Signal Processing with High Complexity: Prototyping and Industrial Design

Guest Editors: Markus Rupp, Thomas Kaiser, Jean-Francois Nezan, and Gerhard Schmidt
Signal Processing with High Complexity: Prototyping and Industrial Design
Signal Processing with High Complexity: Prototyping and Industrial Design

Guest Editors: Markus Rupp, Thomas Kaiser, Jean-Francois Nezan, and Gerhard Schmidt
Editor-in-Chief
Zoran Salcic, University of Auckland, New Zealand

Associate Editors
Sandro Bartolini, Italy
Neil Bergmann, Australia
Shuvra Bhattacharyya, USA
Ed Brinksma, The Netherlands
Paul Caspi, France
Liang-Gee Chen, Taiwan
Dietmar Dietrich, Austria
Stephen A. Edwards, USA
Alain Girault, France
Rajesh K Gupta, USA
Susumu Horiguchi, Japan

Thomas Kaiser, Germany
Bart Kienhuis, The Netherlands
Chong-Min Kyung, Korea
Miriam Leeser, USA
John McAllister, UK
Koji Nakano, Japan
Antonio Nunez, Spain
Sri Parameswaran, Australia
Zebo Peng, Sweden
Marco Platzner, Germany
Marc Pouzet, France

S. Ramesh, India
Partha Roop, New Zealand
Markus Rupp, Austria
Asim Smailagic, USA
Leonel Sousa, Portugal
Jarmo Henrik Takala, Finland
Jean-Pierre Talpin, France
Juergen Teich, Germany
Dongsheng Wang, China
Contents

Signal Processing with High Complexity: Prototyping and Industrial Design, Markus Rupp, Thomas Kaiser, Jean-Francois Nezan, and Gerhard Schmidt
Volume 2006 (2006), Article ID 90363, 2 pages

Rapid Industrial Prototyping and SoC Design of 3G/4G Wireless Systems Using an HLS Methodology, Yuanbin Guo, Dennis McCain, Joseph R. Cavallaro, and Andres Takach
Volume 2006 (2006), Article ID 14952, 25 pages

Generation of Embedded Hardware/Software from SystemC, Salim Ouadjaout and Dominique Houzet
Volume 2006 (2006), Article ID 18526, 11 pages

Efficient Design Methods for Embedded Communication Systems, M. Holzer, B. Knerr, P. Belanović, and M. Rupp
Volume 2006 (2006), Article ID 64913, 18 pages

Fixed-Point Configurable Hardware Components, Romuald Rocher, Daniel Menard, Nicolas Herve, and Olivier Senteiys
Volume 2006 (2006), Article ID 23197, 13 pages
Editorial

Signal Processing with High Complexity: Prototyping and Industrial Design

Markus Rupp,1 Thomas Kaiser,2 Jean-Francois Nezan,3 and Gerhard Schmidt4

1 Institute for Communication and RF Engineering, Vienna University of Technology, Gusshausstrasse 25/389, 1040 Vienna, Austria
2 Institut für Kommunikationstechnik, Leibniz Universität Hannover, Appelstrasse 9a, 30167 Hannover, Germany
3 IETR/Image Group Lab, France
4 Harman/Becker Automotive Systems, 89077 Ulm, Germany

Received 10 July 2006; Accepted 11 July 2006

Copyright © 2006 Markus Rupp et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Some modern applications require an extraordinary large amount of complexity in signal processing algorithms. For example, the 3rd generation of wireless cellular systems is expected to require 1000 times more complexity when compared to its 2nd generation predecessors, and future 3GPP standards will aim for even more number-crunching applications. Video and multimedia applications do not only drive the complexity to new peaks in wired and wireless systems but also in personal and home devices. Also in acoustics, modern hearing aids, or algorithms for dereverberation of rooms, blind source separation and multichannel echo cancellation are complexity hungry. At the same time the anticipated products also put on additional constraints like size and power consumption when mobile and thus battery powered. Furthermore, due to new developments in electro-acoustic transducer design, it is possible to design very small and effective loudspeakers. Unfortunately, the linearity assumption does not hold any more for this kind of loudspeakers, leading to computationally demanding nonlinear cancellation and equalization algorithms.

Since standard design techniques would either consume too much time or not result in solutions satisfying all constraints, more efficient development techniques are required to speed up this crucial phase. In general such developments are rather expensive due to the required extraordinary high complexity. Thus, de-risking of a future product based on rapid prototyping is often an alternative approach. However, since prototyping would delay the development, it often makes only sense when it is well embedded in the product design process. Rapid prototyping has thus evolved by applying new design techniques more suitable to support a quick time to market requirement.

This special issue focuses on new development methods for applications with high complexity in signal processing and on showing the improved design obtained by such methods. Examples of such methods are virtual prototyping, HW/SW partitioning, automatic design flows, float to fix conversions, and automatic testing and verification.

We received seven submissions of which only four were accepted.

In Rapid industrial prototyping and SoC design of 3G/4G wireless systems using an HLS methodology the authors Yuanbin Guo et al. present their industrial rapid prototyping experiences on 3G/4G wireless systems using advanced signal processing algorithms in MIMO-CDMA and MIMO-OFDM systems. Advanced receiver algorithms suitable for implementation, HW/SW partitioning, automatic design flows, float to fix conversions, and automatic testing and verification are presented. This design experience demonstrates that it is possible to enable an extensive architectural analysis in a short time frame using HLS methodology by abstracting the hardware design iterations to an algorithmic C/C++ fixed-point design, which in turn significantly shortens the time to market for wireless systems.

In Generation of embedded hardware/software from systemC the authors Salim Ouadjiaout and Dominique Houzet present a design flow to reduce the SoC design cost. This design flow unifies hardware and software using a single high level language and thus decreases the manual errors by rewriting design code. It integrates hardware/software (HW/SW) generation tools and an automatic interface synthesis through a custom library of adapters. The approach is validated on a hardware producer/consumer case study and on the design of a given software-radio communication application.
In **Efficient design methods for embedded communication systems** the authors Martin Holzer et al. analyze a complete design process to exhibit inefficiencies. The lack of an integrated design methodology is argued. High level characterisation, virtual prototyping, automated hardware/software partitioning, and floating-point to fixed-point data conversion are bottlenecks to solve in such a methodology. For each point, authors present and compare several tools and algorithms leading to an efficient fast prototyping framework. Examples are given in the field of high-complexity communication systems but can be extended to other complex application fields.

In **Fixed-point configurable hardware components** the authors Romuald Rocher et al. propose a flexible scheme for fixed-point optimization in order to better exploit advances in VLSI technology. After determining the dynamic range and the binary point, a data word-length optimization follows by introducing a suitable user-defined cost function. This central cost function, which, for example, depends on chip area and/or energy consumption, is to be minimized under the constraint of a pre-defined thresholded signal-to-quantization noise ratio (SQNR). Through use of analytical models the design time can be significantly reduced. A 128-tap LMS filter design exemplarily explores the fixed-point search space and demonstrates the benefits of the proposed scheme.

**Markus Rupp**

**Thomas Kaiser**

**Jean-Francois Nezan**

**Gerhard Schmidt**

**Markus Rupp** received his Dipl.-Ing. degree in 1988 at the University of Saarbruecken, Germany, and his Dr.-Ing. degree in 1993 at the Technische Universität Darmstadt, Germany. He is presently a Full Professor for digital signal processing in mobile communications at the Technische Universität of Vienna. He is Associate Editor of IEEE Transactions on Signal Processing, EURASIP Journal of Applied Signal Processing, EURASIP Journal on Embedded Systems and is elected AdCom Member of EURASIP. He authored and co-authored more than 200 papers and patents on adaptive filtering, wireless communications, and rapid prototyping.

**Thomas Kaiser** received the Ph.D. degree in 1995 with distinction and the German habilitation degree in 2000, both in electrical engineering from Gerhard-Mercator-University Duisburg. In the summer of 2005 he joined Stanford’s Smart Antenna Research Group (SARG) as a Visiting Professor. Now he holds a chair on communication systems at the University of Hannover, Germany, and is a founder of the spin-off company mimoOn GmbH. He has published more than 100 papers and has co-edited four books on ultra-wideband and smart antenna systems. He is the founding Editor-in-Chief of the IEEE Signal Processing Society e-letter. His research interest focuses on applied signal processing with emphasis on multi-antenna systems, especially its applicability to ultra-wideband systems.

**Jean-Francois Nezan** is an Assistant Professor at National Institute of Applied Sciences of Rennes (INSA) and a member of the IETR laboratory in Rennes. He received his postgraduate certificate in signal, telecommunications, images, and radar sciences from Rennes University in 1999, and his engineering degree in electronic and computer engineering from INSA-Rennes Scientific and Technical University in 1999. He received his Ph.D. degree in electronics in 2002 from the INSA. His main research interests include image compression algorithms and multi-DSP rapid prototyping.

**Gerhard Schmidt** received his Dipl.-Ing. degree in 1996 and his Dr.-Ing. degree in 2001, both at Darmstadt University of Technology, Germany. Presently, he is working as a senior research engineer in the acoustic signal processing group at Harman/Becker Automotive Systems in Ulm, Germany. His main research interests include adaptive methods for speech and audio processing.
Rapid Industrial Prototyping and SoC Design of 3G/4G Wireless Systems Using an HLS Methodology

Yuanbin Guo,1 Dennis McCain,1 Joseph R. Cavallaro,2 and Andres Takach3

1 Nokia Networks Strategy and Technology, Irving, TX 75039, USA
2 Department of Electrical and Computer Engineering, Rice University, Houston, TX 77005, USA
3 Mentor Graphics, Portland, OR 97223, USA

Received 4 November 2005; Revised 10 May 2006; Accepted 22 May 2006

Many very-high-complexity signal processing algorithms are required in future wireless systems, giving tremendous challenges to real-time implementations. In this paper, we present our industrial rapid prototyping experiences on 3G/4G wireless systems using advanced signal processing algorithms in MIMO-CDMA and MIMO-OFDM systems. Core system design issues are studied and advanced receiver algorithms suitable for implementation are proposed for synchronization, MIMO equalization, and detection. We then present VLSI-oriented complexity reduction schemes and demonstrate how to interact these high-complexity algorithms with an HLS-based methodology for extensive design space exploration. This is achieved by abstracting the main effort from hardware iterations to the algorithmic C/C++ fixed-point design. We also analyze the advantages and limitations of the methodology. Our industrial design experience demonstrates that it is possible to enable an extensive architectural analysis in a short-time frame using HLS methodology, which significantly shortens the time to market for wireless systems.

