DRESC Development framework
- SDR-ADRES instance (Configuration, VHDL-code)
- Complementary reference C-code and Matlab System model for 802.11a/n, 3GPP-LTE and 802.16e
- A platform integration example
- Documentation

The standard AHB®-bus interface to the platform makes seamless integration with existing platforms possible.

3.4 Digital Front-End

Programmable solutions intrinsically suffer from higher power consumption when compared to dedicated solutions. In the considered platform, the digital front-end tiles, which implement signal detection and pre-synchronization functions, require both very high energy efficiency and sufficient programmability to implement detection of different standards on the same tile. The key to this combination lies in the hierarchical power up of the DFE functionality.

3.4.1 DFE Architecture

The DFE consists of multiple ‘tiles’ (3 in the considered case, as represented in Figure 13). A single tile contains the digital receive and transmit logic to interface to a single antenna.

The transmitter part of a DFE tile consists of a buffer and a VLSI interpolation filter. The interpolation filter is based on an optimized implementation of a 19-taps half-band filter (with hamming window) for a fixed up-sampling of factor two. A start command can be issued allowing the samples to be clocked out towards the analog front-end through the filters. The transmit (TX) buffers have a programmable threshold that triggers an interrupt once the number of available samples falls below this threshold. This interrupt is handled by the platform controller.

The receiver part of a DFE tile contains a chain made of the VLSI decimation filters, the buffers and compensation units for DC offset and carrier frequency offset (CFO). The decimation filter impulse response is derived from a 19-taps half-band filter with hamming window performing an energy efficient factor two down-sampling. Next to the data path, two dedicated micro-processor cores are implemented. The first handles the front-end automatic gain control (AGC) and the DFE power management. The second core is optimized for time synchronization.

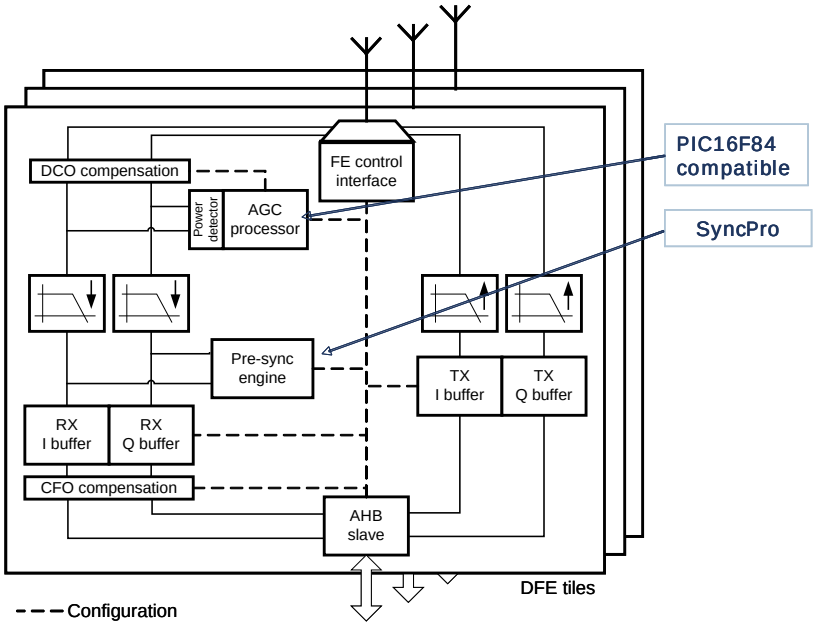

Figure 13: DFE Architecture

The components of the DFE cooperate in the hierarchical wake up approach according to the state machine depicted in Figure 14. In a typical SISO receive scenario the DFE is powered on and the command to start scanning is given. AGC functionality will be activated and I&Q sample data will be monitored. If the AGC has detected a possible preamble, it will assert the AGCdone signal and the clocks of the receive filters, the data buffer and synchronization processor will be activated. This state remains until the synchronization processor signals a synchronization event or until a timeout occurs. Upon synchronization, an interrupt towards the platform will be generated to notify that data is available at the data interface. The platform will signal the end of the packet and the state is moved to the Scan.

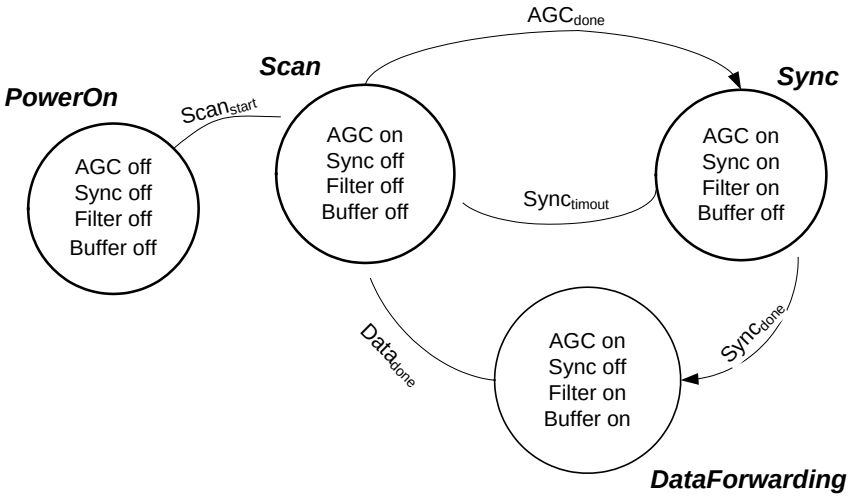

Figure 14: DFE State machine

3.4.2 Power detection and AGC controller

A power detection unit with variable delay line of 8 or 32 samples determines the received signal power and is capable of DC offset estimation. A dedicated microcontroller is used to implement the AGC algorithm that removes DC offset and optimizes the ADC range based on the analog front-end control pins. The controller also determines which other parts of the DFE RX are activated after an AGC event is detected (gradual wakeup). The controller architecture is depicted in Figure 15. It is clocked at the sample rate (40MHz). It has an instruction memory of 512 14-bit words and a data scratchpad of 32 8-bit words, both implemented as register file macros. The instruction set and the architecture of the controller are compatible with an industry-standard Microchip PIC16F84. This allows for reuse of the available tools, including c-compilers and debuggers.

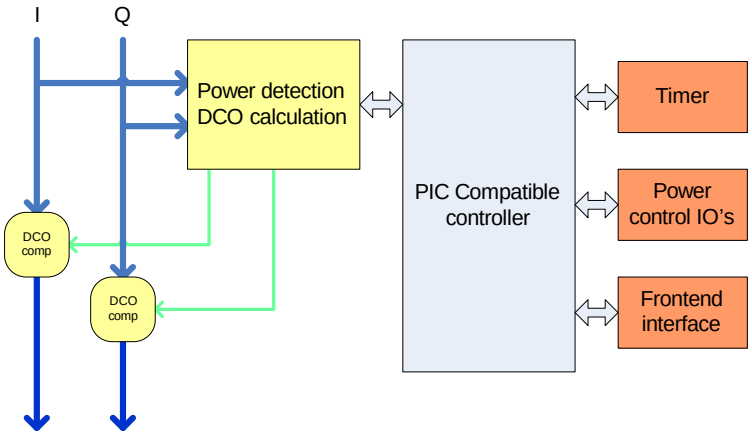

Figure 15: Power detection and AGC controller

3.4.3 Synchronization engine

The dedicated application specific processor (ASIP) as 'synchronization engine' performs signal detection and coarse time synchronization (Schuster, 2006). The prime target applications considered for the design of the ASIP are time synchronization for 802.11a and 802.16e. Besides, the engine is designed to be flexible enough to support future standards such as 3GPP LTE.

Time Synchronization usually consists of two parts:

1. First one correlates over the input signals to expose the periodic structure of a packet preamble.
2. Then, one scans the correlation results for a peak above a certain threshold. The index of the peak indicates the start of the packet.

The code consists of computation intensive and control intensive parts. To implement these kinds of algorithms, the SyncPro ASIP (Figure 16) with specialized instruction set, features four types of instructions: Scalar instructions for control, vector instructions for computation intensive tasks and instructions to generate and evaluate vectors.

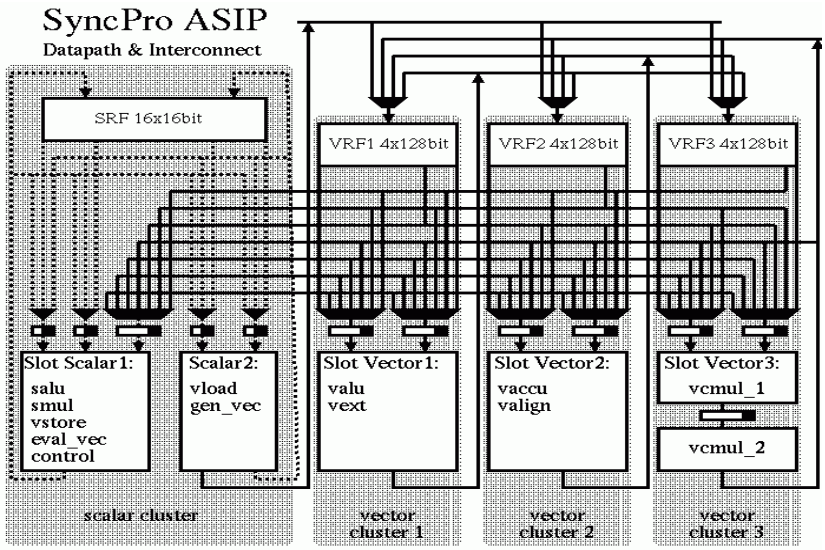

Figure 16: SYNCPRO synchronization ASIP

In the conception of the ASIP architecture, energy efficiency was considered a key design target. Therefore, special attentions were paid to the selection of the instruction-set, parallelization, storage elements (register files, memories) and interconnect. The resulting ASIP architecture is a clustered VLIW processor with scalar and vector slots, as shown in Figure 16. It was modeled thus with LISATek 2005.2.1 (www.coware.com). The processor has 5 issue slots organized in 4 clusters. The two slots in the first cluster operate on 16bit data and the three other clusters on 128bit wide vectors. Each of the vector clusters contains a vector register file. The vector register

files have 3 read and one write port. Two of the read ports are dedicated to the FUs in the particular slot and one is used for operand broadcasting. For operand broadcasting, a very flexible software controlled interconnect is implemented.

3.4.4 Implementation facts

Table 4: DFE Physical Design Results (TSMC 90G VDD=1V T=25C)

Active Area		mm2
3 DFE Tiles		2.7
Memories		1.4
Total Std. Cell		1.3
Layout		4
Power consumption (TC)		**mW**
Leakage	total	5.72
Scan (1 tile)		7.1
Synchronizing (1 tile)	max	22.74
Data forwarding (RX SISO)		11
Data forwarding (RX MIMO)		23
IO		
Internal		529
External		184
Clocks (200MHz, 40/80MHz)		8

3.4.5 Performance facts

In the above table, notice the differences in power consumption in different states. Clearly significant power saving is achieved by residing in state Scan state, and only switching the synchronization engine on upon power detection by the AGC. State Sync is an intermediate state, which will decide either to proceed to packet decoding, or if no synchronization is found within a predefined time limit to return to idle mode. The operation of the DFE can be interpreted as 'reacting' on incoming data and events.

Table 5: Syncpro Validation Results

Synchronization	Wireless test	Load on SyncPro
802.11a/n	OK	~ 50%
802.16e	OK	~ 80%
3GPP-LTE	Under development	?

Table 5 shows that the Syncpro ASIP achieves the required performance and flexibility within very tight energy constraints. It can be calculated that the processor delivers a theoretical maximum performance of 5 GOPS (32bit equivalent) at a peak power of 25mW. Energy efficiency is hence 200 MOPS/mW (fully loaded).

3.4.6 Availability

The DFE is available as an open IP block with following parts:

- VHDL-code
- Complementary reference firmware for 802.11a/n and 802.16e
- A platform integration example
- Documentation

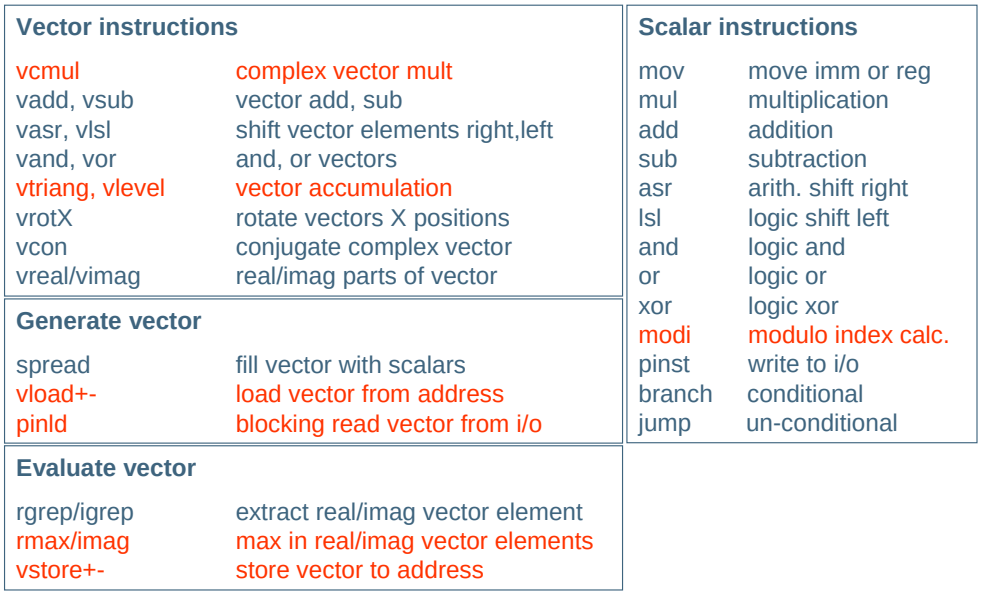

Figure 17: SyncPro instruction set

The standard AHB®-bus interface to the platform makes seamless integration with existing platforms possible. The control interface to the front-end is a 16bit bidirectional interface that can be controlled by both the DFE for low latency interaction and the AHB-bus for general configurations. The AGC algorithm is programmed in "C" on micro-controller, while the synchronization algorithm is programmed in assembly on an Application Specific Instruction Processor. See Figure 17 for an overview of the instruction set.