Copyright © 2006 Yuanbin Guo et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. INTRODUCTION

The radical growth in wireless communication is pushing both advanced algorithms and hardware technologies for much higher data rates than what current systems can provide. Recently, extensions of the third generation (3G) cellular systems such as universal mobile telecommunications system (UMTS) lead to the high-speed downlink packet access (HSDPA) [1] standard for data services. On the other hand, multiple-input multiple-output (MIMO) technology [2, 3] using multiple antennas at both the transmitter and the receiver has been considered as one of the most significant technical breakthroughs in modern communications because of its capability to significantly increase the data throughput. Code-division multiple access (CDMA) [4] and orthogonal frequency-division multiplexing (OFDM) [5] are two major radio access technologies for the 3G cellular systems and wireless local area network (WLAN). The MIMO extensions for both CDMA and OFDM systems are considered as enabling techniques for future 3G/4G systems.

Designing efficient VLSI architectures for the wireless communication systems is of essential academical and industrial importance. Recent works on the VLSI architectures for the CDMA [6] and MIMO receivers [7] using the original vertical Bell Labs layered space-time (V-BLAST) scheme have been reported. The conventional bank of matched filters or Rake receiver for the MIMO extensions was implemented with a target at the OneBSTM base station in [8] for the flat-fading channels [2, 3]. However, in a realistic environment, the wireless channel is mostly frequency selective because of the multipath propagation [9]. Interferences from various sources become the major limiting factor for the MIMO system capacity. Much more complicated signal processing algorithms are required for desirable performance.

For the MIMO-CDMA systems, the linear minimum mean-squared error (LMMSE) chip equalizer [10] improves the performance by recovering the orthogonality of the spreading codes, which is destroyed by the multipath channel, to some extent. However, this in general sets up a problem of matrix inversion, which is very expensive for hardware implementation. Although the MIMO-OFDM systems eliminate the need for complex equalizations because of the use of cyclic prefix, the data throughput offered by the conventional V-BLAST [2, 7, 8] detector is far from the theoretic bound. Maximum-likelihood (ML) detection is theoretically optimal, however, the prohibitively high complexity makes it not implementable for realistic systems. A suboptimal QRD-M symbol detection algorithm was proposed in [5] which approaches the ML performance using limited-tree search.
However, its complexity is still too high for real-time implementation.

These high-complexity signal processing algorithms give tremendous challenges for real-time hardware implementation, especially when the gap between algorithm complexity and the silicon capacity keeps increasing for 3G and beyond wireless systems [11]. Much more processing power and/or more logic gates are required to implement the advanced signal processing algorithms because of the significantly increased computation complexity. System-on-chip (SoC) architectures offer more parallelism than DSP processors. Rapid prototyping of these algorithms can verify the algorithms in a real environment and identify potential implementation bottlenecks, which could not be easily identified in the algorithmic research. A working prototype can demonstrate to service providers the feasibility and show possible technology evolutions [8], thus significantly shortening the time to market.

In this paper, we present our industrial experience in rapidly prototyping these high-complexity signal processing algorithms. We first analyze the key system design issues and identify the core components of the 3G/4G receivers using multiple-antenna technologies, that is, the MIMO-CDMA and MIMO-OFDM, respectively. Advanced receiver algorithms suitable for implementation are proposed for synchronization, equalization, and MIMO detection, which form the dominant part of receiver design and reflect different classes of computationally intensive algorithms typical in future wireless systems. We propose VLSI-oriented complexity reduction schemes for both the chip equalizers and the QRD-M algorithm and make them more suitable for real-time SoC implementation. SoC architectures for an FFT-based MIMO-CDMA equalizer [4] and a reduced complexity QRD-M MIMO detector are presented.

On the other hand, there are many area/time tradeoffs in the VLSI architectures. Extensive study of the different architecture tradeoffs provides critical insights into implementation issues that may arise during the product development process. However, this type of SoC design space exploration is extremely time consuming because of the current trial-and-optimize approaches using hand-coded VHDL/Verilog or graphical schematic design tools [12, 13].

Research in high-level synthesis (HLS) [14–16] aimed at automatically generating a design from a control data flow graph (CDFG) representation of the algorithm to be synthesized into hardware. The specification style of the first commercial realization of HLS is a mixture of functionality and I/O timing expressed in languages such as VHDL, Verilog, SystemC [17], Handel-C [18], or System Verilog. While the behavioral coding style appears more algorithmic (use of loops for instance), the mixture of such behavior with I/O cycle timing specification provides an awkward way to specify cycle timing that often overconstrains the design. This specification style was introduced by Knapp et al. [19] and was the basis for behavioral tools such as Behavioral Compiler introduced in 1994 by Synopsys, Monet introduced by Mentor Graphics in 1997, Volare from Get2Chip (acquired in 2003 by Cadence), CoCentric SystemC Compiler introduced in 2000 by Synopsys, and Cynthesizer from Forte (based on SystemC [17]). The first three tools were based on VHDL/Verilog. All but Cynthesizer are no longer in the market. C-Level’s HLS tool (no longer in the market) used specifications in a subset of C where pipelining had to be explicitly coded. Celoxica’s HLS tool was initially based on cycle-accurate Handel-C [18] with explicit specification of parallelism. Their tool is now called Agility Compiler and it supports SystemC. BlueSpec Compiler targets mainly control-dominated designs and uses System Verilog with Bluespec’s proprietary assertions as the language for specification. Reference [20] presented a Matlab-to-hardware methodology which still requires significant manual design work. To meet the fast changing market requirements in wireless industry, a design methodology that can efficiently study different architecture tradeoffs for high-complexity signal processing algorithms in wireless systems is highly desirable.

In the second part, we present our experience of using an algorithmic sequential ANSI C/C++ level design and verification methodology that integrates key technologies for truly high-level VLSI modeling of these core algorithms. A C-level system synthesizer [21] is applied to explore the VLSI design space extensively for these different types of computationally intensive algorithms. We first use two simple examples to demonstrate the concept of the methodology and how to make these high-complexity algorithms interact with the HLS methodology. Different design modes are proposed for different types of signal processing algorithms in the 3G/4G systems, namely, throughput mode for the VLSI design of the QRD-M MIMO detector is also implemented in a Wildcard PCMCIA card [23]. Our industrial experience demonstrates that it is possible to enable an extensive architectural analysis in a short-time frame using HLS methodology, which leads to significant improvements in rapid prototyping of 3G/4G systems.

The rest of the paper is organized as follows. We first describe the model of 3G/4G wireless systems using MIMO technologies and identify the prototyping and methodology requirements. We then present our prototyping experience for advanced 3G MIMO-CDMA receivers and 4G MIMO-OFDM systems in Sections 3 and 4, respectively.
2. SYSTEM MODEL AND PROTOTYPING REQUIREMENTS

2.1. CDMA downlink system model and design issues

The system model of the MIMO-CDMA downlink with $M$ Tx antennas and $N$ Rx antennas is described here, where usually $M \leq N$. First, the high-data-rate symbols are demultiplexed into $KM$ lower-rate substreams using the spatial multiplexing technology [2], where $K$ is the number of spreading codes used for data transmission. The substreams are broken into $M$ groups, where each substream in the group is spread with a spreading code of spreading gain $G$. The groups of substreams are then combined and scrambled with long scrambling codes and transmitted through the $m$th Tx antenna. The baseband functions are usually implemented in either DSP or FPGA technologies as shown in the physical design block diagram in Figure 1. In a realistic physical implementation, the transmitter has other major modules besides the digital baseband. The protocol stack starts from the media-access-control (MAC) layer up to the network layer, application layer, and so forth. A modern implementation for a wideband system usually applies a direct digital synthesizer (DDS), for example, a component from analog devices or a digital front-end module in FPGA design. A numerically controlled oscillator (NCO) modulates the digital baseband to a digital intermediate frequency (IF). This digital IF waveform is then converted to an analog waveform using a high-speed digital-analog converter (DAC). An analog intermediate frequency (IF) and radio frequency (RF) up-converters modulate the signal to the final radio frequency. The signal passes through a power amplifier (PA) and then is transmitted through the specific antenna.

A system model for the advanced MIMO-CDMA downlink receiver is shown in Figure 2. At the receiver side, corresponding RF/IF down-converters and analog-to-digital converter (ADC) recover the analog signals from the carrier frequency and sample them to digital signals. In an outdoor environment, the signal passing the wireless channel can experience reflections from buildings, trees, or even pedestrians, and so forth. If the delay spread is longer than the coherence time, this will lead to the multipath frequency-selective channel. Significantly, more advanced receiver algorithms are required in these environments besides simple raised-cosine pulse shaping [9] because the simple pulse shaping is not enough for various channel environments. Synchronization is usually the first core design block in a CDMA receiver because it recovers the signal timing with the spreading codes from clock shift and frequency offsets.

For a CDMA downlink system in a multipath fading channel, the orthogonality of the spreading codes is destroyed, introducing both multiple-access interference (MAI) and intersymbol interference (ISI). HSDPA is the evolutionary mode of WCDMA [1], with a target to support wireless multimedia services. The conventional Rake receiver [8] could not provide acceptable performance because of the very short spreading gain to support high-rate data services. LMMSE chip equalizer is a promising algorithm to restore
the orthogonality of the spreading code and suppress both the ISI and MAI [10]. However, this involves the inverse of a large correlation matrix with $O((NF)^3)$ complexity for MIMO systems, where $N$ is the number of Rx antennas and $F$ is the channel length. Traditionally, the implementation of an equalizer in hardware has been one of the most complex tasks for receiver designs.

In a complete receiver design, some channel estimation and covariance estimation modules are required. The equalized signals are descrambled and despread and sent to the multistage interference cancellation (IC) module. Finally, the output of the IC module will be the input to some channel decoder, such as turbo decoder or low-density parity check (LDPC) decoders. The advanced receiver algorithms including synchronization, MIMO equalization, interference cancellation, and channel decoder dominate the receiver complexity. In this paper, we will focus on the VLSI architecture designs of the synchronization and channel equalization because they represent different types of complex algorithms. Although there are tremendous separate architectural research activities for interference cancellation and channel coding in the literature, they are beyond the scope of this paper and are considered as intellectual property (IP) cores for system-level integration.

2.2. System model and design issues for MIMO-OFDM

MIMO-OFDM is considered as an enabling technology for the 4G standards. The OFDM technology converts the multi-path frequency-selective fading channel into flat fading channel and simplifies the channel equalization by inserting cyclic prefix to eliminate the intersymbol interference. The MIMO-OFDM system model with $N_T$ transmit and $N_R$ receive antennas is shown in Figure 3. At the $p$th transmit antenna, the multiple bit substreams are modulated by constellation mappers to some QPSK or QAM symbols. After the insertion of the cyclic prefix and multipath fading channel propagation, an $N_F$-point FFT is operated on the received signal at each of the $q$th receive antennas to demodulate the frequency-domain symbols.

It is known that the optimal maximum-likelihood detector [24] leads to much better performance than the original V-BLAST symbol detection. However, the complexity increases exponentially with the number of antennas and symbol alphabet, which is prohibitively high for practical implementation. To achieve a good tradeoff between performance and complexity, a suboptimal QRD-M algorithm was proposed in [5] to approximate the maximum-likelihood detector. The QR-decomposition [25] reduces the effective channel matrices for $N_T$ transmit and $N_R$ receive antennas to upper-triangular matrices. The M-search algorithm limits the tree search to the $M$ smallest branches in the metric computation. The complexity is significantly reduced compared with the full-tree search of the maximum-likelihood detector. However, the QRD-M algorithm is still the bottleneck in the receiver design, especially for the high-order modulation, high MIMO antenna configuration, and large $M$. It is shown by a Matlab profile that the M-algorithm can occupy more than 99% of the computation in a MIMO-OFDM 4G simulation chain. It can take days or even weeks to generate one performance point. This not only slows the research activity significantly, but also limits the practicability of the QRD-M algorithm in real-time implementation. However, the tree search structure is not quite suitable for VLSI implementation because of intensive memory operations with variable latency, especially for a long sequence. Extensive algorithmic optimizations are required for efficient hardware architecture.
On the other hand, since there is still no standardization of 4G systems, the tremendous efforts to build a prestandard real-time end-to-end complete system still do not give much commercial motivation to the wireless industries. However, there is a strong motivation to demonstrate the feasibility of implementing high-performance algorithms such as the QRD-M detector in a low-cost real-time platform to the business units. There is also a strong motivation to shorten the simulation time significantly to support the 4G research activities. Implementation of the high-complexity MIMO detection algorithms in a hardware accelerator platform with compact form factor will significantly facilitate the commercialization of such superior technologies. The limited hardware resource in a compact form factor and much lower clock rate than PC demands very efficient VLSI architecture to meet the real-time goal. The efficient VLSI hardware mapping to the QRD-M algorithm requires wide-range configurability and scalability to meet the simulation and emulation requirements in Matlab. This also requires an efficient design methodology that can explore the design space efficiently.

2.3. Architecture partitioning requirement

“System-on-a-chip with intellectual property” (SoC/IP) is a concept that a chip can be constructed rapidly using third-party and internal IP, where IP refers to a predesigned behavioral or physical description of a standard component. The ASIC block has the advantage of high throughput speed, and low power consumption and can act as the core for the SoC architecture. It contains custom user-defined interface and includes variable word length in the fixed-point hardware datapath. field-programmable gate array (FPGA) is a virtual circuit that can behave like a number of different ASICs which provide hardware programmability and the flexibility to study several area/time tradeoffs in hardware architectures. This makes it possible to build, verify, and correctly prototype designs quickly.

The SoC realization of a complicated end-to-end communication system, such as the MIMO-CDMA and MIMO-OFDM, highly depends on the task partitioning based on the real-time requirement and system’s resource usage, which roots from the complexity and computational architecture of the algorithms. The system partitioning is essential to solve the conflicting requirements in performance, complexity, and flexibility. Even in the latest DSP processors, computational intensive blocks such as Viterbi and turbo decoders have been implemented as ASIC coprocessors. The architectures should be efficiently parallelized and/or pipelined and functionally synthesizable in hardware. A general architecture partitioning strategy is shown in Figure 4. The SoC architecture will finally integrate both the analog interface and digital baseband together with a DSP core and be packed in a single chip. The VLSI design of the physical layer, one of the most challenging parts, will act as an engine instead of a coprocessor for the wireless link. Unlike a processor type of architecture, high efficiency and performance will be the major target specifications of the SoC design.

2.4. Rapid prototyping methodology requirements

The hardware design challenges for the advanced signal processing algorithms in 3G/4G systems lead to a demand for new methodologies and tools to address design, verification, and test problems in this rapidly evolving area. In [26], the authors discussed the five-ones approach for rapid prototyping of wireless systems, that is, one environment, one automatic documentation, one code revision tool, one code, and one team. This approach also applies to our general requirements of prototyping. Moreover, a good development environment for high-complexity wireless systems should be able to model various DSP algorithms and architectures at the right level of abstraction, that is, hierarchical block diagrams that accurately model time and mathematical operations, clearly describe the real-time architecture, and map naturally to real hardware and software components and algorithms. The designer should also be able to model other elements that affect baseband performance, channel effects, and timing recovery. Moreover, the abstraction should facilitate the modeling of sample sequences, the grouping of the sample sequences into frames, and the concurrent operation of multiple rates inherent in modern communication systems.
The design environment must also allow the developer to add implementation details when, and only when, it is appropriate. This provides the flexibility to explore design tradeoffs, optimize system partitioning, and adapt to new technologies as they become available.

The environment should also provide a design and verification flow for the programmable devices that exist in most wireless systems including general-purpose microprocessors, DSPs, and FPGAs. The key elements of this flow are automatic code generation from the graphical system model and verification interfaces to lower-level hardware and software development tools. It also should integrate some downstream implementation tools for the synthesis, placement, and routing of the actual silicon gates.

3. ADVANCED 3G RECEIVER REAL-TIME PROTOTYPING

The advanced HSDPA receiver for rapid prototyping is the evolutionary mode of WCDMA [1] to support wireless multimedia services in the cellular devices. MIMO extensions are proposed for increased data throughput. In this section, we present our real-time industrial prototyping designs for the advanced receiver using high-complexity signal processing algorithms.

3.1. System partitioning

Because of the real-time demonstration requirement, the complete system design needs a lot of processing power. For example, the turbo decoder for the downlink receiver alone occupies 80% of the area of a Virtex II V6000. We apply the Nallatech BenNUEY multiple-FPGA computing platform for the baseband architecture design. Each motherboard can hold up to seven BenBlue II user FPGAs in a single PCI motherboard. These FPGAs include Xilinx Virtex II V6000 to V8000. Multiple I/O and analog interface cards can also be attached to the PCI card. This provides a powerful platform for high-performance 3G demonstration. We also apply TI’s C6000 serial DSP to support high-speed MAC layer design.

In the transmitter, the host computer runs the network layer protocols and applications. It has interfaces with the DSP, which hosts the media-access-control (MAC) layer protocol stack and handles the high-speed communication with FPGAs. A DSP interface core in the transmitter reads the data from the DSP and adds cyclic redundancy check (CRC) code. After the turbo encoder, rate matching, and interleaver, a QPSK/QAM mapper modulates the data according to the hybrid automatic retransmission request (HARQ) control signals. With the common pilot channel (CPICH) and synchronization channel (SCH) information inserted, the data symbols are spread and scrambled with pseudonoise (PN) long code and then ported to the RF transmitter. At the receiver, the searcher finds the synchronization point. Clock tracking and automatic frequency control (AFC) are applied for fine synchronization. After the matched filter receiver, received symbols are demodulated and deinterleaved before the rate dematching. Then after a turbo decoder decodes the soft decisions to a bit stream, a HARQ block is followed to form the bit stream for the upper-layer applications. In Figure 5, we also depict other key advanced algorithms including channel estimation, chip-level equalizer, and multistage interference cancellation to eliminate the distortions caused by the wireless multipath and fading channels. The clock tracking and AFC which are slightly shaded will be used as the simple cases to demonstrate the concept of using Catapult C HLS design methodology. The darkly shaded blocks in the MIMO scenario will be the focus for high-complexity architecture design.
3.2. CDMA receiver synchronization

3.2.1. Clock-tracking algorithm

The mismatch of the transmitter and receiver crystals will cause a phase shift between the received signal and the long scrambling code. The “clock-tracking” algorithm [27] will track the code sampling point. The IF signal is sampled at the receiver and then down-converted with a digital demodulation at local frequency. The separated I/Q channel is then downsampled to four phases’ signals at the chip rate, which is 3.84 MHz. By assuming one phase as the in-phase, we compute the correlation of both the earlier phase and the later phases with the descrambling long code according to the frame structure of HSDPA. When the correlation of one phase is much larger than another phase (compared with a threshold), it will then be judged that the sample should be moved ahead or delayed by one-quarter chip. Thus the resolution of the code tracking can be one quarter of a chip. This principle is shown in Figure 6.

The system interface for clock tracking is also depicted in Figure 6. At the downsampling block after the DDC (digital down-converter) Xilinx core, the in-phase, early, late phases are sent to both the Rake receiver and clock tracking. The long code will be loaded from ROM block. The clock-tracking algorithm computes both early/late correlation powers after descrambling, chip-matched filter, and accumulation stages. A flag is generated to indicate early, in-phase or late as output. This flag is used to control the adjustment signal of a configurable counter. The adjusted in-phase samples are then sent to the Rake receiver for detection. Thus the clock tracker is integrated with IP cores and the other HDL designer blocks (downsampling, MUX, etc.).

3.2.2. Automatic frequency control

The frequency offset is caused by the Doppler shift and frequency offset between the transmitter and the receiver oscillators. This makes the received constellations rotate in addition to the fixed channel phases, and thus dramatically degrades performance. AFC is a function to compensate for the frequency offset in the system. For a software definable radio (SDR) type of architecture, the frequency offset is computed with a DSP algorithm and controlled by a numerical control oscillator (NCO).

We apply a spectrum-analysis-based AFC algorithm. The principle is explained with the frame structure of HSDPA in Figure 7. There are 15 slots in each frame. In each slot, the first 5 bits are pilot symbols and the second 5 bits are control signals. Each symbol is spread by a 256-chip long code. So in the algorithm, we first use a long code to descramble the received signal at the chip rate. We then do the matched filtering by accumulating 256 chips. By using the local pilot’s conjugate, we get the dynamic phase of the signal with the frequency offset embedded. To increase the resolution, we finally accumulate each of the 5 pilot bits as one sample. The 5-bit control bits are skipped. Thus the sampling rate for the accumulated phase signals is reduced to be 1500 Hz. These samples are stored in a dual-port RAM for the spectrum analysis using FFT. After the descrambling and matched filter, as well as accumulation, we achieve a very stable sinusoid waveform for the frequency offset signal as shown in the figure.

3.3. VLSI system architecture for FFT-based equalizer

LMMSE chip equalizer is promising to suppress both the intersymbol interference and multiple-access interference [4] for a MIMO-CDMA downlink in the multipath fading channel. Traditionally, the implementation of equalizer in hardware has been one of the most complex tasks for receiver designs because it involves a matrix inverse problem of some large covariance matrix. The MIMO extension gives even more challenges for real-time hardware implementation.