4 References

Mei, B (2003). "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse Grained Reconfigurable Matrix", FPL 2003, pp. 61-70. Lisbon, Portugal.

Mei, B (2002). "DRESC: A Retargetable Compiler for Coarse-Grained Reconfigurable Architectures," FPL2002, pp. 166-174. Montpellier, France.

Novo, D (2008), "Energy-performance exploration of a CGA-based SDR processor", Journal on Signal Processing Systems: Springer

Schuster, T (2006), "Design of a Low Power Pre-Synchronization ASIP for Multimode SDR Terminals", SAMOS VI, July 2006, Samos, Greece

Rau, R. B. (1995). "Iterative Modulo Scheduling," HP Lab, Tech Report: HPL-94-115. www.coware.com

CHAPTER 3 – THE SANDBRIDGE SANDBLASTER SB3500 SDR MPSOC BASEBAND PROCESSOR FOR 4G HANDSETS

Abstract

The Sandblaster architecture is a high-performance vector architecture targeted at digital signal processing applications. The Sandblaster 1.0 architecture was targeted at implementing the physical layer of 3G wireless standards, with peak data rates of up to 15 Mbps. In this paper, we describe an object code compatible version 2.0 of the Sandblaster architecture, which is targeted at the 4G standards, which support higher data-rates and more complex algorithms.

To achieve the necessary performance to implement 4G standards, the 2.0 version of the architecture extends the 4-MAC Sandblaster 1.0 architecture to a 16-MAC architecture, with accompanying changes in the width of the vector register file. It also introduces new vector operations that are specialized to key algorithms specified in the 4G standards.

The architectural enhancements have been implemented in the SB3500 chip fabricated in low power 65nm process technology. The chip is fully functional, provides nearly 30 GMACs of DSP performance at 600MHz and validates the design objectives of the 4G standards.

1 Introduction

Historically, processor-based Software Defined Radios (SDRs) have not had enough performance to implement modern communications systems. Digital Signal Processors (DSPs) are now capable of executing many billions of operations per second at power efficiency levels appropriate for handset deployment. This has brought Software Defined Radio (SDR) to prominence.

To enable physical layer processing in software, processors should support many levels of parallelism. Since algorithms often have stringent requirements on response time, multithreading is an integral technique in reducing latencies. In the Sandbridge SB3011 processor design [Glossner, 2004] thread-level parallelism is supported by providing hardware for up to eight independent programs to be simultaneously active on a single Sandblaster core. This approach hides the latency in physical layer processing.

In addition to thread-level parallelism, the processor also supports data-level parallelism through the use of a vector unit. In the inner kernel of signal processing or baseband routines, the computations appear as vector operations of moderate length. Filters, FFTs, correlations, etc., all can be specified in this manner. Efficient, low power support for data level parallelism effectively accelerates inner loop signal processing.

To accelerate control code, the processor supports issuing multiple operations per cycle. Since control code often limits overall program speed-up due to Amdahl's Law, it is helpful to allow

control code and vector code to execute simultaneously. This is provided through a compound instruction set and multithreaded organization. The Sandblaster core provides instruction level parallelism by allowing multiple operations to issue in parallel. For example, a branch, an integer, and a vector operation may all issue simultaneously [Jinturkar, 2003]. In addition, many compound operations are specified within an instruction class, such as load with update, and branch with compare.

Obtaining full utilization of multiprocessor resources has historically been a difficult challenge. Much of the programming effort can be spent determining which processors should receive data from other processors. Execution cycles are often wasted for data transfers. Statically scheduled machines such as Very Long Instruction Word architectures and visible pipeline machines with wide execution resources complicate programmer productivity by requiring manual tracking of up to 100 in-flight instruction dependencies. When non-associative DSP arithmetic is present, nearly all compilers are ineffective and the resulting burden falls upon the assembly language programmer.

In the process of implementing these systems, a number of techniques have been developed for both low power optimization and efficient execution. Real-time constraints add another level of complexity. The remainder of this paper discusses the techniques and optimizations used.

Experience with programming various 3G standards on the SB3010 and SB3011 chip implementations [Glossner, 2006] of the original Sandblaster 1.0 architecture identified certain areas for architectural improvement to support higher data rates. Also, the next set of wireless standards (so-called 4G), such as WiMax and LTE, would require more processing power than could be provided by a low-power implementation of the 1.0 architecture. The Sandblaster 2.0 architecture was developed so as to allow the software implementation of the physical layer of the various 3.5G and 4G standards.

The most significant change to the architecture is the introduction of 16-wide vector operations, in contrast to the 4-wide vector operations of the original 1.0 architecture. Also, the 2.0 architecture contains instructions that are specialized for the efficient execution of key 3.5G and 4G kernels.

Section 2 of this paper gives an overview of the architecture. Section 3 focuses on the 16-wide vector operations. Section 4 discusses additional enhancements. The SB3500 chip is presented in Section 5. The performance achievable from these changes is discussed in Section 6. We present related work in Section 7 and provide concluding remarks in Section 8.

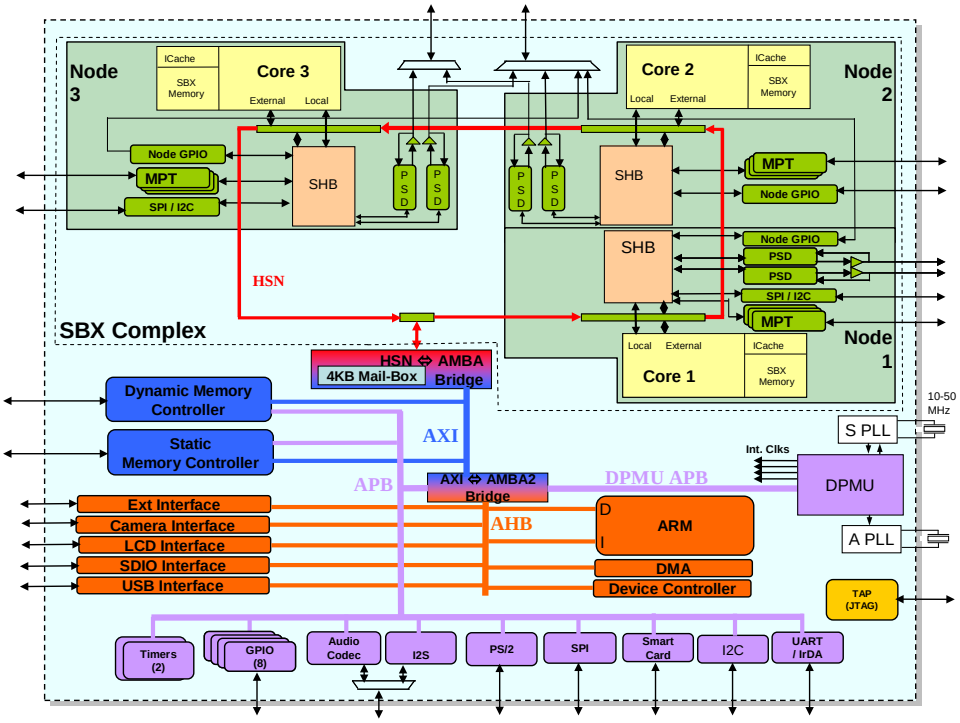

Figure 18: SB3500 Chip

2 Architecture Overview

The Sandblaster architecture uses a 64-bit instruction word, which consist of a serial S-bit and up to three 21-bit compound operations. If the S-bit is not set (e.g. 0), then all three operations are executed in parallel, otherwise they are executed serially. If an operation in a serial instruction is a taken branch, then operations subsequent to the branch will not be executed. All branches have 8 byte boundaries. In particular, it is not possible to branch into the middle of an instruction word, even if that instruction is a serial instruction.

There are 4 categories of operations: branch, integer, memory and vector. A parallel instruction can contain at most one operation from each category; a serial instruction has no such restriction.

The Sandblaster architecture specifies several heterogeneous register files. In general, a register file is accessed by only one category of instructions. A list of the more commonly used registers and their properties can be found in Figure 18. Only the general purpose and vector registers can be loaded from or stored to memory. The other registers must be moved to/from a general purpose register using the copy from/to special purpose register (cfsr/ctsr) operations. The cfsr/ctsr instructions are also used for accessing system specific registers.

The Sandblaster architecture is a load/store architecture; i.e. only load or store operations access memory. These operations use register+immediate addressing, where the base is provided by a General Purpose Register (GPR).

2.1 Vector Unit

The vector operations and registers experienced the biggest change in the upgrade of Sandblaster 1.0 to Sandblaster 2.0.

The vector registers in the 1.0 architecture were 160 bits wide, and were connected to memory by a 64-bit data-path. In the 2.0 architecture, the vector registers are 256-bit wide and connected to memory using a 256 bit data path. Further, the mask and accumulator registers have been expanded from 4 & 40 bits to 32 and 64 bits respectively.

Name	Number	Bits	Category	Notes
General Purpose	16	32	Integer, Memory	can be loaded/stored
Condition	8	1	Branch	cb0 also set/used by integer
Jump Target	2	32	Branch	
Loop Count	2	32	Branch	
Vector	8	256	Vector	can be loaded/stored
Mask	4	32	Vector	
Accumulator	4	64	Vector	
Search	2	32	Vector	

Table 6: Register files

In most cases, a SIMD operation in the 1.0 architecture operated on 4 values in parallel. By contrast, the 2.0 architecture operates on 16 short (16-bit) values or 8 (32-bit) integer values in parallel. Also, the 1.0 operations were fairly general-purpose. In the 2.0 architecture, the general purpose operations are augmented by operations specialized for key kernels of 4G wireless communications systems.

2.1.1 Element-wise operations

The element-wise operations include common operations such as logical, shift and arithmetic operations that read 2 registers, perform 16 short or 8 integer operations in parallel, and write the results back to a third register.

Element-wise multiplies are only done on short values; a single operation can specify 16 short multiplies to be done in parallel. In one set of variants, either the upper or lower 16 bits of the 32 bit product are written as the result. Alternatively, the full 32 bit product may be written to 2 registers. There are also multiply-and-add or subtract variants.

Complex multiplies treat the register file contents as though they were alternating short real and imaginary values, so that a register contains 8 short complex numbers. A complex multiply uses 4 short multipliers, so implementing 8 complex multiplies in parallel would have required 32 multipliers. Instead, the Sandblaster 2.0 complex multiply operations multiply either the upper or

lower halves of registers together, writing the complex product to the upper or lower half of the result register.

2.2 Reduction operations

Reduction operations take the multiple elements of a vector and combine the values into a scalar result, which is then written to an accumulator register. The multiply reduction operations do element-wise multiplies, and then sum the products together.

Complex multiplies can also be reduced; in this case, the 32 bits of the real part of the sum are stored in the lower 32 bits of the accumulator and the imaginary part of the sum is stored in the upper 32 bits of the accumulator.

Masked sum operations can also be reduced. These come in two variants; one adds up all the sums. The other variant adds up half the sums into the lower 32 bits of the accumulator and the adds up the other half into the upper 32 bits of the accumulator.

Search operations are also a type of reduction operation. The elements of a register are compared against each other and an accumulator to find the maximum (or minimum). These instructions also modify a pair of registers - the position and count registers - so that the position of the maximum/minimum can be determined.

To search an array for the minimum value, first the count and pos values are cleared using the ctsr register, and the act register is set to 0x7fff_ffff. After a sequence of rsearchmin operations, act will contain the smallest value seen so far, count will contain the number of array elements examined, and pos will contain the position where the smallest value was encountered.

2.3 Specialized operations

Significantly, the Sandblaster 2.0 architecture introduced groups of vector operations designed to improve the performance of specific algorithms required by the 4G standards.

One group of operations is used to implement fast-fourier transforms (FFTs). These operations do 4 complex multiplies per cycle, producing 8 complex elements of the result. Depending on the operation, the values are either written to one register or to the upper/lower halves of two registers.

Galois field arithmetic support is provided by operations that do polynomial multiply, multiply-reduce and multiply-and-add and compute the polynomial-modulus.

Viterbi decoding adds operations that perform 16 viterbi butterflies in parallel, reading 3 registers (2 for the state and 1 for weight) and writing the resulting state into 2 registers. The trace-back bits are written to the accumulator registers. There are also flavors of the masked sum operations that are used to compute the branch metrics from the input samples.

Turbo decoding is supported by operations that compute the forward, backward and likelihood values. The turbo-decode operations assume that the constraint length of the convolutional

coders is 3. They execute two steps of the forward (or backward) pass per operation. In addition, they may combine the forward (backward) steps with the result of prior backward (forward) steps to compute the likelihood. This likelihood is stored in an accumulator register.

Dibit turbo decode is supported by similar operations. However, dibit turbo-decoding only does one step of the forward/backward pass per operation.

2.4 Other operations

The other vector operations rearrange data in the registers. These include:

- packing/unpacking 8 bit data to 16 bit data
- packing/unpacking 16 bit data to 32 bit data,
- shuffling the elements of a pair of register
- copying the contents of an accumulator to all elements of a register
- rotating register pairs
- shifting accumulator data into a register

2.5 Fixed point operations

Digital signal processing typically uses fixed-point arithmetic. Consequently, all the vector operations that do addition, subtraction, multiplies, and left-shift have a fixed-point version. Further, operations such as complex-multiplies and the specialized operations come only in the fixed point versions.

Fixed point arithmetic differs from the standard 2s complement arithmetic in several ways:

- a fixed point multiply is further multiplied by 2
- if the result of an arithmetic operation overflows the number of bits available, it is saturated to the maximum/minimum representable value
- when converting from a type with more bits to fewer bits, the upper bits are used.

When an operation does a sequence of fixed-point arithmetic computations, one can saturate after each intermediate computation. Alternatively, one can keep all intermediate results at full precision (i.e. use enough bits so that there is no possibility of overflow) and then saturate the combined result. The vector operations from the Sandblaster 1.0 architecture saturated after each intermediate operation; the operations introduced in the 2.0 architecture saturate only the final result.

2.6 Rotation/Shifting

Many wireless kernels involve data-streaming, which involve operating on subsequences of data offset from each other by one or two positions. For example, consider a 16-tap FIR filter:

```
for( j=0; j<M; j++ )
  sum = 0;
  for( i=0; i<16; j++)
    sum += x[i+j]*c[i];
  z[i] = sum
```

In this example, the dot-product of c[0..15] with x[0..15] is computed, then the dot-product with x[1..16], x[2..17] and so on. The 2.0 architecture supports this idiom through register pair rotation.

In register pair register rotation, each pair of even/odd registers is treated as a circular shift register. The values in them can be shifted by 1, 2 or 4 shorts. The pseudo-code below illustrates the 1-short (i.e. 16 bit) rotate:

```
val_e0 = ve[0];
val_o15 = vo[15];
for( i=0; i<15; i++ )
  ve[i] = ve[i+1];
  vo[i+1] = vo[i-1];
ve[15] = val_o15;
vo[0] = val_e0;
```

Note that elements in the even and odd registers are shifted in opposite directions by a rotate.

Array x from in the example can be streamed using the register rotation operation, *rrot*, as follows:

- Load x[0..15] into an even register
- Load x[16..31] into an odd register in reverse order
- After every complete execution of the inner loop, rotate the even/odd pair by 1.
- After 16 iterations of the outer loop, load the next 16 values of the x array into the odd register, in reverse order

Shifting can also be used to save a series of accumulated values. Reduction operations as well as certain specialized operations store their result into accumulator registers. The *rshift* operation can then shift the value from an accumulator into a vector file, as shown in the following pseudo-code:

```
for( i=0; i<15 i++ )
  ve[i] = ve[i+1];
ve[15] = aca;
```

In this example, 16 bits from the accumulator are shifted into an even register. If the target had been an odd vector register, it would have been in the opposite direction, as in the rotate instructions. Different variants of the instruction can shift different 16, 32, and 64 bits from an accumulator into the vector register. The *rshift0* operation additionally clears (i.e. sets to 0) the shifted accumulator register.

A 16 tap FIR filter would use a *rmulreds*, *rshift0* and *rrot* operation. Of these, 1 operation is to do the actual computation and the other 2 are overhead to rearrange the data. This is a fairly common occurrence. Consequently, the architecture has some operations that combine an operation with accumulator shifting and clear and register-pair rotation. These include the *rmulreds1r* operation.

2.7 I-cache

The Sandblaster 2.0 architecture adds operations to control the instruction cache. These are part of the branch category, and use the jump-target registers to specify an instruction address. This address is used by operations to flush a set and to prefetch an instruction into the cache.

The prefetch instructions allow the cache to be warned up, so that the initial cold-miss penalty can be avoided. This improves the worst-case run-time of an algorithm, thereby improving real-time performance.

2.8 Integer unit

The integer unit has added several operations. These are:

- Parity: compute the even parity of a word
- Galois field: compute the polynomial product and modulo. These operations are similar to the operations in the vector unit.
- Reversal: swap the bits or bytes of a word.
- Traceback: help traverse the trace-back array generated by the Viterbi vector operations. One operation generates the address of the next word, and the other extracts the appropriate bit.

2.9 DMA

In most systems, a direct memory access (DMA) engine is provided as a memory-mapped peripheral. In the Sandblaster 2.0 architecture, the DMA has been made part of the architecture. The DMA control registers are part of the architected state, and accessed via cfsr/ctsr instructions. A process can be swapped even if it has a DMA operation in flight; the DMA is architected so that a DMA operation can be halted in-flight and the control registers copied out. After the process has been resumed, the control registers can be restored, and the DMA restarted from where it was halted.

The DMA also implements scatter and gather functions. Thus, apart from block copies, the DMA can be programmed to implement gathers. The scatter and gather can occur at a granularity of 1, 2, 4 or 8 bytes.

3 Programming Techniques

A good programming model should adequately abstract most of the programming complexity so that 20% of the effort may result in 80% of the platform utilization [Silvén, 2005]. While there are still some objections to a multithreaded programming model [Lee, 2006], to date this model is widely adopted, particularly after the introduction of the Java programming language.

With concurrent multithreaded hardware and a multithreaded software programming model, it is possible for a kernel to be developed that automatically schedules software threads onto hardware threads. It should be noted that while the hardware scheduling may be fixed, the software should be free to use any scheduling policy desired. The POSIX pthreads open standard provides cross platform capability as the library can compile across a number of systems including Unix, Linux, and Windows.

The Sandbridge approach includes a complete parallelizing tool chain which removes the need for tedious DSP assembly language programming. A well-known problem for DSP compilers is saturating arithmetic [Glossner, 2003]. Fixed point (fractional) datatypes are non-associative and require special treatment within a compiler. The Sandblaster processor overcomes these limitations by providing both vector architectural execution and compiler algorithms that can determine the type of the variable and thus maintain serial semantics even under parallel execution. The compiler is also able to automatically generate threads for the processor [Jinturkar, 2004]. It ensures all synchronization and provides automatic vector generation for efficient code generation. Furthermore, ultra-fast simulation, profiling, and debugging of code that is embodied in the Sandblaster development environment is a key enabler of fast application development.

Rather than designing custom blocks for every function in the communication system, a small and power-efficient core can be highly optimized and replicated to provide a platform for broadband communications - an SDR processor capable of executing operations appropriate to broadband communications. This approach scales well with semiconductor generations and allows flexibility in configuring the system for future specifications and any field modifications that may be necessary.

3.1 General Programming

The Sandblaster processor provides access to a fixed number of threads. These threads are called hardware threads or contexts. However, an application, written in C, can contain a virtually unlimited number of POSIX threads (pthreads, also called software threads), which are scheduled on the hardware threads by the real-time operating system (RTOS).

The Sandblaster programming interface provides the ability to create, destroy, and join the threads through the common include library <pthreads.h>. This open mechanism is completely portable across multiple processors and compilers. In fact, the compiler uses the exact same pthreads mechanism when automatically generating threads. The underlying operating system supports the inherent pthreads schedule model to prioritize and schedule threads. The interface also models a number of peripherals include A/D and D/A converters, General Purpose IO (GPIO), and control peripherals. The RTOS has been kept lightweight to reduce overhead.

3.2 Coding Guidelines

We have previously published coding guidelines that allow our compiler to take advantage of parallel resources [Jinturkar, 2003]. Most of these guidelines are generic in the sense that if

followed, compilation on other platforms (e.g. x86) should be improved. All are high-level language practices. The most salient points are summarized:

- Minimize the use of malloc().
- Pass arrays explicitly and not enclosed within structures
- Use arrays of shorts since the most efficient vector datatype in the Sandblaster processor is 16 bits.
- Use floating point only when necessary as it is emulated.
- Avoid unrolling loops manually as the compiler can do it more efficiently.

4 SB3500 Chip implementation

The architecture enhancements have been incorporated into the SB3500 chip implementation. As shown in Figure 18 at the beginning of this section, the chip contains 3 Sandblaster 2.0 cores. Each of the cores typically runs at 600MHz while providing twice the power efficiency of the SB3011 chip design. The peak performance of the chip is nearly 30 GMACs at handset power dissipation levels. As with the SB3011 chip implementation, the chip contains a full ARM subsystem with all the peripherals required to operate a smart phone device including USB 2.0, camera, video, smart card, SIM, keyboard, and LCD ports. The chip is enhanced with a split transaction AXI bus to allow HD video processing while performing 4G baseband communications. The chip is fabricated in 65nm technology and is fully functional.

5 Results

With a tool set capable of automatically generating parallel DSP code and a fully functional high-performance low-power chip, the physical layers of multiple communications protocols including WCDMA [Glossner, 2003], GSM/GPRS [Kalavai, 2006], 1xEVDO [Watanabe, 2006], TD-SCDMA [Shamsunder, 2004], NTSC Video Decode [Kotlyar, 2006], WiMax [Iancu, 2006], WiFi [Ramadurai, 2006], GPS [Iancu, 2004], AM/FM radio [Iancu, 2003], DVB [Iancu, 2004], and SINCGARS [Beheshti, 2004] have been implemented. Real-time response has been achieved and the effectiveness of our approach has been validated.

In addition to communications systems, the processor is also capable of multimedia. A number of applications have been developed including MP3 [Jinturkar, 2006], MPEG4 [Agrawal, 2005], and H.264 [Ramadurai, 2005].

Based on the analysis of these systems combined with 4G WiMax and LTE analysis, we have implemented kernels for various wireless standards, and measured the number of instructions used. Some of the results are summarized in Table 7. As can be seen, the combination of specialized operation support and combined compute/rotate/shift operations allow us to achieve close to optimal performance.

Algorithm	Type/Phase	Instructions
FIR	16-tap real	1 /output
	16-tap complex	2 /complex output
FFT	Core	N/6*(logN-1)
	bit-reversal	N/3 for N point FFT
64 state Viterbi	forward pass	2 per bit
	Traceback	2 per bit
Turbo decode	Forward	21/32 per bit
	Backward+ likelihood	22/32 per bit
Dibit turbo decode	Forward	12/8 per dibit
	Backward+ likelihood	13/8 per dibit

Table 7: Performance

6 Conclusions

We have presented a description of the Sandblaster 2.0 architecture. We have described the major enhancement to the base 1.0 architecture that includes wider vectors and application specific instruction support. We have briefly described the SB3500 chip implementation that incorporates the architectural extensions. The chip validates the performance and power design objectives of the architecture. Based on previously implemented systems along with the kernel analysis we can implement future 4G standards completely in software on implementations of this processor.

7 References

Agrawal, Sitij (2005). S. Agrawal, S. Jinturkar, V. Ramadurai, M. Moudgill, and J. Glossner, "Multithreading MPEG4 Encoder on Sandblaster DSP," in Proceedings at the 2005 Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC), Santa Clara, CA, October 24-27, 2005.

Beheshti, Babak (2004). B. Beheshti, J. Glossner, D. Routenberg, L. Zannella, and P. Steensma, "Evaluation of Military Waveform Processing on a COTS Reconfigurable SDR Processing Platform," in Proceedings of the SDR'04 Technical Conference and Product Exposition, Volume A, 16-18 November, 2004, Scottsdale, Arizona, pp. 147-151.

Glossner, John (2003). J. Glossner, D. Iancu, E. Hokenek, and M. Moudgill, "A Reconfigurable Baseband for 2.5/3G and Beyond," in Proceedings of the 2003 World Wireless Congress, San Francisco, CA, May 27-30, 2003, pp. MC.11-1-6.

Glossner, John (2003). J. Glossner, S. Dorward, S. Jinturkar, M. Moudgill, E. Hokenek, M. Schulte, and S. Vassiliadis, "Sandbridge Software Tools," in Proceedings of the 3rd annual Systems, Architectures, Modeling, and Simulation (SAMOS) Conference, Samos, Greece, July 21-24, 2003, pp. 142-148.

Glossner, John (2004). J. Glossner, M. Schulte, M. Moudgill, D. Iancu, S. Jinturkar, T. Raja, G. Nacer, and S. Vassiliadis, "Sandblaster Low-Power Multithreaded SDR Baseband

Processor," in Proceedings of the 3rd Workshop on Applications Specific Processors (WASP'04), Stockholm, Sweden, September 7th, 2004, pp. 53-58.

Glossner, John (2006). J. Glossner and D. Iancu, "The Sandbridge SB3011 SDR Platform" Proceedings of the Symposium on Trends in Communications (SympoTIC'06), Invited keynote, Bratislava, Slovakia, June 24-26, 2006.

Iancu, Daniel (2003). D. Iancu, J. Glossner, H. Ye, Y. Abdelilah, and S. Stanley, "Reduced Complexity Software AM Radio," Proceedings of the Symposium Trends in Communications (SympoTIC '03), pp. 122-125, Bratislava, SLOVAKIA, October 26 – 28 2003.

Iancu, Daniel (2004). D. Iancu, J. Glossner, H. Ye, M. Moudgill, and V. Kotlyar, "Rake Receiver Enhanced GPS System," in Proceedings of the SDR'04 Technical Conference and Product Exposition, Volume A, Scottsdale, Arizona, November 16-18, 2004, pp. 97-105.

Iancu, Daniel (2004). D. Iancu, H. Ye, Y. Abdelilah, E. Surducan, and John Glossner, "On the Performance of Multiple OFDM Receivers for DVB," in Proceedings of the Joint IST Workshop on Mobile Future & Symposium on Trends in Communications (SympoTIC'04), Bratislava, Slovakia, , October 24-26, 2004, pp. 1-4.

Iancu, Daniel (2006). D. Iancu, H. Ye, E. Surducan, M. Senthilvelan, J. Glossner, V. Surducan, V. Kotlyar, A. Iancu, G. Nacer, and J. Takala, "Software Implementation of WiMAX on the Sandbridge SandBlaster Platform," in Proceedings of the 6th Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS'06), Samos, Greece, July, 2006.

Jinturkar, Sanjay (2003). S. Jinturkar, J. Glossner, E. Hokenek, and M. Moudgill, "Programming the Sandbridge Multithreaded Processor," in Proceedings of the 2003 Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC), March 31-April 3, 2003, Dallas, Texas.

Jinturkar, Sanjay (2004). S. Jinturkar, J. Glossner, V. Kotlyar, and M. Moudgill, "The Sandblaster Automatic Multithreaded Vectorizing Compiler," in Proceedings of the 2004 Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC), Santa Clara, California, September 27-30, 2004.

Jinturkar, Sanjay (2006). S. Jinturkar, V. Ramadurai, V. Kalashnikov, G. Nacer, and J. Glossner, "Implementing MP3 Decoder on Sandblaster DSP," in Proceedings of the 2006 Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC), Santa Clara, CA, November, 2006.

Kotlyar, Vladimir (2006). V. Kotlyar, D. Iancu, J. Glossner, Y. He, and A. Iancu, "Real-time Software Implementation of NTSC Analog TV on Sandblaster SDR Platform," in Proceedings of the 4th Karlsruhe Workshop on Software Radios, Karlsruhe, Germany, March 22-23, 2006, pp. 171-176.

Kalavai, R. (2006). R. Kalavai, M. Senthilvelan, S. Agrawal, S. Jinturkar, and J. Glossner, "Implementation of GSM/GPRS Physical Layer on Sandblaster DSP," in Proceedings of the SDR'06 Technical Conference and Product Exposition, Orlando, Florida, November, 2006.

Lee, Edward (2006). E. Lee, "The Problem with Threads," Computer Magazine, IEEE Press, May 2006.