In our previous paper [4], we proposed an efficient algorithm to avoid the direct matrix inverse in the chip equalizer.
by approximating the block Toeplitz structure of the correlation matrix with a block circulant matrix. With a timing and data-dependency analysis, the top-level VLSI design blocks for the MIMO equalizer are shown in Figure 8. In the front end, a correlation estimation block takes the multiple input samples for each chip to compute the correlation coefficients of the first column of \( R_{rr} \). Another parallel data path is for the channel estimation and the \((M \times N)\) dimensionwise FFTs on the channel coefficient vectors. A submatrix inverse and multiplication block take the FFT coefficients of both channels and correlations from DPRAMs and carry out the computation. Finally an \((M \times N)\) dimensionwise IFFT module generates the results for the equalizer taps \( \hat{w}_{opt} \) and sends them to the \((M \times N)\) MIMO FIR block for filtering. To reflect the correct timing, the correlation and channel estimation modules and MIMO FIR filtering at the front end will work in a throughput mode on the streaming input samples. The FFT-inverse-IFFT modules in the dotted-line block construct the postprocessing of the tap solver. They are suitable to work in a block mode using dual-port RAM blocks to communicate the data.

4. ADVANCED RECEIVER FOR 4G MIMO-OFDM

4.1. Reduced-complexity QRD-M detection

The complexity of the optimal maximum-likelihood detector in MIMO-OFDM systems increases exponentially with the number of antennas and symbol alphabet. This complexity is prohibitively high for practical implementation. In this section, we explore the real-time hardware architecture of a suboptimal QRD-M algorithm proposed in...
[5] to approximate the maximum-likelihood detector. It is shown that the symbol detection is separable according to the subcarriers, that is, the components of the \( N_r \) subcarriers are independent. Thus, this leads to the subcarrier-independent maximum-likelihood symbol detection as \( d_{_{\text{ML}}}^k = \arg \min_{d \in \{0,1\}^{N_r}} \| y^k - \hat{H}^k d^k \|^2 \), where \( y^k = [y_1^k, y_2^k, \ldots, y_{N_r}^k]^T \) is the \( k \)th subcarrier of all the receive antennas, \( \hat{H}^k \) is the channel matrix of the \( k \)th subcarrier, \( d^k = [d_1^k, d_2^k, \ldots, d_{N_r}^k]^T \) is the transmitted symbol of the \( k \)th subcarrier for all the transmit antennas. The QR-decomposition [25] reduces the \( K \) effective channel matrices for \( N_t \) transmit and \( N_r \) receive antennas to upper-triangular matrices. The M-search algorithm limits the tree search to the \( M \) smallest branches in the metric computation. The complexity is significantly reduced compared with the full-tree search of the maximum-likelihood detector. The procedure is depicted in Figure 9 for an example with QPSK modulation and \( N_t \) transmit antennas where only the survival branches are kept in the tree search.

4.2. System-level hardware/software partitioning

As explained earlier, there is a new requirement for a precommercial functional verification and demonstration of the high-complexity 4G receiver algorithms. To reduce the high industrial investment of complete system prototyping before the standard is available, it makes more sense to focus on the core algorithms and demonstrate them by the hardware-in-the-loop (HITL) testing. Although the Nallatech system could also be applied for this purpose, we prefer an even more compact form factor. Thus, we propose to use Annapolis WildCard to meet both the HITL and simulation acceleration requirements. The WildCard is a single PCMCIA card which contains a Virtex II V4000 FPGA for laptops. The details of the hardware platform are found in [23].

To achieve simulation-emulation codesign, an efficient system-level partitioning of the MIMO-OFDM Matlab chain is very important. The simulation chain is depicted in Figure 10. In the simplified simulation model, the MIMO transmitter first generates random bits and maps them to constellation symbols. Then the symbols are modulated by IFFT’s. A multipath channel model distorts the signal and adds AWGN noises. The receiver part is contained in the function Hard_qrdm_fpga, which consists of the major subfunctions such as demodulator using FFT, sorting, QR decomposition, the M-search algorithm in a C-MEX file, the demapping, and the BER calculator.

In the implementation of the QRD-M algorithm, the channel estimates from all the transmit antennas are first sorted using the estimated powers to make \( \hat{P}^{(n_1)} \leq \hat{P}^{(n_2)} \leq \cdots \leq \hat{P}^{(n_t)} \). The data vector \( d^k \) is also reordered accordingly. Then the QR decomposition algorithm is applied to the estimated channel matrix for each subcarrier as \( Q^H_k \hat{H}^k = R_k \), where \( Q_k \) is the unitary matrix and \( R_k \) is an upper-triangular matrix. The FFT output \( y^k \) is premultiplied by \( Q_k^H \) to form a new receive signal as \( Y_k = Q_k^H y^k = R_k d^k + w_k \), where \( w_k = Q_k^H z_k \) is the new noise vector. The ML detector is equivalent to a tree search beginning at level (1) and ending at level \( (N_t) \), which has a prohibitive complexity at the final stage as \( \Theta((N_t)^{N_r}) \). The M-algorithm only retains the paths through the tree with the \( M \) smallest aggregate metrics. This forms a limited tree search which consists of both the metric update and the sorting procedure. The readers are referred to [5] for details of the operations.

The top five most time-consuming functions in the simulation chain are shown in Figure 11 for the original C-MEX design for 64-QAM. The run time is obtained by the Matlab “profile” function. Function “Hard_qrdm” is the receiver function including all “m_mex_orig,” “channel,” “qr,” and “mapping” subfunctions, where the QR-decomposition calls the Matlab built-in function. It is shown that for the original floating-point C-MEX implementation, the C-MEX implementation of the M-search function “m_mex_orig” dominates more than 90% of the simulation time. Moreover, all the other functions consume negligible time compared with the M-search function.

The M-search algorithm in the C-MEX file is thus implemented in the FPGA hardware accelerator. APIs talk with the CardBus controller in the card board. The controller then communicates with the processing element (PE) FPGA through the local address data (LAD) bus standard interface, which is part of the PE design. The data is stored in the input buffer and a hardware “start” signal is asserted by writing to the in-chip register. The actual PE component contains the core FPGA design to utilize both the multistage pipelining in the MIMO antenna processing and the parallelism in the subcarrier. After the output buffer is filled with detected symbols, the interrupt generator asserts a hardware interrupt signal, which is captured by the interrupt wait API in the C-MEX file. Then the data is read out from either DMA channel or status register files by the LAD output multiplexer.
To achieve the bidirectional data transfer, both the source and destination DMA buffers are needed.

The architecture is designed in multistage processing elements with shared DPRAM for communication between stages. Each stage processes the detection of one Tx antenna. The symbol detection of each antenna includes three major tasks: the metric computation, sorting, and symbol detection as shown in Figure 12. An example for the antenna nT4 is shown in Figure 13. All the central antennas have the same operations with much higher complexity than the first and last antennas.

4.3. Partial limited tree search

Although the number of complex multiplications is an important complexity indicator because it determines the number of multipliers in a VLSI design, the real-time latency bottleneck is the sorting function. This is because the metric computation can be pipelined in the VLSI architecture with a regular structure, but the sorting function involves extensive memory access, conditional branching, element swapping, and so forth depending on the ordering feature of the input sequence.

Theoretically, the fastest sort function has the complexity at the order of $O(MC \times \log_2(MC))$. However, the complexity of the full sorting is too high. For example, for 64-QAM with $M = 64$, the sequence length is 4096. Then there are at least 40152 operations. If the sequence needs to be stored in block memory, this means at least these many cycles in hardware latency without counting the swapping, branching overheads. This results in 500 microseconds for a single subcarrier and one antenna assuming 100 MHz clock rate, which is very challenging to meet the real-time requirement.

However, we note that because we only retain the $M$ smallest survivor branches, we do not care about the order of the other sequences above the $M$ smallest metric. So only the $M$ smallest metrics from the $MC$ metric sequence need to be sorted. Using this observation, we modified the standard “quick-sort” procedure to the so-called “partial quick-sort” architecture.

For the partial quick-sort architecture, the metric sequence is computed separately and stored in the tmpMetric shared DPRAM blocks. Moreover, the Qsort index DPRAM contains the initial value of the sequence indices. A “iStack” RAM block acts as the stack to store the temporary boundary of the partitioned potential subsequences $il$, $ir$. A partial Qsort Core loads/writes data from and to the DPRAM blocks according to a finite-state machine (FSM) according to the logic flow of the partial quick-sort procedure. If the partitioned and exchanged subsequence reaches a short length, the short subsequence is sorted using the insert sort.
5. **CATAPULT C HLS DESIGN METHODOLOGY**

### 5.1. Classical hardware implementation technologies

The most fundamental method of creating hardware design for an FPGA or ASIC is by using industry-standard hardware description language (HDL), such as VHDL or Verilog [13], based on data flow, structural or behavioral models. The design is specified in a register-transfer level (RTL) where the cycle-by-cycle behavior is fully specified. The details of the microarchitecture are explicitly coded. When writing the RTL description, the designer has to specify what operations are executed in what cycles, what registers are going to store the results of the operations, how and when memories are going to be accessed, and so forth. The RTL design process is manual and the intrinsic architecture tradeoffs need to be studied offline. After the architecture is crafted, the RTL code is written and validated using simulation by comparing the behavior of the RTL against the behavior of the original algorithm. After a few iterations of simulating, debugging, and code fixing, the RTL design is ready to be synthesized into the target technology (ASIC or FPGA). The results of synthesis may reveal that the design will not run at the specified frequency due to delays that were not fully accounted for when crafting the architecture such as delays from routing, multiplexing, or control logic. The results of synthesis may also reveal that the design exceeds the allocated budget for either area or power. However, it is not easy to change a design dramatically once the hardware architecture is laid out.

### 5.2. Raising the level of abstraction

The fundamental weakness of the RTL design methodology is that it forces designers to mix both algorithmic functionality (what the design computes) with detailed cycle timing of how that functionality is implemented. This means that the RTL code is committed to a given performance and interface requirements in conjunction to the target ASIC/FPGA technology. The low level of abstraction makes the RTL code complex and highly dependent on the crafted architecture.
Raising the level of the abstraction was recognized by researchers as a necessary step to address the issues with RTL outlined above. The most important tasks in HLS are scheduling and allocation tasks that determine the latency/throughput as well as the area of the design. Scheduling involves assigning every operation (node in the CDFG) into control steps (c-steps). Resource allocation is the process of assigning operations to hardware with the goal of minimizing the amount of hardware required to implement the desired behavior. The hardware resources consist primarily of functional units, storage elements (registers/memory), and multiplexes. Once the operations in a CDFG have been scheduled into c-steps, an implementation consisting of an FSM and a data path can be derived. Depending on the delay of the operations (dependent on target technology), the clock frequency constraint, and performance or resource constraints, a variety of designs can be produced from the same specification. Parallelism between operations in the same basic block (data-flow graph) is analyzed and exploited according to what hardware resource is allocated. Parallelism across control boundaries is exploited using loop unrolling, loop pipelining, and by analyzing data dependencies across conditionals. The research studied ways to optimize the hardware by means of how functional resources are allocated, how operations are scheduled and mapped to the available resources, and how variables are mapped to registers or to memory.