Ramadurai, Vaidyanathan (2005). V. Ramadurai, S. Jinturkar, M. Moudgill, and J. Glossner, "Multithreading H.264 Decoder on Sandblaster DSP," in Proceedings at the 2005 Global

Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC), Santa Clara, CA, October 24-27, 2005.

Ramadurai, Vaidyanathan (2006). V. Ramadurai, S. Jinturkar, S. Agarwal, M. Moudgill, and J. Glossner, "Software Implementation of 802.11a blocks on Sandblaster DSP," in Proceedings of the SDR'06 Technical Conference and Product Exposition, Orlando, Florida, November, 2006.

Shamsunder, Sanyogita (2004). S. Shamsunder and J. Glossner, "Reduced Complexity Software Receivers for TD-SCDMA Downlink," in Proceedings at the 2004 Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC), Santa Clara, California, September 27-30, 2004.

Silvén , Olli (2005). O. Silvén and K. Jyrkkä, "Observations on Power-Efficiency Trends in Mobile Communication Devices," in Proceedings of the 5th Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), Lecture Notes in Computer Science, Vol. 3553, Samos, Greece, July 2005, pp. 142-151.

Watanabe, S. (2006). S. Watanabe, Y. Kunisawa, D. Kamisaka, "A Software Radio Implementation of CDMA2000 1xEV DO on a Single DSP Chip Designed for Mobile Hand Terminal," in Proceedings of the IEEE Vehicular Technology Conference, , Montréal, Canada 25 – 28 September 2006.

CHAPTER 4 – THE EVP VECTOR PROCESSOR AS ENABLER FOR SOFTWARE-DEFINED RADIO IN HANDHELD DEVICES

1 Introduction

Future mobile handsets will need to support multiple wireless communication links, potentially including 2G up to 4G cellular, wireless local-area network (WLAN), personal-area network (PAN), broadcast, and positioning. A layered structure of such a future network, adapted from (Becher, 2001), is shown in Figure 19.

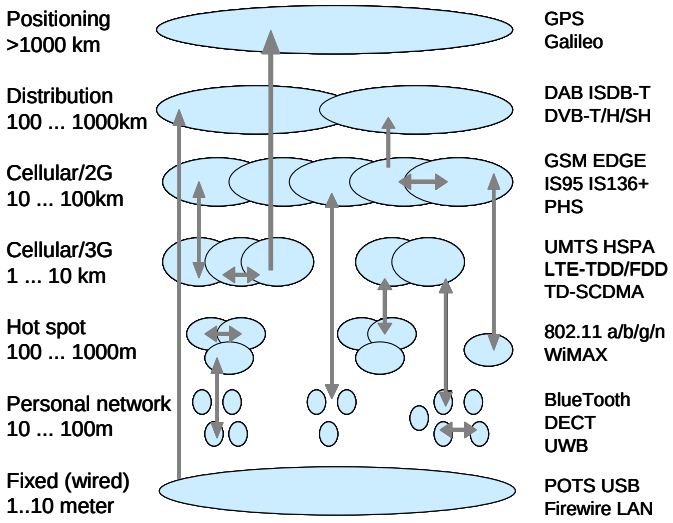

Figure 19: Layered structure of a seamless future network

These layers are to be integrated in a common, flexible, and seamless IP core network, supporting global roaming and a single access number per user. This requires both horizontal (intra system) and vertical (inter system) handover, as indicated by the arrows.

For each of these layers there exists a multitude of standards, often regional. Some handheld devices may have to support multiple standards per layer, e.g. in a world phone.

In a typical scenario, multiple standards have to be supported simultaneously in standby mode, plus one standard active. In a high-end scenario, however, several links may be active, e.g. GSM (standby), DVB-T (data downlink), UMTS (uplink), Bluetooth (BT), and GPS.

Individual standards typically evolve over the years towards higher bit rates, more features, and more services. For example, 3G cellular standards will need to support high-speed packet access (HSPA), and for WLAN multiple-antenna schemes are being implemented (IEEE 802.11n).

For a given standard, new algorithms are continuously developed to improve performance (lower BER, more efficient spectrum usage, reduced power consumption). Having a method to upgrade handsets is then attractive, for example by flexibility through programmability.

The combination of the above trends forms a powerful argument for so-called software-defined radio (SDR) (Becher, 2001).

In Section 2 we identify the global architecture and various stages of the baseband processing of an SDR. In Section 3, based on prior analysis of baseband algorithms a list of requirements for an SDR processor is presented. Based on these requirements a vector processor is presented in Section 4: the EVP (Embedded Vector Processor), being applied in 3G and beyond. Section 5 will discuss the software development tools for such an architecture, and explain the EVP SDK in some detail. Detailed load numbers for UMTS baseband kernels, including benchmark data, are presented in Section 6.

2 HW architecture for SDR baseband

Estimates for the computational load [GHz] for baseband processing are given in Figure 20, with (Kokozinski, 2002) as main source. Interestingly, the numbers roughly appear to apply to both non-optimized programs on a Pentium 3 as well as to optimized (assembly) programs running on current DSPs used in GSM handsets.

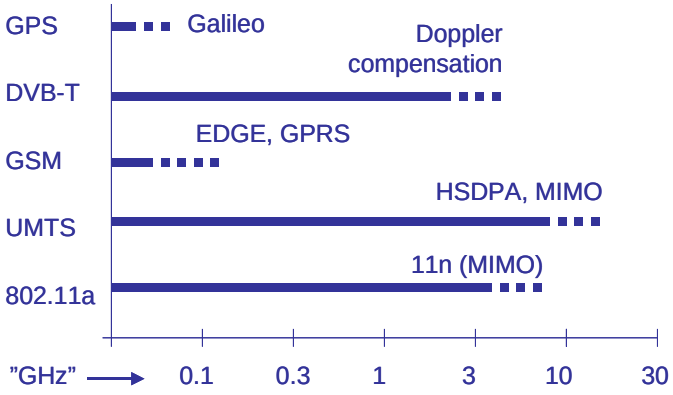

Figure 20: Load estimates for various SDR standards

The digital baseband processing for SDR can be split into three stages: a filter stage, a modem stage, and a codec stage, as shown in Figure 21. The loads are more or less evenly distributed across these three baseband stages. Nevertheless, the stages have very different characteristics, as analyzed in more detail in (van Berkel, 2004).

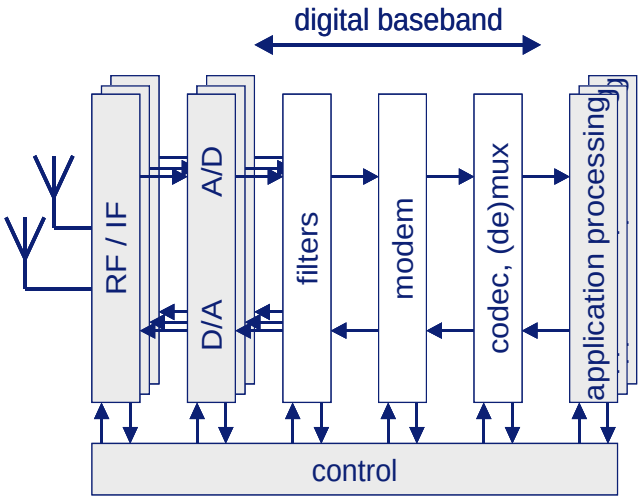

Figure 21: A crude SDR architecture, with the baseband section split into filters, modem, and channel codec

2.1 Filter stage

Various transmitter and receiver filters are required for band limitation, like (root) raised cosine filters and sample-rate conversion. Given their high computational load (e.g. 2-5 billion multiplications and additions per second for UMTS), their regularity, and the commonality among the algorithms involved, full programmability would add insufficient value to compensate for the additional power consumption. A configurable multi-standard filter approach is usually more appropriate.

2.2 Modem stage

The modem stage, sometimes called “inner transceiver” or “signal conditioner”, appears to be the most diverse across the different standards. It includes functions such as rake reception, synchronization, joint detection, channel estimation and equalization, FFT, OFDM (de)mapping, interference cancellation, etc. Furthermore, new modulation schemes are proposed within the continuous evolution of standards to improve throughput and performance. This happens even for older standards, like the evolution still ongoing in EDGE. Next to this, manufacturers are challenged to differentiate their products by improving algorithms to reduce Bit Error Rates (BERs) or transmit power for the same BER. In this stage dBs can be gained or lost by choosing and optimizing the right or wrong algorithms. In our opinion, this is the stage where programmability offers most value (Moerman, 2008).

2.3 Codec stage

The codec stage, sometimes called “outer transceiver”, involves a variety of functions: (de)multiplexing, (de)puncturing, (de)interleaving, and a variety of channel codecs (e.g. convolution, Turbo, Reed-Solomon). The performance of these functions is determined by standard algorithms, and allows little differentiation among manufacturers. Given the

considerable similarities among standards and algorithms, and given the considerable processing requirements for higher bit rates (e.g. > 100 Mbps for LTE class 4), a fully programmable solution does not appear to be justifiable, although certain parts like the address generation for (de)interleaving might require a simple form of programmability.

2.4 Baseband hardware architecture

The observations as mentioned above on the different baseband stages result in a proposal for a multi-standard hardware architecture (Figure 22), comprising:

- a general purpose microcontroller for link/MAC layer processing and for controlling baseband and RF tasks;
- a configurable filter processor;
- one or more programmable vector processor(s) for number crunching, mostly in the modem stage;
- one or more multi-standard weakly programmable channel decoders, e.g. Viterbi, Turbo.

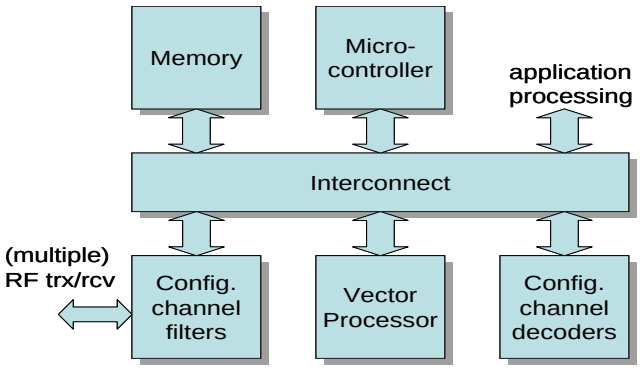

Figure 22: Schematic hardware architecture for SDR/BB

3 Vectorization of baseband kernels

Making vector parallelism explicit in program code is commonly called “vectorization”. Depending on the algorithm at hand, this can be relatively straightforward, or it may require some ingenuity. Unfortunately, the state of the art of vectorizing compilation today cannot fully exploit the architectures discussed in Section 4 *and* at the same time achieve *efficient* code for SDR (see Section 5). Although we have been able to generate code of acceptable efficiency for some algorithms, we rely on manual vectorization for the time being.

Vectorization of several key algorithms is presented in an earlier paper, for various types of algorithms (Van Berkel, 2004). Algorithms investigated included a Golay correlator for UMTS-FDD, a rake receiver for UMTS-FDD, Symbol-timing estimation for 802.11a, Fast Fourier Transforms, and Viterbi decoding. In the rest of the article we assume a vector processor that supports P (P a power of 2) identical operations to be executed in parallel (Single Instruction Multiple Data, SIMD), as well as load (store) operations of P adjacent values from (into) a vector memory (note: for the first generation of the EVP, P has been set to 16).

3.1 SDR vector processor requirements

From the vectorized baseband kernels above we can now collect the requirements for an SDR vector processor, as worked out in more detail in (Van Berkel, 2004).

R1. In addition to the 16-bit data types common for DSPs, 8-bit data types are useful for the incoming radio signals. They allow a higher memory density and twice the parallelism in the SIMD data path. Next to this, efficient handling of complex data types is very beneficial.

R2. Vectorization of most presented algorithms scale well with *P* (although in practice, a limit will occur, e.g. for 64 complex samples for the 802.11a algorithms). In order to be prepared for future evolution, it is valuable to keep the vector processor *scalable*, that is, to parameterize its architecture and its implementation (including tools) by P.

R3. Orthogonal to pure SIMD parallelism, VLIW (Very Long Instruction Word (Hennessy, 2003)) parallelism can further accelerate algorithms, as rake (pipelined memory access + code generation + de-correlation + integration) as well as FFT (pipelined multiply + add/subtract + shuffle).

R4. The speed-up that can be achieved by combining vector and VLIW parallelism is limited by the fraction of the program code that can be vectorized. This limitation is known as Amdahl's Law (Hennessy, 2003). For example, when 90% of an algorithm can be sped up by a factor *P*=32, the overall speed up is still less than a factor eight. In order to counter this limitation, a parallel speed-up is *also* required for the non-vectorizable parts of an algorithm. An example of this is the trace-back function of the Viterbi decoder. To further counter Amdahl's Law, parallel address calculations and loop control are critical, even more so than for traditional DSPs (Lapsley, 1994).

R5. Beyond "pure" SIMD, shuffling of data within a vector (Gwennap, 1998) is key to many algorithms, as FFT ('butterflies') and Viterbi trellis construction. Such a shuffle operation can also support vector rotation, which can be used to implement non-aligned read access to vector memory as required e.g. for Golay correlation and rake reception.

R6. Also intra-vector operations (a.k.a. vector reductions) are important, e.g. for rake integration at small spreading factors (summing e.g. sets of 4 values).

R7. A useful, but rather CDMA-specific capability is the "generate the next *P* successive code chips" instruction for a variety of composite codes.

R8. As we aim at handheld devices, small silicon area (including data and program memories) is essential, as well as low power consumption.

R9. The processor must be conveniently programmable, supported by effective and efficient tools.

Note that R2-R4 deal with computational performance (*effective* number of operations per second), and that R5-R7 deal with functional capabilities.

4 The EVP vector processor

4.1 Architecture overview

As indicated in section 3, vector processing is one way to exploit the abundant and often regular parallelism encountered in many baseband algorithms. Using SIMD instructions (Single Instruction Multiple Data) arithmetic operations or load/store operations can be applied to *P* (e.g. *P*=16) data elements in parallel. Compared to other programmable architectures, SIMD execution results in low power consumption (R8), because the "overhead" of address calculations, data memory address decoding, instruction fetching/decoding, and control is shared by *P* operations. A similar reasoning holds for silicon area per MOPS.

Based on these findings, and the requirements of section 3.1 (R*n*), the EVP (Embedded Vector Processor) family has been defined. The EVP is a productized version of the CVP (Van Berkel, 2003), taking into account experiences gained in the OnDSP architecture (Kneip, 2002). Originally developed to support 3G and WLAN standards, the current architecture, depicted in Figure 23, proves to be highly versatile (as demonstrated for example in application in the domain of TV broadcast reception and TD-SCDMA cellular communication). The first instance (labeled the VD32040) is now in mass production. Its main architectural features are:

- Dominant Data size of 16 bits, as in conventional DSPs, with (limited) support for 8-bit and 32-bit data. A single SIMD vector comprises 16 data elements of 16-bit, supporting alternative views as 32 elements of 8 bits, and 8 elements of 32 bits. Accumulator registers have extension bits to support higher-resolution accumulation. Main data types are integer and fixed point, with support for complex numbers (2×8 or 2×16 bits) (R1).
- VLIW execution model which supports parallelism among multiple vector functional units (FUs), e.g. MAC, ALU. This VLIW parallelism comes in addition to vector parallelism. The maximum VLIW-parallelism available equals six vector operations (R3).
- VLIW instruction which may also specify several operations on scalar functional units (R4).
- Extensive support for address calculations (ACUs, e.g. post- increment, increment, modulo), which helps to keep many functional units busy.

Although the SIMD width is in principle scalable (R2), the value of *P* has been set to 16 (i.e. 16 data elements, totaling 256 bits in a vector) for the first products, the VD3204x series. The first instance of this line, the VD32040, is already in mass production in actual TD-SCDMA handsets (Cetto, 2007).

When fabricated in a 45 nm low power CMOS process, the VD32040 layout measures about 3 mm^2 including 4 K x 256 data memory and a 128-Kbyte program memory, can run at 320 MHz (worst case commercial), and dissipates less than 0.5 mW/MHz (including memories) for a typical application code mix (R8)

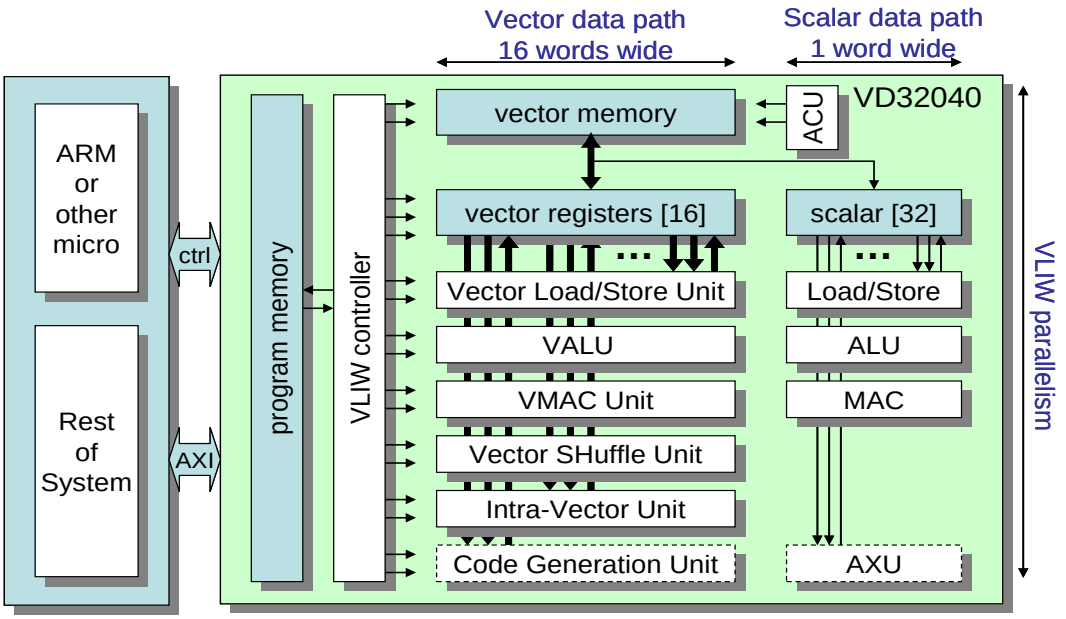

Figure 23: The EVP architecture

4.2 Vector operations

Diving into the VD32041 in some more detail, the VDCU, or Vector Data Computation Unit, is the central vector processing kernel of the core. It contains 6 vector-wide sub-units, each operating independently from the others, as controlled through the VLIW instruction set.

Operating on 256-bit vectors, the VDCU supports both SIMD and intra-vector-style computations with native support for integer and fixed point in various data sizes (8-, 16- and 32-bit). Complex data types have support at 16-bit fixed point precision. Typically, units have a single cycle initiation interval; meaning a new operation can be started every single cycle. Most operations also have a latency of a single cycle, so results can be used in subsequent operations in the next clock cycle.

In order to supply the functional units within the VDCU with sufficient data, the EVP is based on a register-oriented load/store architecture. The main register file is the 16-entry vector register file, capable of holding 16x16 = 256 words of 16-bit data. Using the vector load/store unit (VLSU), vector registers can be transferred to or from the data memory every clock cycle. Vector-aligned load and store operations, as well as unaligned vector loads, are supported. All other vector-processing units operate on data from this register file, and not directly on memory.

Two ‘classical SIMD’ functional units supply the basic SIMD-type support. The vector arithmetic logical unit (VALU) performs conventional SIMD-style arithmetic and logic operations, data-type conversions, barrel shift operations, and data move/copy operations. The unit also supports some domain-specific functionality, including operand selection via the use of mask registers, division, and conditional operations like conditional add/subtract. The vector multiply/accumulate (VMAC) unit performs multiply and MAC operations in 16-, 32-, or 40-bit

result formats, with optional rounding and saturation. The adder from the MAC is usable as such, to offload the VALU when dealing with addition-intense algorithms.

Next to the SIMD-style VALU, also an intra-vector ALU is available, the Intra-Vector Unit (IVU). This unit, not found in classical SIMD architectures, takes care of operations having interaction between elements within a vector. The IVU supports operations such as 'segmented adds', where groups of (2,4,8,...) elements within a vector are added into a new, smaller vector. This is useful e.g. in rake receivers with spreading factor smaller than the vector length. Other examples of IVU operations are maximum and minimum search within a vector.

The vector shuffle unit (VSHU), another non-SIMD type unit, shuffles elements within a vector by operating on a complete vector, with a granularity of 8, 16, or 32 bits (R5). The elements can be rotated, shifted, or copied within the vector or arbitrarily permuted, by element, according to freely programmable patterns. Insertion and extraction of words in the vector are also possible.

The code generation unit (CGU) is a domain-specific unit that tailors performance of the core, and is used to create scramble and channelization codes. It supports CDMA-code generation: in a single clock cycle 16 successive complex code chips are generated (R7). The unit can be configured for a wide variety of codes (UMTS, CDMA2000, GPS, etc).

All operations (including the vector store operation) can be performed in masked mode, where a bit-vector (located in the 8-entry mask register file) indicates active vector elements at 8-bit granularity. The vector mask ALU can be used to compute these masks, based on results from for example vector comparison operations.

4.3 Scalar and control operations

In order not to be limited too much by Amdahl's Law (Hennessy, 2003) (R4), a vector processor also should be able to perform scalar and control processing, in parallel to the vector operations. If not, these operations have to be done on the vector data path, blocking this for vector operations, or handed over to a neighboring control processor (like an ARM), limiting the throughput of the system due to inter-processor communication overhead. The target is to have the EVP cores operate autonomously, and communicate with a host processor only on large granularity, for example only on frame boundaries. For this purpose, strong scalar and control capabilities are included in the data path of this processor.

The SDCU, the Scalar Data Computation Unit, operates on 8-, 16-, and 32-bit integer/fixed-point data and behaves as a 16/32-bit RISC CPU data path. Like the VDCU, the architecture is register file based, having a central register file with 32 general-purpose registers of 16 bits each. Within the SDCU, the scalar load/store unit (SLSU), scalar ALU (SALU) and MAC unit (SMAC), and the predicate ALU (PALU) are available. Scalar results can be broadcast to the VDCU for use as vector input operands. Vice versa, a scalar can be selected from a vector register as input into the SDCU.

The SDCU is normally not used for address computations. Instead, for this purpose the dedicated ACU, the Address Computation Unit, provides parallel pointer arithmetic access for both scalar

and vector memory accesses on the shared data memory. As is common for DSP-like processors, it supports complex addressing modes like auto-increments and circular buffer addressing. For 'C' language support, also base register plus offset addressing modes are supported.

The PCU, Program Control Unit, takes care of program execution and flow control. To support conditional program flow, the PCU can perform conditional execution via predicates set in the 8-entry predicate file. Predicates can be computed, for example through compare operations, and can be combined in various ways through the predicate ALU in the scalar path. This way, C conditional expressions can be computed efficiently. Most instructions can be predicated, to eliminate branching altogether for short if-statements, removing pipeline penalties. Zero-overhead looping further reduces pipeline impacts, allowing loops to run at full speed, not needing additional cycles for loop counter computations and branching overhead.

The EVP uses a compressed VLIW encoding. This way, all functional units can be controlled in parallel, while taking opcode space only when actually active. The open (visible) pipeline results in simple, power-efficient control that requires no interlocking or hardware dependency analysis logic, by having the complexity of scheduling and dependencies moved from the hardware to the compiler.

4.4 System Interfaces and Memories

Designed for use as an IP block, the VD32041 core with surrounding subsystem (the shell containing e.g. memories and interfaces) is easy to integrate into a system-on-chip (SoC) design. It is a fully synthesizable, single clock edge design. The VD32040 supports an industry-standard ARM-originated AMBA AXI bus interface. The subsystem facilitates multi-core debugging via its CoreSight™ prepared debug and trace infrastructure, using AMBA advanced peripheral bus (APB) and AMBA advanced trace bus (ATB) interfaces.

The data and program memory are integrated into the EVP subsystem, in order to supply sufficient bandwidth. 256-bit wide internal busses connect both memories to the core. To the external system, they are visible through the AXI bus as regular 64-bit wide memory banks. This way, existing on-chip infrastructure like DMA control can be reused also for the EVP subsystem, both for data transfer and for program download. However, the EVP can also act as a bus master on the AXI bus, allowing autonomous data transfer.

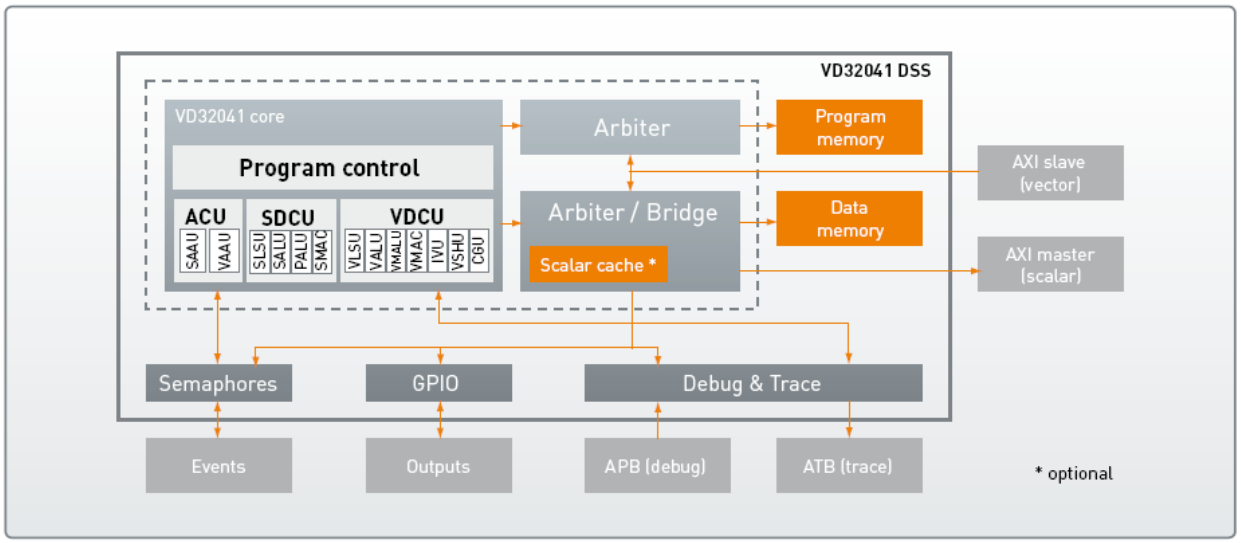

Figure 24: VD32040 Interfaces for SoC Integration

Next to these bus interfaces, there are also dedicated lines to signal between EVP and other subsystems, like other processors or peripherals. They can be configured to act as interrupt lines (source or destination) or as semaphore-like signals.

5 Programming the EVP

The challenge in the EVP is to exploit the two types of parallelism available: data level parallelism (SIMD and vector processing) and instruction level parallelism (implemented using VLIW). The instruction level parallelism can be taken care of by the compiler with relative ease. However, for the data parallelism (SIMD), this is not the case. In this area, even state-of-the-art compilers offer little real help, and so-called vectorizing compilers still only are able to extract a very small amount of data parallelism compared to human programmers. Typically, the compiler does not get much further than parallelizing sum-of-product type of constructs, while parallel architectures prefer different algorithms compared to a regular, sequential architecture, an intelligence that cannot yet be expected from a compiler (R9).

An example here is the selection of FFT algorithms. Classical sequential FFT implementations leave data in a ‘bit-reversed’ addressing order, which is next fixed by a data re-ordering phase. While the FFT kernel is easily vectorized, the (inherently sequential) data reordering phase will quickly become the bottleneck (Amdahl). Instead, selection of a so-called self-sorting FFT eliminates the post-processing phase altogether, while (on a suitable architecture) having zero penalty as the reordering operations can be scheduled in parallel with other vector operations (Smriti, 2006). In case of the EVP, the multipliers will remain nearly fully occupied, resulting in a 64-point complex FFT kernel taking only 50 cycles.

5.1 EVP-C: C with a plus

Although compilers do not yet bring much in the area of vectorization, the use of high level languages like C give a clear increase in programmer productivity. The use of high level data

types and structures eases programming, while tedious tasks as register allocation and VLIW scheduling are taken care of by tools.