The first commercial encarnalizations of HLS took an incremental approach to HLS and most HLS synthesis tools have, to this date, followed that trend. The goal was to improve productivity by partially raising the abstraction of RTL and applying HLS techniques to synthesize such specifications. The specification style is a mixture of functionality and I/O timing expressed in languages such as VHDL, Verilog, SystemC [17], Handel-C [18], or System Verilog. One of the main reasons for the desire of keeping I/O timing in the specification is to explicitly code interface timing into the specification. Interface exploration and synthesis are not built in as an intrinsic part of such methodologies. While the behavioral coding style appears more algorithmic (e.g., use of loops), the mixture of such behavior with I/O cycle timing specification provides an awkward way to specify cycle timing that often overconstrains the design.

### Algorithmic specification

The algorithmic specification is expressed in ANSI C/C++ where the function to be synthesized is specified either at the source (with a #pragma design top) or during synthesis. The interface of the function determines what data goes in and out of the function, though it does not specify how the data is transferred over time (that is determined during synthesis). For instance, the specification for an FIR filter may look as in Algorithm 1.

```
#pragma design top
void fir (int 8 x, int 8 *y) {
    static int 8 taps [12];
    ...
}
```

(i) The source is concise, the easiest to write, maintain, and debug. Because of its high-level of abstraction, its behavior can be simulated at much higher speeds ($\times10000$ faster) than RTL, cycle accurate, or traditional behavioral-level specifications.

(ii) The source can be leveraged as algorithmic intellectual property (IP) that may be targeted for various applications and ASIC/FPGA technologies.

(iii) Obtaining a new architecture is a matter of changing architectural constraints during synthesis. This reduces the risk of prolonged manual recoding of the RTL to address last-minute changes in requirements or to address timing closure or to satisfy power and area budgets.

(iv) By avoiding manual coding of the architecture in the source, functional bugs that are common when coding RTL are also avoided. It is estimated that 60% of all bugs are introduced when writing RTL. The importance of avoiding such bugs cannot be overstated.

### 5.3.1. Algorithmic specification

Catapult C synthesis is the first HLS approach that raises the level of abstraction by clearly separating algorithmic function from the actual architecture to implement it in hardware (interface cycle timing, etc.). The inputs to the Catapult C are (a) the algorithmic specification expressed in sequential, ANSI-standard C/C++ and (b) a set of directives which define the hardware architecture. The clear separation of function and architecture allows the input source to remain independent of interface and performance requirements and independent of the ASIC/FPGA target technology. This separation provides important benefits.
behavior at the interface. Catapult C provides feedback to determine if a variable was optimized and it is possible to further numerically refine the algorithm to improve performance, area, and power. All numerical refinement should be done at the source level as the generated RTL by Catapult C should faithfully reflect the bit-accurate behavior of the source. Floating-point arithmetics at the precision of the C floating-point types are not practical for hardware implementation so they are not fully supported for synthesis. They can be mapped to fixed-point types, but that flow is mainly used for estimation as the bit accuracy of the source is in general not preserved in the generated RTL.

While it is possible to use the native C integers in conjunction with shifts and bit masking to model bit-accurate fixed-point and integer arithmetics, it makes the source less readable and makes coding of the algorithm more error prone. By using bit-accurate fixed-point and integer data types, it is possible to cleanly numerically refine an algorithm that originally used floating-point types. The data types are simply classes that encapsulate the data and provide the relevant operators to perform arithmetic, access bits or bit ranges, rounding and saturation, and so forth. Catapult C supports the integer and fixed-point data-type library provided by SystemC as well as the Algorithmic C data-type library. The Algorithmic C library provides arbitrary-length bit-accurate fixed-point and integer types that offer fast simulation speed with uniform and well-defined synthesis semantics. For either data-type libraries, the precision, quantization, and overflow modes are specified as template parameters. For example, the int8 used in the FIR example is a typedef to the Algorithmic C type ac_int<8, true>, where the first parameter specifies its bit width and the second parameter specifies whether it is unsigned or signed. Moreover, it is possible to define new data types by using the class or structure to simplify the C level code. Such an example shown in Algorithm 2 will be used later in this paper to show the code style.

There are no restrictions on what functions are called with the only exception that recursive functions are not supported (though recursion based on template parameters is supported). Functions that are not marked as designs are inlined during synthesis. Memory allocation and deallocation are not supported. Pointers are supported provided that the object that they point to is statically determinable. This condition is not overly restrictive as function inlining helps to resolve pointers.

Algorithm 2: Catapult C code for complex class definition.

```c

class Cplx {
public: int r, i;
Cplx (int r, int i) : r(r), i(i) {}
Cplx operator + (const Cplx & other) {
    return Cplx (r + other.r, i + other.i);
};
Cplx operator * (const Cplx & other) {
    return Cplx (r*other.r - i*other.i, 
                i*other.r + r*other.i);
};
Cplx conj() { return Cplx (r, -i); }
int pow() { return r*r + i*i; }
}
```

5.3.2 Architectural synthesis

The design is crafted during synthesis by interactively applying architectural constraints, analyzing the results, and further refining the architectural constraints if necessary. There are two constraints that the user must provide: the target technology and the clock frequency. If no other constraint is specified, default constraints are used to define the interface and required performance. A number of analysis tools provide feedback about the schedule, allocation, variable-to-register mappings, latency/throughput, area, timing reports, schematics with cross-probing links among them, and to the input source.

Architectural constraints define the interface, how internal arrays are mapped to storage (memory, registers, register file), and the level of interloop or intraloop parallelism that is exposed to synthesis. Interface synthesis provides a way to instruct synthesis to transfer data with a specific protocol. The generation of all the appropriate signals and their scheduling constraints are automatically captured by synthesis. For example, the interface could be a register/register file, memory, a FIFO, a bus, and so forth. The granularity of the data transfer is also defined during interface synthesis. For example in many cases, the even and odd array elements are transferred concurrently to increase the transfer bandwidth. Internal arrays may be mapped to any defined storage elements including memories or register files supporting different numbers of concurrent reads and writes.

Parallelism between operations across iterations of a loop (intraloop parallelism) can be exploited using either loop unrolling or loop pipelining. Loop unrolling (whether full or partial) unfolds one or more iterations of the loop by creating one or more copies of the body. For instance, unrolling a loop by 2 reduces the iteration count in half. If there is intraloop parallelism (and resource constrains do not interfere), the new loop body will have less than twice the latency of the original loop body. The net effect is that the latency of the loop is reduced. Loop pipelining on the other hand is a structural approach that overlaps execution of subsequent loop iterations. If there is intraloop parallelism (and resource constraints do not interfere), then the interval between iterations of a loop will be shorter than the latency of the loop body. The net effect is that the overall latency of the loop is reduced or that the throughput of the loop is increased. Interloop parallelism is mainly accomplished by loop merging. Loop merging cannot only improve the latency of the design by merging two or more loops in one, but often reduces storage requirements as data, that otherwise would need to be saved, is immediately consumed in the merged loop.

5.3.3 Integrated Catapult C verification methodology

Catapult C provides a flow to verify that the functionality of the generated RTL is consistent with the original C source.
This verification flow consists of building all the necessary SystemC wrappers and transactors to use the original C test-bench to test the generated RTL. This is an important piece of the C-based methodology as manual generation of test-benchs is time consuming, error prone, and architecture dependent. The verification flow also provides a convenient way to gather toggle information that is useful for power estimation.

6. APPLYING CATAPULT C METHODOLOGY FOR 3G/4G: DESIGN FLOW AND EXPERIMENTAL RESULTS

In this section, we describe how we apply Catapult C [21] in an integrated design flow for our high-complexity 3G/4G core algorithms design. We also show our experimental results and demonstrate how to achieve both architectural efficiency and productivity for modeling, partitioning, and synthesis of the complete system.

The first author at Nokia Research Center became a Beta user of Mentor’s HLS technology in 2002 in order to develop a complete rapid prototyping methodology for advanced wireless communication systems. The computationally intensive nature of wireless algorithms made it a perfect match for HLS synthesis. Catapult C was officially released in the ACM Design Automation Conference (DAC) 2004 in San Diego, Calif, where the first author was one of the speakers in an expert panel.

To explore the VLSI design space, the system-level VLSI design is partitioned into several subsystem blocks (SBs) according to the functionality and timing relationship. The intermediate tasks will include high-level optimizations, scheduling and resource allocation, module binding, and control circuit generation. The proposed procedure for implementing an algorithm to the SoC hardware includes the following stages as shown in Figure 14 and is described as follows.

(1) Algorithm verification in Matlab and ANSI C/C++: in the algorithmic-level design, we first use Matlab to verify the floating-point algorithm based on communication theory. The matrix-level computations must be converted to plain C/C++ code. All internal Matlab functions such as FFT, SVD, eigenvalue calculation, complex operations, and so forth need to be translated with efficient arithmetic algorithms to C/C++.

(2) Catapult C HLS: RTL output can be generated from C/C++ level algorithm by following some C/C++ design styles. Many FUs can be reused in the computational cycles by studying the parallelism in the algorithm. We specify both timing and area constraints in the tool and let Catapult C schedule efficient architecture solutions according to the scheduling directives. The number of FUs is assigned according to the time/area constraints. Software resources such as registers and arrays are mapped to hardware components, and FSMs for accessing these resources are generated. In this way, we can study several architecture solutions efficiently and achieve the flexibility in architectural exploration.

(3) RTL integration and module binding: in the next step of the design flow, we use HDL designer to import the RTL output generated by Catapult C. A test bench is built in HDL designer corresponding to the C++ test bench and simulated using ModelSim. At this point, several IP cores might be integrated, such as the efficient cores from Xilinx CoreGen library (RAM/ROM blocks, CORDIC, FIFO, pipelined divider, etc.) and HDL ModuleWare components for the test bench.

(4) Gate-level hardware validation: Leonardo spectrum or precision-RTL is invoked for gate-level synthesis. Place and route tools such as Xilinx ISE are used to generate gate-level bit-stream file. For hardware verification and validation, a configurable Nallatech hardware platform is used. The hardware is tested and verified.
by comparing the logic analyzer or ChipScope probes with the ModelSim simulation.

6.1. Architecture scheduling and resource allocation

In general, more parallel hardware FUs mean faster design at the cost of area, while resource sharing means smaller area by trading off execution speed. Even for the same algorithm, different applications may have different real-time requirements. For example, FFT needs to be very fast for OFDM modulation to achieve high data throughput, while it can be much slower for other applications such as in a spectrum analyzer. The best solution would be the smallest design meeting the real-time requirements, in terms of clock rate, throughput rate, latency, and so forth. The hardware architecture scheduling is to generate efficient architectures for different resource/timing requirements.