Programs for the EVP are written in EVP-C, a superset of ANSI-C. EVP-C contains a number of extensions for vector processing. The extensions include not only vector data types, but also function 'intrinsics' to describe vector operations using a function call syntax, as many vector operations (as element permutations) do not have an equivalent C operator. Using these extensions, the programmer can express the parallelism (often difficult in plain C), and the compiler is thereby capable of using the full vector parallelism of the EVP. In Figure 25 an example code fragment in EVP-C is displayed.

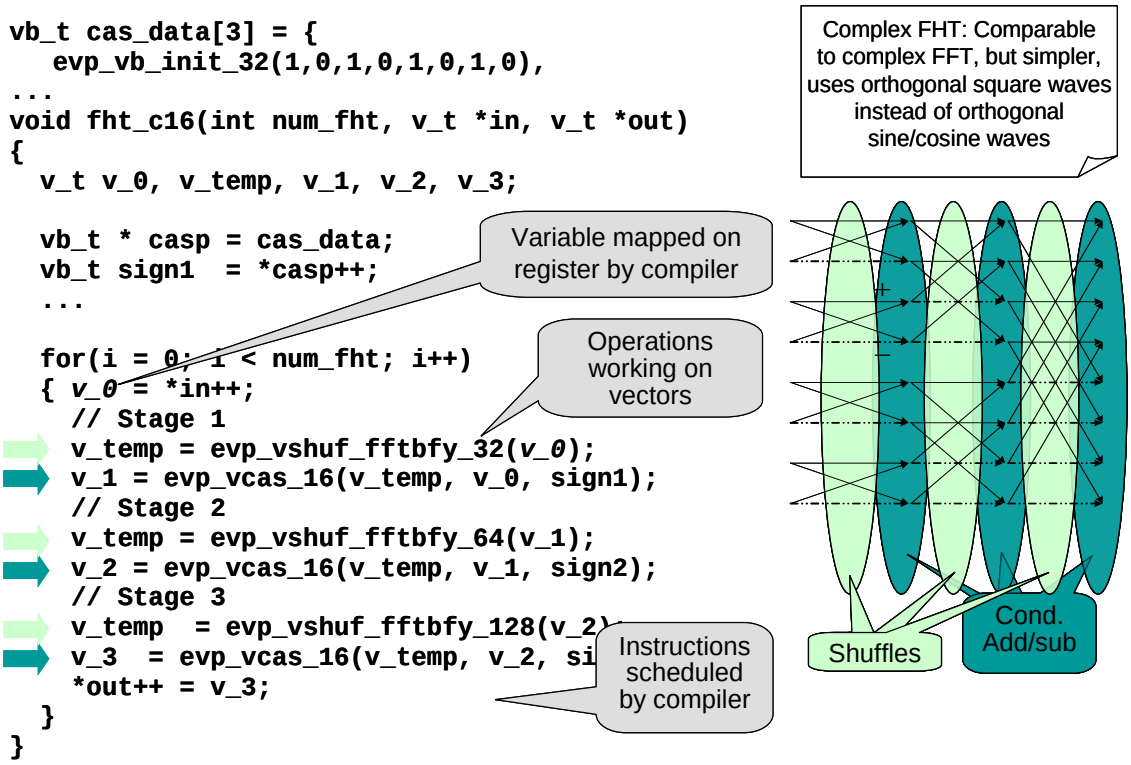

Figure 25: An example of EVP-C: the Fast Hadamard Transform as parallel program

In this example, two 'intrinsics' are used (as no equivalent C operator symbol is available): the vector shuffle (using a predefined pattern, hence named `evp_vshuf_fftbfy`) and the conditional add/subtract (`evp_vcas_16`), taking next to the two input operands a bit vector indicating which elements are to be added (solid lines), and which element pairs are to be subtracted (dashed lines). The complete data flow as depicted in the figure is now mapped on the inner loop in the C code, in 8 C lines. Due to the parallelism (and the capabilities of the compiler, like software pipelining), this loop is translated into a 4-instruction inner loop; giving a throughput of one complete FHT every 4 cycles.

5.2 EVP firmware development flow

To aid in developing the parallel code, without moving the programmer directly into the full complexity of a new tool chain, we enable a staged development approach (see Figure 26). After getting familiar with a new algorithm (stage 0, outside the scope of our tools) we offer what we call 'host emulation' (stage 1). In this case, the user works in his normal C++ development environment (as EVP-C is a full subset of C++). Using supplied libraries implementing in a bit-true fashion the intrinsics of the EVP, he can write his algorithms in a familiar environment, for example to check the fixed point behavior and parallelism. Also debugging and functional verification can be done in this stage. Only in a second optimizing stage, he can move to the real EVP tool chain, to verify and optimize cycle-accurate execution.

The EVP tool flow consists of standard tools like a linker, a high-speed instruction set simulator model (including wrappers for MatLab and System-C), a profiler, and an integrated debugger. The tools are integrated into an Eclipse-based IDE (Integrated Development Environment), although all tools can also be used in command-line mode, for batch execution.

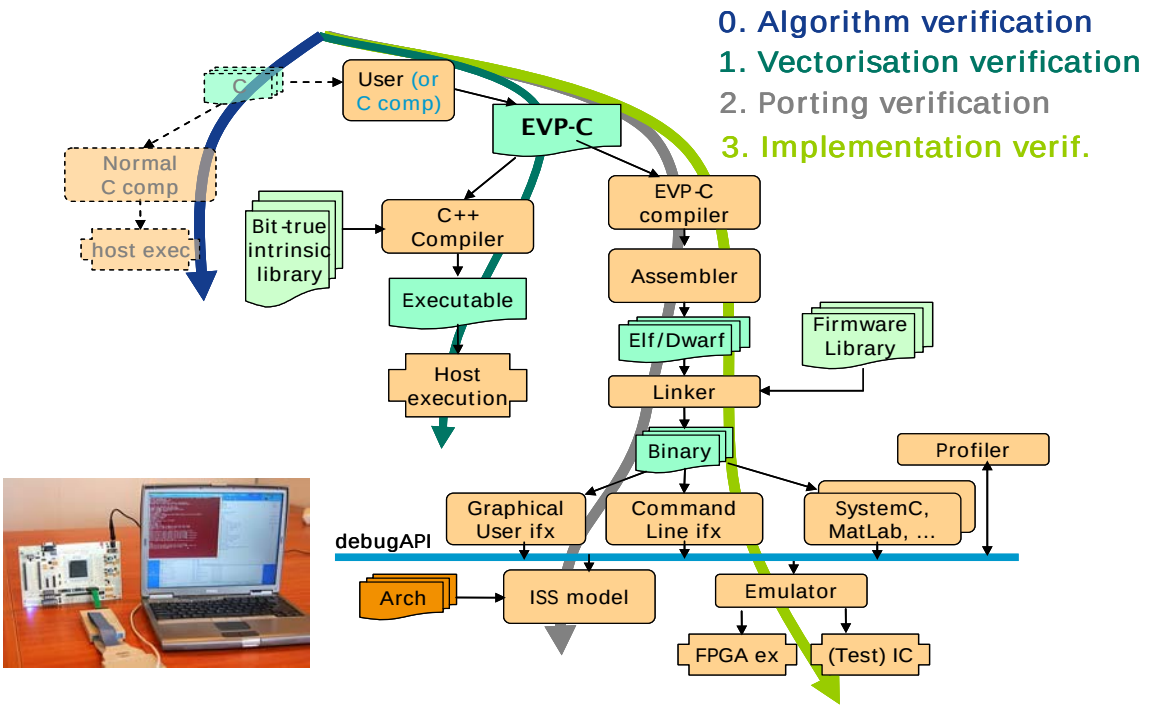

Figure 26: The EVP firmware development flow

6 SDR results

In the preceding sections we presented a vector-processor architecture and toolset for SDR, using several baseband algorithms from the UMTS and WLAN domain as drivers. In addition to its efficiency for isolated algorithms, the resulting architecture (Figure 22) needs to be assessed in an overall system context.

6.1 UMTS-FDD

Today's GSM handsets deploy programmable DSPs for all baseband signal processing, with all the associated flexibility benefits. For 3G standards, such as UMTS, this is not entirely practical, at least for the time being. In (Van Berkel, 2004) we can see that Turbo decoding alone requires about 55 clock cycles per symbol on the EVP. For UMTS 3GPP R'99 this results in an acceptable 35 MIPS for a 640 kbps channel. For 3GPP release 5, however, a 14 Mbps data rate would result in an EVP load in excess of 700 MHz, and power consumption close to 1 Watt in 90 nm CMOS. Accordingly, we chose the architecture of Figure 22. This hardware-software partitioning is markedly different from (Glossner, 2003), where all channel (de-) coding is also mapped on DSP software.

1. 3G+ standards show a much more dynamic computational load than 2G standards, both due to the nature of the employed algorithms and the large number of different use cases. This can be illustrated for four UMTS scenarios (3GPP, 2003): UMTS R'99 idle mode, only multi-cell synchronization.
2. UMTS R'99 connected mode with flat fading conditions (1 rake finger only), 3 dedicated channels (DCH), and neighbor-cell broadcast channel (BCH) monitoring.
3. UMTS R'99 connected mode in a scattering environment with multiple paths (six rake fingers) with the same transport channel configuration as before.
4. UMTS R'99 connected mode in a scattering environment with multiple paths (six rake fingers) with the same transport channel configuration plus an HSDPA (High Speed Downlink Packet Access) link (3GPP R5) with 15 downlink channelization codes.

The EVP load numbers for these four scenarios, based on simulation of kernels, are summarized in Table 8. Note the variation in load distributions! More advanced receiver algorithms such as interference cancellation, chip-rate equalization, or joint detection will be required in the future, both to increase the system capacity and to improve the reception quality. This is from our point of view one of the most compelling justifications for SDR.

Table 8: EVP loads for the modem stage; 4 scenarios

	EVP load [MIPS]			
UMTS task	1	2	3	4
PSCH search (Golay)	18	18	18	18
CPICH search	98	17	17	17
CPICH despreading		4	22	33
CPICH symbol rate		1	1	2
DCH despreading		3	16	16
DCH symbol rate		1	6	6
HS-SCCH despreading				15
HS-SCCH symbol rate				1
HS-DSCH despreading				12
HS-DSCH symbol rate				22
Overall peak load	*116*	*44*	*80*	*142*
Overall average load	*4*	*28*	*64*	*126*

6.2 Rake receiver for UMTS-FDD

On the EVP VD32040 the inner loop of the rake keeps most functional units busy. A single EVP VLIW instruction specifies:

- load and align a sample vector of P=16 data chips,
- auto increment by P=16 of the pointer to these chips,
- generate a vector of the next P=16 code chips,
- correlate a vector of data and a vector of code chips,
- intra-add result to one or few symbols ($S \leq P$) or to one partial symbol ($S > P$), depending on the spreading factor

For the UMTS-FDD chip rate of 3.84 MHz this implies a load of about 0.3 MHz per rake finger for the chip-rate processing, including loop pre-ambles and post-ambles for blocks of 512 chips. Including some symbol-rate processing this will increase to 0.4-0.5 MHz, depending on the spreading factor.

6.3 Golay correlator for UMTS-FDD

On the EVP VD32040 as many as ½P= 8 complex additions/subtractions (2 × 16 bit) can be computed in parallel. Automatic scheduling of the EVP-C version of the Golay correlator on the VD32040 requires 22 cycles for ½P symbols. Manual optimization of the memory accesses and schedule reduces this to 16 cycles for ½P symbols. Note that the 16 adders are busy for 13/16 of the time (van Berkel, 2004). Assuming correlation on two sample phases of 4 MHz each, the VD32040 load becomes $2 \times 4.10^6 \times 16\, cycles/(½P = 8)$, or approximately 16 MHz.

7 Conclusion

The modem stage of an SDR requires software flexibility to cope with the multitude of wireless standards, their evolution, and with algorithmic improvement (including bug fixes and in-filed upgrades) without the need to re-spin an IC. Vector parallelism in combination with VLIW can offer the computation power required for this.

The EVP, with its powerful functional units (Shuffle, Intra-Vector, Code Generation) outperforms conventional DSPs by an order of magnitude or more, in a power-efficient way. This is proven in reality, having a TD-SCDMA based on EVP in mass production. Accordingly, the EVP can be a key component of an SDR, enabling the flexibility of a software programmable architecture, optimizing the balance between reception quality and power consumption, while saving silicon area by both intra-standard and inter-standard re-use.

8 References

3GPP TS 25.211 (2003-09), “Physical channels and mapping of transport channels onto physical channels (FDD) – (Release 5)”, v. 5.5.0

Becher, Reinhard et al (2001), “Broad-Band Wireless Access and Future Communication Networks”, Proc. of the IEEE, Vol. 89, No. 1, pp. 58-75, Jan 2001.

Cetto, Marc et al (2007), “World’s First TD-SCDMA 3G HSDPA / GSM Multi-Mode Mobile Phone”; http://www.3g.co.uk/PR/Nov2007/5407a.htm

Glossner, John et al (2003), “A Software-Defined Communications Baseband Design”, IEEE Communications Magazine, pp. 120-128, Jan. 2003.

Gwennap (1998), “G4 Is First PowerPC With Altivec”, in Microprocessor Report, Nov. 16, 1998, pp 17-19

Hennessy, D. Patterson (2003), Computer Architecture, 3rd edition, Morgan Kaufmann Publishers.

Kneip, Johannes et al (2002), “Single Chip Programmable Base-band ASSP for 5GHz Wireless LAN Applications”, in IEICE Trans. Electron., Vol.E85-C, No.2, pp 359-367, 2002

Kokozinski, Rainer et al (2002), “The Evolution of Hardware Platforms for Mobile `Software Defined Radio’, Int. Symp. on PIMRC, Vol. 5, pp. 2389-2393, 2002.

Lapsley, Phil et al (1994-1996), “DSP Processor Fundamentals”, Berkeley Design Technology, Inc. 1994-1996.

Moerman, Kees (2008), “How will Software (Defined) Radio Enable Next Generation LTE Terminals?”, proceedings Wireless Congress: Systems & Applications, November 2008.

Smriti, Mahima et al, “A generic self-sorting FFT implementation on the VD32040 vector processor”, proceedings GSPx 2006

Van Berkel, Kees et al (2003), “CVP: A Programmable Co Vector Processor for 3G Mobile Base-band Processing”, Proc. World Wireless Congress, May 2003.

Van Berkel, Kees et al (2004), “Vector Processing as an Enabler for Software-Defined Radio in Handheld Devices”, proceedings Eurasip 2004

CHAPTER 5 – SDR IN WIRELESS INFRASTRUCTURE: HOW XILINX FPGAS ARE ENABLING SDR

1 Summary

Success in wireless infrastructure demands a common platform that supports the latest technologies, enables deployment across all geographies, and scales to meet the demands of network operators, while also minimizing capital outlay and operational costs. Xilinx has a long history of understanding and solving the challenges facing wireless basestation designers. We have developed a comprehensive range of cost effective processing devices and customizable building blocks targeted at a wide range of air interface standards such as 3GPP LTE, TD-SCDMA, WiMAX, W-CDMA/HSPA, CDMA2000 and Multi-Carrier GSM.

Xilinx is particularly focused on delivering IP to support 3GPP LTE development, and has released a number of optimized IP solutions targeted at this standard. For example, our LTE Digital Front End for remote radio heads and radio cards was the first comprehensive LTE solution available to the market, and has been widely downloaded and integrated by customers looking to gain an advantage in their system development.

2 Semiconductor Solutions for Wireless Infrastructure

Xilinx delivers high-performance, cost-effective solutions for wireless networking equipment in areas such as RF digital front-end (DFE) signal processing and baseband processing, as well as advanced interfacing, connectivity, and bridging applications. Xilinx's fit in the basestation is illustrated below in Figure 27. Xilinx also provides 3GPP-LTE optimized radio and baseband IP cores.

A notable trend is a move away from ASIC technology to flexible programmable technology in basestation design. There are several key reasons for this shift:

- ASIC development costs are high - >$20M at 65nm - and development is long and costly in manpower. A small change to the standard can necessitate expensive ASIC re-spins.
- Replacing an ASIC design in the field carries significant costs ($1,000 per site truck-roll to upgrade a network of 50,000 base stations costs upward of $50 million).
- The shift towards SDR (software-defined radio)/multimode basestations places a much higher premium on reconfigurable and reprogrammable architectures.

This culmination of these trends in the wireless industry and the semiconductor industry are converging to support the widespread deployment of SDR technology in wireless infrastructure.

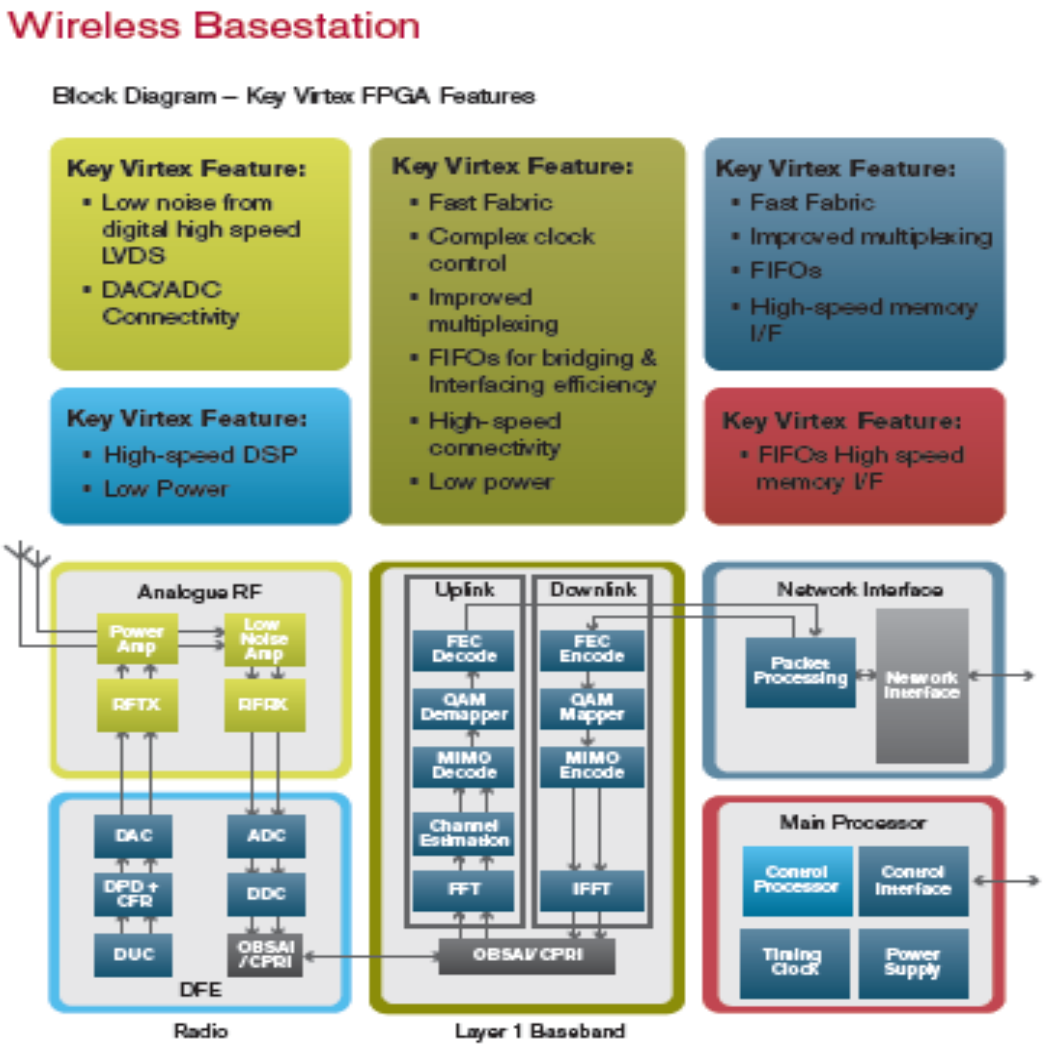

Figure 27: Xilinx devices fit into most areas of the basestation due to the features color-coded above.

3 LTE eNodeB Targeted Design Platform (TDP)

Xilinx has a large portfolio of LTE IP components today, including a complete LTE digital front end, as well as the uplink and downlink for baseband processing. These components also scale from femtocells to macrocells. The Xilinx LTE eNodeB Targeted Design Platform, illustrated in Figure 28, extends this portfolio to a board-level basestation reference design which can support both FDD and TDD variants of LTE. To verify the quality and compliancy of the platform and IP, it will be tested against industry standard test mobile and UE equipment. Available in a modular fashion, system designers can choose only the hardware and software components they require to speed up the development process.

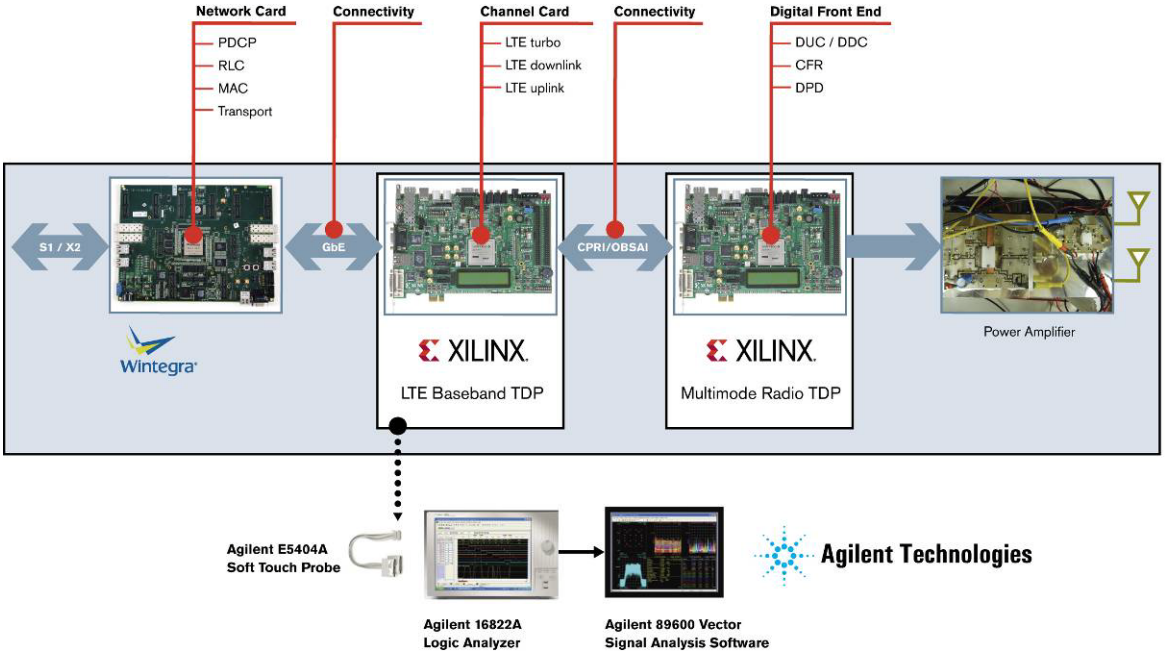

Figure 28: Xilinx's LTE eNodeB Targeted Design Platform is essentially a basestation reference design.

The LTE eNodeB Targeted Design Platform (TDP) comprises (from left to right) of a Layer 2 platform from Wintegra, an LTE Baseband TDP by Xilinx, a multimode radio TDP by Xilinx, and a power amplifier. Agilent test equipment and test suites are used to verify the individual TDPs, as well as the overall system.

It is important to note that while this TDP is focused on LTE from the IP and testing perspective, it is inherently an SDR platform that can support a number of waveforms/air interface protocols.

4 Radio Processing in FPGAs

Radio design has become a primary focus in all wireless systems as OEMs strive to produce higher performance channels at lower cost. Soaring energy costs and power losses—such as the 50% loss of power in the cable from amplifier to antenna—are driving the demand for greater efficiencies and a new wave of innovation, such as advanced remote radio heads. This has necessitated a move to more digital pre-processing of signals and to the development of algorithms for digital up (DUC) and down conversion (DDC), crest factor reduction (CFR) and digital pre-distortion (DPD). All are very high-performance areas of signal processing in which Xilinx FPGAs excel.

The key radio design challenges, illustrated in Figure 29, include:

- Maximizing power amplifier efficiency and improving overall performance,
- Reducing costs while providing coverage support for multiple technologies, traffic types, geographies, and climate conditions,

- Increasing basestation range and number of users without sacrificing call quality or increasing costs,
- Delivering critical system reliability to support five 9s reliability (i.e., 99.999% availability = less than 5 minutes of downtime per year).

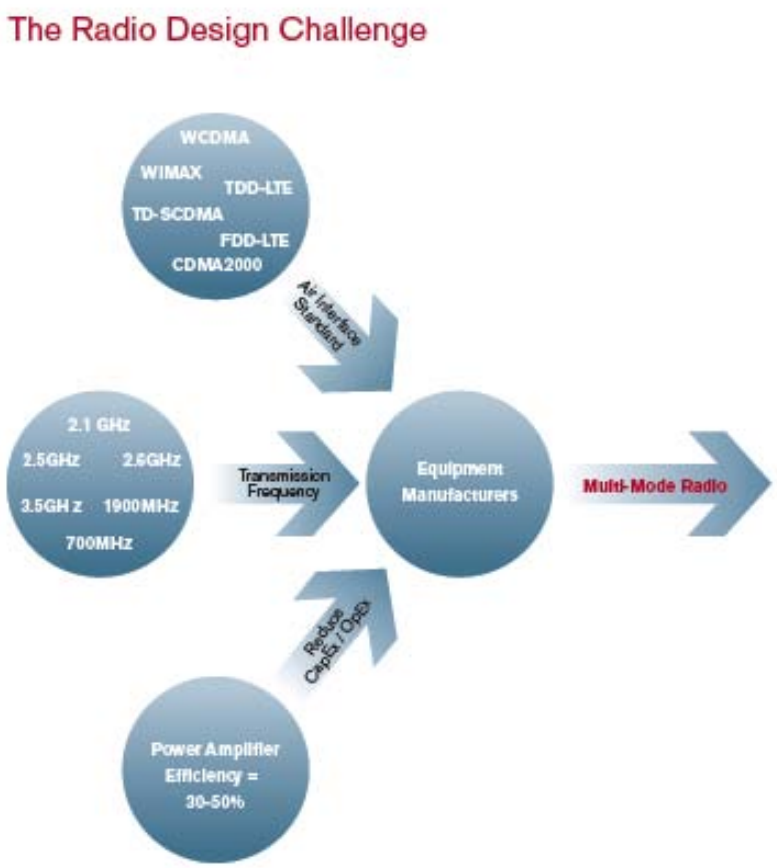

Figure 29: The radio design challenges necessitating the move to multi-mode or SDR radios

FPGAs are ideally suited to address these design challenges due to their high performance, flexibility, integration and scalability. Figure 30 shows an example of how FPGAs can provide the most cost effective and low power radio solution. This example is for a single sector HSPA radio card supporting 3 carriers at 15MHz bandwidth. The initial design utilized six ASSP devices, and consumed upwards of 8W of power at a list price of $219 (1K unit list pricing). These six ASSPs were required for the manufacturer to have sufficient diversity to make their devices applicable to more than just one market. More integrated ASSPs would have brought lower volumes, making development too risky. Retargeting the functions into a single FPGA led to dramatic power reduction, as well as unit cost savings of more than 30%. The added flexibility of Xilinx technology leads to even more potential cost savings over the lifetime of the product in the field, as shown in the table.

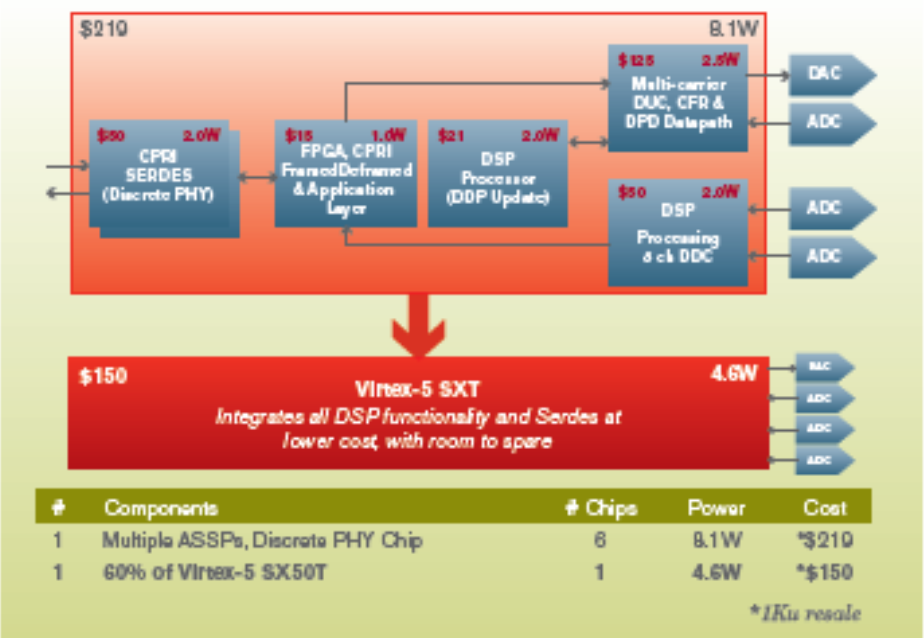

Figure 30: Case study of ASSP vs FPGA radio implementations

Xilinx today offers both reconfigurable silicon, as well as air interface specific IP, to provide radio solutions for LTE (both TDD and FDD), WiMAX, W-CDMA, TD-SCDMA and CDMA2000.