In Catapult C, first we specify the general requirements on the CLK rate, standard I/O, and handshaking signals such as RESET, START/READY, DONE signals for a system. The detailed procedure within Catapult C is shown in Figure 15. Then we can specify the building blocks in the design by choosing different technique libraries, for example, RAM library and CoreGen library. This will map the basic components to efficient library components such as divider or pipelined divider from the C/C++ language operator “/.”

In a C-level design, the arrays are usually mapped to memory blocks. In some cycles, some FUs might be in IDLE state. These FUs could be reused by other similar computations that occur later in the algorithm. Thus, there will be many possible resource multiplexings in an algorithm. Multiplexing FUs manually is extremely difficult when the algorithm is complicated, especially when hundreds or even thousands of operations use the same FUs. Therefore, multiple FUs must be applied even for those independent computations in many cases. The size can be several times larger with the same throughput as in Catapult C solution. In Catapult C, we specify the maximum number of cycles in resource constraints. We can analyze the bill of material (BOM) used in the design and identify the large-size FUs. We can limit the number of these FUs and achieve a very efficient multiplexing. With the detailed reports on many statistics such as the cycle constraints and timing analysis, we can easily study the alternative high-level architectures for the algorithm and rapidly get the smallest design by meeting the timing as much as possible.

The programming style is essential to specify the hardware architectures in the C/C++ program. Several high-level conventions are defined to specify different architectures to be used. We use Catapult C to design architectures in two basic modes according to the behavior of the real-time system: the throughput mode for front-end processing and the block mode for postprocessing modules.

6.2. Throughput-mode front-end processing architectures

Throughput mode assumes that there is a top-level main loop. In each computation period, the data is inputted into the function sample by sample. The function will process for
each sample input. Usually, no handshaking signals are required. The temporary values are kept by using static variables. The throughput is determined by the latency of the processing for each sample. Therefore, it is more suitable for the sample-based signal processing algorithms. Typical computations for this mode are filtering and accumulation-type computations in wireless systems.

6.2.1. Clock tracking

The clock-tracking algorithm could be designed with a conventional manual layout architecture in HDL designer. We would most likely build a parallel architecture with duplicate FUs as in Figure 16. First, we will have a descrambling procedure that is a complex multiplication with the long code. Then we will have a chip-matched filter that is basically mapped to an accumulator. Then after each symbol, we need to compute power and accumulate for each frame. We finally have a comparator to make a decision. Altogether, we will have copies for both early and late paths. This requires 16 multipliers and 12 adders. This architecture is optimal for fully pipelined computation where a sample will be inputted at the chip rate for each 10 cycles. The pipeline is idle for the other 38 cycles. However, in our system, since we use a 4 MHz clock rate, only one sample will be inputted at the chip rate for each 10 cycles. The pipeline is idle for the other 9 cycles and the resources are wasted.

We now use Catapult C flow to design and schedule the architecture of this algorithm. The throughput-mode code is shown in Algorithm 3. We use the complex class to declare the interface variables “eRake,” “LRake,” which denote early- and late-phase Rake outputs, respectively. We also obtain the “LongCode” for each chip to do the descrambling. Note that we use static to declare the accumulation variables “aEarly” and “aLate” as well as the chip counter “i” and frame counter “j.” All the I/O variables are not arrays. Thus, this module is called for each chip duration based on the streaming input chip samples. With Catapult C, we scheduled several solutions for the same source code by setting different constraint directives as in Table 1. In these designs, FUs are multiplexed within the timing constraints. Because of the computation dependency, there will be a necessary latency for the first computation result to come out even if we use many FUs. For example, in solution 1, although we use 8 multipliers and 6 adders, the best we can achieve is 7-cycle latency. The size is huge with 5600 FPGA lookup tables (LUTs). By setting the number of constraints and the maximal acceptable number of cycles (10 cycles), we will have different solutions with sizes ranging from 2000 to 1300 LUTs. We can choose the smallest design, that is, solution 4, for implementation while still meeting the timing constraint.

Figures 17 and 18 show the computation procedures of two typical solutions of clock tracking in Gantt graphs. The horizontal axis is the cycle for one period, and the vertical axis shows the mapped FUs for each computation. The long bars denote multiplications and short bars denote either “+” or “−” operations. The connections lines show the data dependency between operations. Figure 17 shows the fully parallel speed-constrained solution 1 with 8 multipliers. All 8 multipliers are used in parallel in cycle 1. Then 4 MULTs are used again in cycle 3. But in several other cycles, they are not used any more for the rest of the computation period. However, as shown in solution 4 in Figure 18, one single multiplier is reused in each cycle, by avoiding the dependency. After each multiplication, an addition follows and for the cycles 2–9, multiplications and additions are done in parallel. Moreover, we still meet the 10-cycle timing constraint easily. In solution 4, the hardware is used most efficiently. This is almost the minimal possible size that could be achieved theoretically for this particular algorithm. The savings in hardware can also reduce the power consumption that is a critical specification for mobile systems.

6.2.2. MIMO covariance estimation
design space exploration

There are two major front-end modules for covariance estimation and channel estimation for the MIMO chip equalizer in Figure 8. These modules essentially are similar to the clock-tracking algorithm. While the clock-tracking algorithm computes the cross-correlation between the chip
#include "mc_bitvector.h"
#pragma design top
uint2 ctrk (Cplx eRake, Cplx LRake, Cplx LongCode, short threshold)
{
    static Cplx aEarly(0,0), Cplx aLate(0,0);
    static unsigned pAE = 0, pAL = 0; static uint9 i = 0, j = 0;
    int 2 flag = 0; Cplx tEarly, tLate;
    //descramble
    tEarly = eRake * LongCode. conj();
    tLate = LRake * LongCode.conj();
    //accumulate through symbol
    aEarly = aEarly + tEarly;
    aLate = aLate + tLate;
    ++i;
    if (i = 256) //accumulate through frame
        pAE = aEarly · Pow();
        pAL = aLate · Pow();
        aEarly = Cplx(0, 0); aLate = Cplx(0, 0);
        i = 0; ++j;
        if (j = 150) { j = 0; //compare early/late gate with threshold
            if ((pAE − threshold) > pAL) flag = −1;
            else if ((pAL − threshold) > pAE) flag = 1;
            else flag = 0;
        }
    return(flag);
}

Algorithm 3: Catapult C code for throughput-mode-based clock tracking.

| Solution | LUTs | Cycle | MULT # | ADD # | MUX (LUT) |
|----------|------|-------|--------|-------|-----------|
| 1        | 5628 | 7     | 8      | 6     | 1221      |
| 2        | 2004 | 10    | 2      | 2     | 1152      |
| 3        | 1426 | 16    | 1      | 1     | 623       |
| 4        | 1361 | 10    | 1      | 2     | 616       |

6.3. Block-mode postprocessing architectures and integration

6.3.1. AFC

Corresponding to the AFC algorithm shown in Figure 7, we designed the hardware architecture as shown in Figure 20. IP cores from different sources are integrated in HDL designer.
Figure 17: Gantt graph for speed-constrained architecture for clock tracking from Catapult C scheduling: 8 multipliers, 6 adders, 4 subtractors, 7-cycle latency.

Figure 18: Gantt graph for area-constrained architecture for clock tracking: 1 adder, 1 subtractor, 10-cycle latency.

Figure 19: CLB versus number of input bits for the covariance estimation block (MIMO correlation block) with different architectures.

A Xilinx core direct digital synthesis (DDS) block controlled by the AFC module generates the local frequency to demodulate the RF front-end received signal. Some ROM cores are used to store the long codes and pilot symbols as well as the phase coefficients for the FFT. Three separate Catapult C blocks are pipelined: the AFC accumulation block, the 256-point FFT block, and a SearchMax block. The accumulator and descrambler need to process for each input sample and will work in a throughput mode. The search is invoked by the FFT once the FFT is finished. The processes will use dual-port RAMs for communication. All the IP cores are integrated in HDL designer with additional glue logic.

Although the Xilinx core library also provides a variety of FFT IP cores, they are usually for high-throughput applications, and they usually have considerably large sizes. But in our algorithm, the FFT only processes once for each complete frame block, so we can relax the timing constraint to get a very compact design with minimum resource usage. In block mode, the function processes once after a block of data is ready. Example Catapult C code style is shown in Algorithm 4 for a 32-point radix-2 FFT/IFFT module. The minimum resource code for 256-point FFT is essentially the same as that of a 32-point FFT except for the cosine/sine coefficients and the number of loops. First, we need to include the "mc_bitvector.h" to declare some Catapult C-specific bit vector types such as the int16, uint2, and so forth. We first convert the cosine/sine phase coefficients to integer numbers and store them in two vectors that will be mapped to ROM hardware as cos_v and sin_v. If we consider the FFT module as the top level of the partitioned Catapult C module, we need to declare the #pragma design top. The input and output arrays ar0[], ai0[] could be mapped to dual-port RAM blocks in hardware. The flag is a signal to configure whether it is an FFT or IFFT module. It can be seen that the Catapult C style is almost the same as the ANSI-C style. There is no need to specify the timing information in the source code.

In the core algorithm, there are different levels of loop structures, that is, the stage level, the butterfly-unit level, and the implementation of the butterfly units. Based on the loop structure and the storage hardware mapping, we can specify the different architecture constraints within the Catapult C GUI interface to generate the desired RTL architecture. The hardware interface used RAM blocks to pass the data. Catapult C will generate FSMs for the write enabling, MEM address/data bus, and control logic. The complete AFC algorithm only needs to be updated once in each frame length, which is 10 milliseconds. We designed several solutions with only 1 multiplier and 1 adder reused for each MULT and ADD operation. The latency is larger than the Xilinx core, but the area is smaller. Finally, for all three blocks and different-point FFT, we achieve the same minimal size around 1000 LUTs, saving about ×3 in the number of LUTs over the Xilinx Core as shown in Table 2. Although in general a design with more functional units could work faster than a design with less functional units, it still depends on the design of parallelism and pipelining. This extreme example that using more functional units does not necessarily promise a fast design is also demonstrated in the

\[ N = 4, L = 10, \text{scalable} \]

\[ N = 4, L = 10, \text{merged} \]

\[ N = 2, L = 10, \text{scalable} \]

\[ N = 1, L = 5 \]
“256 Catapult C (2)” row in the table. The scheduling of an architecture with resource constraints in Catapult C is not always straightforward but needs some insightful architectural constraints. In the architecture with only one multiplier, the butterfly unit is utilized sequentially and only one multiplier is multiplexed for the butterfly unit.

For comparison purpose, we also show an example code of the high-throughput streaming style FFT in part two of Algorithm 4. In this eight-point FFT, we designed a template for one-stage butterfly unit. We also designed a FIFO for streaming data in one-stage butterfly unit. All the stages are laid out in a pipelined mode, by using output of stage 5 as the input of stage 4. Thus, very high throughput can be achieved which will be suitable for the OFDM modulation application.

6.3.2. Design space exploration of MIMO-FFT

The MIMO-FFT/IFFT design in the MIMO chip equalizer architecture shown in Figure 8 is another example of using Catapult C to search for efficient architecture with minimum-resource block mode design. For the multiple FFTs in the tap solver, the keys for optimization of the area/speed are loop unrolling, pipelining, and resource multiplexing. It is not easy to apply the commonality by using the Xilinx IP core for the MIMO-FFTs. To achieve the best area/time tradeoff in different situations, we apply Catapult C to design customized FFT/IFFT modules. We design the merged MIMO-FFT modules to utilize the commonality in control logic and phase coefficient loading. By using merged butterfly unit for MIMO-FFT, we utilize the commonality and achieve much more efficient resource utilization while still meeting the speed requirement. The Catapult C-scheduled RTLs for 32-point FFTs with 16 bits are compared with Xilinx v32FFT Core in Table 3 for a single FFT. Catapult C design demonstrates much smaller size for different solutions, for example, from solution 1 with 8 multipliers and 535 slices to solution 3 with only one multiplier and 551 slices. Overall, solution 3 represents the smallest design with slower but acceptable speed for a single FFT.

For the MIMO-FFT/IFFT modules, we can design a fully parallel and pipelined architecture with parallel butterfly units and complex multipliers laid out in a fully pipelined butterfly tree at one extreme; or we can just reuse one FFT module in serial computation at another extreme. In a parallel layout for an example of 4 FFTs, all the computations are localized and the latency is the same as one single FFT. However, the resource is ×4 of a single FFT module. For a reused module, extra control logic needs to be designed for the multiplexing. The time is equal to or larger than that ×4 of the single FFT computation. However, we can reuse the control logic inside the FFT module and schedule the number of FUs more efficiently in the merged mode. The specifications for 4 merged FFTs are listed in Table 4 with different numbers of multipliers. Compared to 4 parallel FFT blocks (each with 1 MULT) at 2204 slices and 810 cycles or 4 serial FFT at 3240 cycles, the resource utilization is much more efficient, where FU utilization is defined as # Multipliers/(# Cycles * # Multiplications).

The design space for different numbers of merged FFT modules is shown in Figure 21. Figure 21 shows the CLB consumption for different architectures versus the different number of multipliers. For the input and output arrays, two different types of memory mapping schemes are explored. One scheme applies split subblock memories for each input array labelled as SM. This option requires more memory I/O ports but increases the data bandwidth. Another option is a merged memory bank to reduce the data bus. However, the data access bandwidth is limited because of the merged memory block. This demonstrates the design space exploration capability enabled by the Catapult C methodology.

6.4. Scheduling control-dominated architectures

The QRD-M algorithm is a control-dominated architecture because it contains many conditional branches which are extremely difficult with the conventional manual design. The sorting procedure also leads to unpredictable latency depending on the input sequence. Catapult C can synthesize the complex FSM automatically for these types of complex logic. Moreover, it is easy to verify different pipelining tradeoffs. We studied three major different algorithms for the sorting function, each with many partitioning and storage mapping options. The partial sorting C design is described by a flow diagram as shown in Figure 22. To support the high modulation order and large M, the updated metrics are first stored.
in an input buffer memory block. The following inputs and parameters are defined: \( S[N] \) is the input sequence stored in memory blocks; \( I[N] \) is the index sequence initialized to be \( I(i) = i \); \( M \) is the partial factor; \( N = MC \) is the sequence length; \( istack[NSTACK] \) is the stack to store the left and right pointers.

Once the metric update process is ready, it sends a “start” signal to the \( PQSort \) procedure. The partial quick sort has the same concept as the conventional quick sort based on the “partition-exchange” method. First, a “partitioning element” \( a \) is selected from the subsequence. A pair of pointers \( il \) and \( ir \) are defined to set the boundary in terms of “left”
Figure 21: CLB versus number of multipliers for the different architectures of merged MIMO-FFT module: (a) splitted MEM vector processor MIMO-FFT; (b) MEM bank vector processor.

Table 2: Specifications comparison for different solutions of FFT.

| Solution               | BOM | Area (LUT) | Latency (cycles) |
|------------------------|-----|------------|------------------|
| 256 Core               | 12 mult | 3286      | 768              |
| 1024 Core              | 12 mult | 3858      | 4096             |
| 256 Catapult (1)       | 1 m + 1 a + 1 s | 827      | 2076             |
| 256 Catapult (2)       | 4 m + 2 a + 2 s | 1940    | 2387             |
| 1024 Catapult          | 1 m + 1 a + 1 s | 1135    | 9381             |

Table 3: Architecture efficiency comparison.

| Architecture       | MULT | Latency (cycles) | Slices |
|--------------------|------|------------------|--------|
| Xilinx Core        | 12   | 128              | 2066   |
| Catapult C Sol1    | 8    | 570              | 535    |
| Catapult C Sol2    | 2    | 625              | 543    |
| Catapult C Sol3    | 1    | 810              | 551    |

Table 4: Design space exploration for 4 merged 32-point FFTs.

| MULT | Cycles | Slices | Util. | $f_{clk}$ (MHz) |
|------|--------|--------|-------|-----------------|
| 16   | 970    | 570    | 1/7   | 60              |
| 4    | 820    | 810    | 16/40 | 60              |
| 2    | 720    | 1135   | 16/28 | 60              |
| 1    | 680    | 1785   | 16/22 | 60              |

and “right” sides of the subsequence. A stack $i \text{stack}$ with the length of $M \text{STACK}$ is allocated to store the intermediate $i l$ and $i r$ pointers of the pending subsequences. For the first stage, the subsequence is only the full sequence. So $i l = 0$, $i r = N - 1$, and the top pointer of the stack $j \text{stack} = 0$. For a subsequence, when the length is shorter than some size $LQS$, it is faster to use the straight insert sort. To avoid the “worst-case” running time in the quick sort, which usually happens when the input sequence is already in order, the median of the first, middle, and last elements of the current subsequence is used as the “partitioning element.” The partitioning process is carried out in the “do-while” loops for indices $i$ and $j$. For the selected partitioning element $a$, we first scan the $i$ pointer up until we find an element $> a$. Since we do not care about the order of the subsequences larger than $M$, we only need to push the “left” and “right” pointers lower than $M$ to the stack. These pointers popped out from the stack only when the subsequence is short enough for the final “insert-sort” procedure. This not only reduces the size of the stack, but also reduces the latency in processing the subsequences higher than $M$.

For the QRD-M in the MIMO-OFDM system, the runtime comparison of the original and FPGA implementation for the $4 \times 4$ MIMO configuration and 64-QAM modulation is shown in Figure 23. We implemented 2 PEs in the V3000 FPGA in this case. For 64-QAM and $M = 64$, speedup of $\times 100$ is observed with 33 MHz FPGA clock rate competing
with the 1.5 GHz Pentium 4 clock rate. Faster acceleration is achievable using more processing elements with the scalable VLSI architecture and clock rate from P and R result can be up to 90 MHz.

6.5. **Strength and limitations**

As we have shown, the Catapult C HLS methodology demonstrates significant advantages in rapid architecture scheduling and capability to allow extensive design space exploration for customized IP core designs of high-complexity signal processing algorithms. Table 5 compares the productivity of the conventional HDL-based manual design method and the Catapult C-based design space exploration methodology. For the manual design method, we assume that the algorithmic specification is ready and there is some reference design either in Matlab or C code as a baseline source code. The workloads are estimated based on the authors’ extensive experience in using the conventional HDL design. Such estimate is commonly used for project planning by assuming average proficiency in the programming languages. For the Catapult C design, we assume that the fixed-point C/C++ code has been tested in a C test bench using test vectors. The work load does not include the integration stage either within HDL designer or writing some high-level wrapper in VHDL. For the Catapult C design flow, there are possibly many rounds of edit in the C source code to reflect different architecture specifications. It is shown that with the manual VHDL design, it may take much longer design cycle to generate one working architecture than the extensive tradeoff exploration using Catapult C. The improvement in productivity for our prototyping experience of 3G and beyond systems is obvious compared with the conventional HDL-based design methodology.

However, we also notice that there are still some aspects that the Catapult C methodology can improve. Firstly, in terms of the architecture scheduling capability, the efficiency for reusing multiplexers for a large design can be improved from our experience. Moreover, it is still difficult to predict the hardware architectures for control-dominated signal processing algorithms. Of course, this is also partly due to the fact that the control-dominated algorithms usually do not have structures which facilitate architecture pipelining. Secondly, Catapult C still has limitations in system-level simulation with fixed-point analysis. The fixed-point conversion still requires much manual work. Some Matlab-level synthesis tools such as the AccelChip might offer better integrated environment for extensive fixed-point analysis. Thirdly, there are many standard well-defined modules such as the FIR, FFT, besides the many advanced proprietary algorithms for

![Figure 22: The logic flow of the partial quick-sort procedure.](image-url)
an prototyping project. It would be helpful if Catapult C could provide an extensive library for these standard DSP modules. System generator and AccelChip have stronger positions in this IP library feature. However, this IP integration feature should be considered independently with the HLS scheduling capability and the discussion is out of the scope of this paper.

7. CONCLUSION

In this paper, we present our industrial experiences for the 3G/4G wireless systems using a Catapult C HLS rapid prototyping methodology. We discuss core system design issues and propose reduced-complexity algorithms and architectures for the high-complexity receiver algorithms in 3G/4G wireless systems, namely MIMO-CDMA and MIMO-OFDM systems. We also demonstrate how Catapult C enables architecture scheduling and SoC design space exploration of these different classes of receiver algorithms. Different code design styles are demonstrated for different application scenarios. By efficiently studying FPGA architecture tradeoffs, extensive architectural research for high-complexity signal processing algorithms of 3G/4G wireless systems is enabled with significantly improved productivity.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Behnaam Aazhang and Gang Xu for their support in this work. J. R. Cavallaro was supported in part by NSF under Grants ANI-9979465, EIA-0224458, and EIA-0321266. Part of the paper was presented in IEEE RSP’03 and Asilomar’04 conferences.