RF (Radio Card)		
Resource	**Reference Design**	**IP**
LTE Digital Front End		X
Digital Pre-Distortion		X
PC-CFR		X
WCDMA/HSPA Digital Front End		X
WiMAX Digital Front End		X
TD-SCDMA Digital Front End		X

Figure 31: Representative radio solutions and IP available from Xilinx

5 Baseband Processing in FPGAs

The baseband processing signal chain presents both the greatest challenge and one of the best opportunities for innovation in the BTS. It has also become a key area for product differentiation

between manufacturers. Competition has recently intensified with the realization that many of the architectures used for earlier 2G and 3G systems simply will not scale to meet the performance and latency requirements demanded by 3GPP-LTE. This means that the processing chain has to contend with far more throughput than ever before — in much less time. Completing the set of challenges faced by systems architects is the need to develop a system that can meet the aggressive CAPEX and OPEX reduction targets imposed by operators. These place significant pressures on the baseband processing system design.

Some of the key baseband processing design challenges include:

- Support for increasing data rates while reducing latency,
- Usage of advanced algorithms to reduce OPEX and CAPEX,
- Being flexible enough to support multiple air interface protocols, and the evolutions to those standards.

System partitioning worked adequately in wireless standards available before 3GPP-LTE. However, the low-latency performance requirements of 3GPP LTE have put a strain on data flow between the DSP and FPGA and created a bottleneck, as shown in Figure 32. The latest Xilinx LTE Channel Uplink and Downlink LogiCORE™ helps alleviate this bottleneck by integrating more of the baseband processing within the FPGA, bringing additional benefits in reduced cost and power. New functions such as MIMO processing are adding even greater performance requirements to baseband system design. Xilinx also provides an LTE MIMO Encoder.

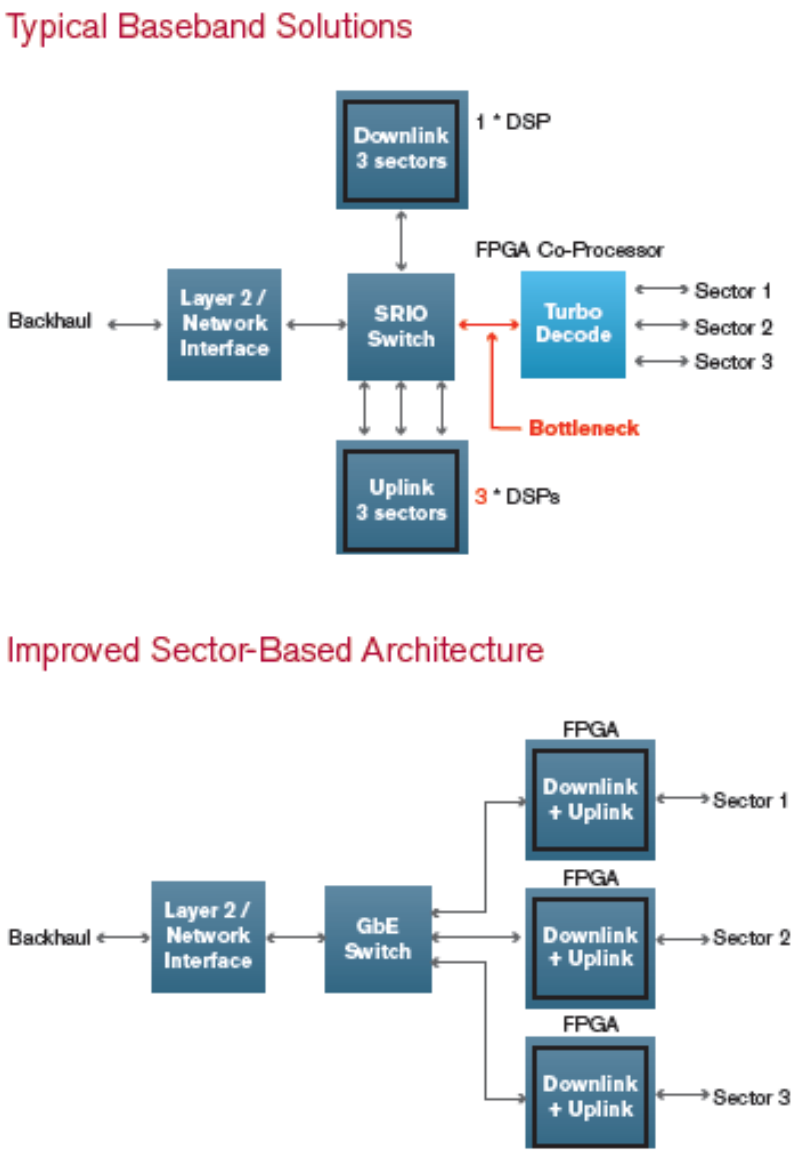

Figure 32: Unlike previous functional architectures, the sector-based architecture can support the data flow of LTE and other high data rate air interfaces

As illustrated on the next page in Figure 33, a complete LTE downlink and uplink for a single sector can be implemented in a single, medium-sized FPGA, such as a Virtex-5 SX50T or SX95T.

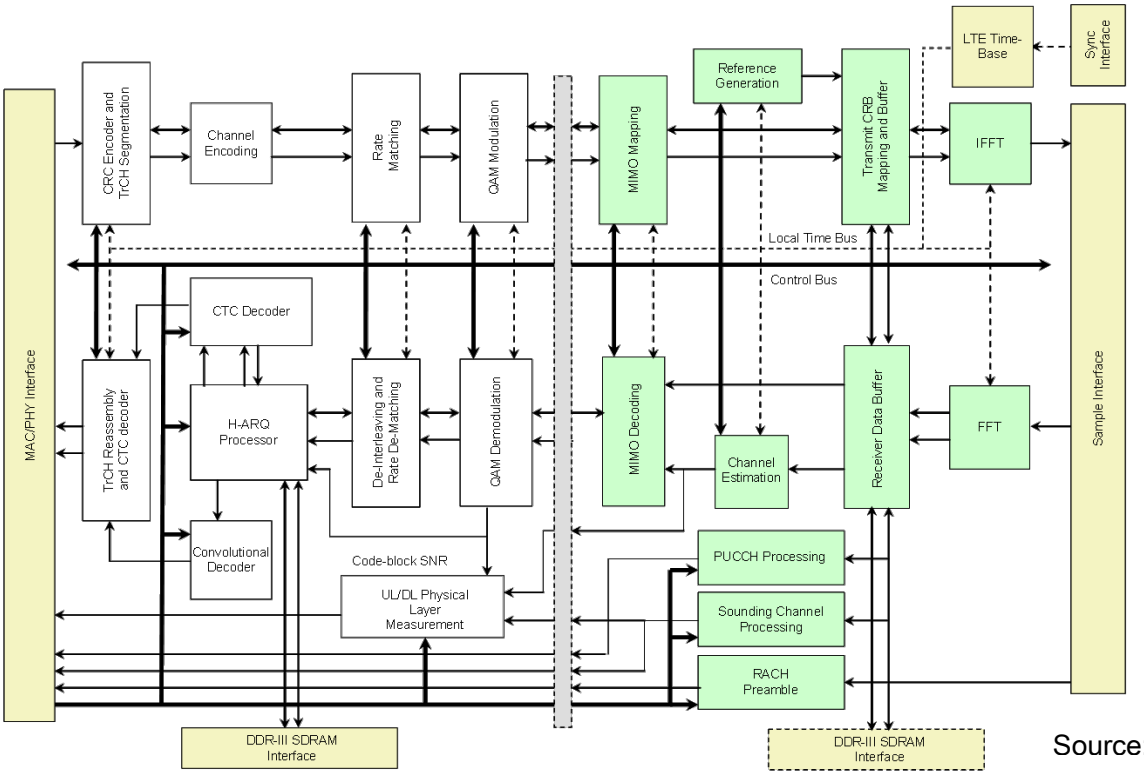

Figure 33: Single sector LTE implementation in an FPGA

Xilinx offers LTE IP, including downlink and uplink reference designs that can scale from femtocell to macrocell. Key building blocks such as the turbo coder, FFT and DFT, which make up the downlink and uplink, are also available individually. Xilinx also has Forward Error Correction (FEC) IP for WiMAX, W-CDMA and CDMA2000.

Baseband Processing (Channel Card)		
Resource	Reference Design	IP
LTE Downlink Reference Design	X	
LTE Uplink Reference Design	X	
LTE Baseband System	X	
LTE MIMO Encoder LogiCORE		X
LTE Channel Encoder LogiCORE		X
LTE Channel Decoder LogiCORE		X
LTE Turbo Encoder LogiCORE		X
LTE Turbo Decoder LogiCORE		X
Viterbi/Convolutional LogiCORE		X
Reed-Solomon LogiCORE		X
FFT LogiCORE		X
DFT LogiCORE		X
WCDMA/HSPA RACH LogiCORE		X
WCDMA/HSPA Searcher LogiCORE		X
WiMAX 802.16e CTC		X

Figure 34: Representative baseband solutions and IP available from Xilinx

6 Wireless Connectivity

The primary interface connection between the radio and baseband card has always been proprietary and, as a result, comparatively expensive. However, this interface is a natural partition that occurs in all BTS architectures. Defining a standard interface opens up the BTS architecture to market forces that reduce costs through increased competition. Currently, two focused initiatives — Open Base Station Architecture Initiative (OBSAI), and Common Public Radio Interface (CPRI) — exist solely to address this need. At the device level, the introduction of DSPs that used SRIO as a connectivity standard for chip-to-chip communication has necessitated other devices to support the standard. Performing the chip-to-chip functions may also require some SRIO switching capability.

These issues have led to key connectivity challenges:

- Reducing overall development costs through standardization,
- Developing a solution compatible with standards that scale to cover speeds ranging from 614Mbps up to 3Gbps (rising to 6Gbps in the near future),
- Devices that meet performance demands with minimum power and cost,
- Capability to interface to the latest DSPs.

Xilinx provides wireless connectivity IP portfolio including support for the latest specifications and line rates of OBSAI, CPRI and SRIO.

GLOSSARY

ADRES: Architecture for Dynamcially a vector processing template used in IMEC's SDR

ARM: Advanced RISC Machines - UK-based company making GPP's.

Baseband: describes signals and systems whose range of frequencies is measured from zero to a maximum bandwidth or highest signal frequency

C: A programming language

CDMA: Code Division Multiple Access

CGA: Course Grain Array used in IMEC's SDR

CMOS: Complementary metal–oxide–semiconductor - a technology for making integrated circuits.

DFE: Digital Front End

DMB: Digital Multimedia Broadcasting - a digital radio transmission technology developed by South Korea as part of the national IT project for sending multimedia such as TV, radio and datacasting to mobile devices such as mobile phones.

DSP: Digital Signal Processor

DVB: Digital Video Broadcasting - a suite of internationally accepted open standards for digital television

Etherstack: a wireless communications company specializing in a variety of standards to provide protocol stacks, waveforms and IP core networks for civil and military communications

EVP: Embedded Vector Processor

FEC: Forward Error Correction (e.g. Viterbi or turbo decoding)

Fixed Point: Non-associative datatype commonly used in DSPs

FPGA: field-programmable gate array - a semiconductor device that can be configured by the customer or designer after manufacturing

GPP: General Purpose Processor

GPRS: General Packet Radio Service

GSM: Global System for Mobile Communications

HSPA: High Speed Packet Access

IMEC: The largest independent research lab in Europe

ISDB-T: Integrated Services Digital Broadcasting - a Japanese standard for digital television (DTV) and digital radio used by the country's radio and television stations.

LMR: Land Mobile Radio

LP CMOS: Low Power CMOS

LTE: Long Term Evolution

Modem: (from modulator-demodulator) - a device that modulates an analog carrier signal to encode digital information, and also demodulates such a carrier signal to decode the transmitted information.

MIMO: multiple-input and multiple-output - the use of multiple antennas at both the transmitter and receiver to improve communication performance.

MPSoC: MultiProcessor System-On-Chip

PMP: Portability, Maintainability, and Performance

RISC: Reduced Instruction Set Computer

Sandbridge Technologies: A US-based company developing SDR baseband processors

SB3500: A 4-way Multithreaded 16-wide vector processor MPSoC from Sandbridge Technologies.

SCA: Software Communications Architecture

SDR: Software Defined Radio - a radio communication system where components that have typically been implemented in hardware (e.g. mixers, filters, amplifiers, modulators/demodulators, detectors. etc.) are instead implemented using software on a personal computer or other embedded computing devices.

SIMD: Single Instruction Multiple Data

ST-Ericsson: A 50-50 joint venture of ST Microelectronics and Ericsson.

TD-SCDMA: Time Division Synchronous Code Division Multiple Access

UMTS: Universal Mobile Telecommunications System. GSM, GPRS, WCDMA, HSPA and LTE are specifications under UMTS.

VLIW: Very Long Instruction Word processor

WCDMA: Wideband Code Division Multiple Access

WiFi: a trademark of the Wi-Fi Alliance for certified products based on the IEEE 802.11 standards

WMAN: Wireless Metro Area Network

WiMAX: Worldwide Interoperability for Microwave Access

WLAN: a Wireless Local Area Network that links two or more computers or devices using spread-spectrum or OFDM modulation technology

Xilinx: The worldwide leader in programmable logic solutions.