REFERENCES

[1] A. Wiesel, L. Garcia, J. Vidal, A. Pagès, and J. R. Fonollosa, “Turbo linear dispersion space time coding for MIMO HS-DPA systems,” in Proceedings of 12th IST Summit on Mobile and Wireless Communications, Aveiro, Portugal, June 2003.
[2] G. D. Golden, C. J. Foschini, R. A. Valenzuela, and P. W. Wolniansky, “Detection algorithm and initial laboratory results using V-BLAST space-time communication architecture,” Electronics Letters, vol. 35, no. 1, pp. 14–16, 1999.
[3] G. J. Foschini, “Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas,” Bell Labs Technical Journal, vol. 1, no. 2, pp. 41–59, 1996.
[4] Y. Guo, J. Zhang, D. McCain, and J. R. Cavallaro, “Efficient MIMO equalization for downlink multi-code CDMA: Complexity optimization and comparative study,” in Proceedings of IEEE Global Telecommunications Conference (GLOBECOM ’04), vol. 4, pp. 2513–2519, Dallas, Tex, USA, November-December 2004.
[5] J. Yue, K. J. Kim, J. D. Gibson, and R. A. Itis, “Channel estimation and data detection for MIMO-OFDM systems,” in Proceedings of IEEE Global Telecommunications Conference (GLOBECOM ’03), vol. 2, pp. 581–585, San Francisco, Calif, USA, December 2003.
[6] Y. Lee and V. K. Jain, “VLSI architecture for an advance DS/CDMA wireless communication receiver,” in Proceedings of the Annual IEEE International Conference on Innovative Systems in Silicon, pp. 237–247, Austin, Tex, USA, October 1997.

[7] Z. Guo and P. Nilsson, “An ASIC implementation for V-BLAST detection in 0.35 µm CMOS,” in Proceedings of the 4th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT ’04), pp. 95–98, Rome, Italy, December 2004.

[8] A. Adjoudani, E. C. Beck, A. P. Burg, et al., “Prototype experience for MIMO BLAST over third-generation wireless system,” IEEE Journal on Selected Areas in Communications, vol. 21, no. 3, pp. 440–451, 2003.

[9] B. Razavi, RF Microelectronics, Prentice Hall Communications Engineering and Emerging Technologies Series, Prentice-Hall, Upper Saddle River, NJ, USA, 1998.

[10] K. Hooli, M. Juntti, M. J. Heikkila, P. Komulainen, M. Latva-Aho, and J. Lilleberg, “Chip-level channel equalization in WCDMA downlink,” EURASIP Journal on Applied Signal Processing, vol. 2002, no. 8, pp. 757–770, 2002.

[11] J. M. Rabaey, “Low-power silicon architecture for wireless communications: embedded tutorial,” in Proceedings of ASP-DAC 2000, Asia and South Pacific Design Automation Conference, pp. 377–380, Yokohama, Japan, January 2000.

[12] A. Evans, A. Silburt, G. Vrckovnik, et al., “Functional verification of large ASICs,” in Proceedings of 35th ACM/IEEE Design Automation Conference (DAC ’98), pp. 650–655, San Francisco, Calif, USA, June 1998.

[13] J. Blasker, A VHDL Primer, Prentice-Hall, Upper Saddle River, NJ, USA, 3rd edition, 1999.

[14] R. Camposano and W. Wolf, Trends in High-Level Synthesis, Kluwer, Boston, Mass, USA, 1991.

[15] G. De Micheli and D. C. Ku, “HERCULES - a system for high-level synthesis,” in Proceedings of the 25th ACM/IEEE Conference on Design Automation (DAC ’88), pp. 483–488, Anaheim, Calif, USA, June 1988.

[16] C.-Y. Wang and K. K. Parhi, “High-level DSP synthesis using concurrent transformations, scheduling, and allocation,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 14, no. 3, pp. 274–295, 1995.

[17] http://www.systeme.org/.

[18] http://www.doc.ic.ac.uk/~akf/handel-c/cgi-bin/forum.cgi.

[19] D. Knapp, T. Ly, D. MacMillen, and R. Miller, “Behaviorial synthesis methodology for HDL-based specification and validation,” in Proceedings of 32nd ACM/IEEE Design Automation Conference (DAC ’95), pp. 286–291, San Francisco, Calif, USA, June 1995.

[20] J.-W. Weijers, V. Derudder, S. Janssens, F. Petré, and A. Bourdoux, “From MIMO-OFDM algorithms to a real-time wireless prototype: a systematic matlab-to-hardware design flow,” EURASIP Journal on Applied Signal Processing, vol. 2006, Article ID 39297, 12 pages, 2006.

[21] Catapult C Manual and C/C++ style guide, Mentor Graphics, 2004.

[22] U. Knippin, “Early design evaluation in hardware and system prototyping for concurrent hardware/software validation in one environment,” in Proceedings of 13th IEEE International Workshop on Rapid System Prototyping (RSP ’02), Aiptix, Darmstadt, Germany, July 2002.

[23] Y. Guo and D. McCain, “Compact hardware accelerator for functional verification and rapid prototyping of 4G wireless communication systems,” in Proceedings of 38th IEEE Asilomar Conference on Signals, Systems and Computers, vol. 1, pp. 767–771, Pacific Grove, Calif, USA, November 2004.

[24] T. M. Aulin, “Breadth-first maximum likelihood sequence detection: basics,” IEEE Transactions on Communications, vol. 47, no. 2, pp. 208–216, 1999.

[25] G. H. Golub and C. F. V. Loan, Matrix Computations, The Jones Hopkins University Press, Baltimore, Md, USA, 1996.

[26] M. Rupp, A. Burg, and E. Beck, “Rapid prototyping for wireless designs: the five-ones approach,” Signal Processing, vol. 83, no. 7, pp. 1427–1444, 2003.

[27] H. Steendam and M. Moeneclaey, “The effect of clock frequency offsets on downlink MC-DS-CDMA,” in Proceedings of IEEE International Symposium on Spread Spectrum Techniques and Applications (ISSSTA ’02), vol. 1, pp. 113–117, Prague, Czech Republic, September 2002.

Yuanbin Guo received the B.S. degree (E.E.) from Peking University, and the M.S. (E.E.) degree from Beijing University of Posts and Telecommunications, Beijing, China, in 1996 and 1999, respectively, and the Ph.D. degree from Rice University, Houston, Tex, in May 2005, in electrical and computer engineering. He was a winner of the Presidential Fellowship in Rice University in 2000. From 1999 to 2000, he was with Lucent Bell Laboratories, Beijing, where he conducted R&D in the Intelligent Network Department. He joined Nokia Research Center, Irving, Tex, in 2002. He is now a Senior Research Engineer and Research Specialist in the Signal Processing Architecture Group of Nokia Networks Strategy and Technology. His current research interests include equalization and detection for multiple-antenna systems, VLSI design and prototyping, and DSP and VLSI architectures for wireless systems, 3GPP long-term evolution (LTE), OFDM, WiMax. He is a Member of IEEE. He has 6 patents pending in wireless communications field.

Dennis McCain received his B.S. degree in electrical engineering from Louisiana State University in 1990 and his M.S. degree in electrical engineering from Texas A&M University in 1992. From 1992 to 1996, he served in the US Army as a Signal Officer responsible for deploying communication networks in tactical environments. From 1996 to 1998, he worked at Texas Instruments and Raytheon Systems as a Digital Design Engineer. In 1999, he joined Nokia Research Center in Dallas, Tex, where he developed prototype wireless communication systems. He is currently a Technology Manager in Nokia Networks Strategy and Technology leading a team responsible for implementing novel physical layer algorithms for next-generation cellular wireless systems. His interests are in the areas of hardware architecture research, digital baseband design, and rapid prototype design flows.

Joseph R. Cavallaro received the B.S. degree from the University of Pennsylvania, Philadelphia, Pa, in 1981, the M.S. degree from Princeton University, Princeton, NJ, in 1982, and the Ph.D. degree from Cornell University, Ithaca, NY, in 1988, all in electrical engineering. From 1981 to 1983, he was with AT&T Bell Laboratories, Holmdel, NJ. In 1988, he joined the faculty of Rice
University, Houston, Tex, where he is currently a Professor of electrical and computer engineering. His research interests include computer arithmetic, VLSI design and microlithography, and DSP and VLSI architectures for applications in wireless communications. During the 1996–1997 academic year, he served at the US National Science Foundation as Director of the Prototyping Tools and Methodology Program. During 2005, he was a Nokia Foundation Fellow and a Visiting Professor at the University of Oulu, Finland. He is currently the Associate Director of the Center for Multimedia Communication at Rice University. He is a Senior Member of the IEEE. He was the Cochair of the 2004 Signal Processing for Communications Symposium at the IEEE Global Communications Conference and general Cochair of the 2004 IEEE 15th International Conference on Application-Specific Systems, Architectures, and Processors (ASAP).

Andres Takach is a Chief Scientist, C-Based Design at Mentor Graphics. He joined Mentor Graphics in 1997, where he has worked on all aspects of high-level synthesis. His fields of interest are in high-level synthesis, synthesis for low-power, embedded system design, and hardware/software codeign. From 1993 to 1997, he was a faculty member at Illinois Institute of Technology, where he conducted research in high-level synthesis and hardware/software codesign. Andres Takach received his Ph.D. degree from Princeton University in 1993 and his B.S. and M.S. degrees in electrical and computer engineering from the University of Wisconsin-Madison in 1986 and 1988, respectively.
Generation of Embedded Hardware/Software from SystemC

Salim Ouadjaout and Dominique Houzet

Institut d’Electronique et de Télécommunications de Rennes (IETR), UMR CNRS 6164, Institut National des Sciences Appliquées (INSA), 20 avenue des Buttes de Coësmes, 35043 Rennes Cedex, France

Received 30 November 2005; Revised 26 June 2006; Accepted 27 June 2006

Designers increasingly rely on reusing intellectual property (IP) and on raising the level of abstraction to respect system-on-chip (SoC) market characteristics. However, most hardware and embedded software codes are recoded manually from system level. This recoding step often results in new coding errors that must be identified and debugged. Thus, shorter time-to-market requires automation of the system synthesis from high-level specifications. In this paper, we propose a design flow intended to reduce the SoC design cost. This design flow unifies hardware and software using a single high-level language. It integrates hardware/software (HW/SW) generation tools and an automatic interface synthesis through a custom library of adapters. We have validated our interface synthesis approach on a hardware producer/consumer case study and on the design of a given software radiocommunication application.

Copyright © 2006 S. Ouadjaout and D. Houzet. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. INTRODUCTION

Technological evolution—particularly shrinking silicon fabrication geometries—enables the integration of complex platforms in a single system on chip (SoC). In addition to specific hardware subsystems, a modern SoC can also include sophisticated interconnects and one or several CPU subsystems to execute software. New design flows for SoC design have become essential in order to manage the system complexity in a short time-to-market. These flows include hardware/software (HW/SW) generation tools, the reuse of pre-designed intellectual property (IP), and interface synthesis methodologies which are still open problems requiring further research activities [1].

EDA tools propose their own solutions to HW/SW generation. Some use SystemC as a starting point for the hardware design, like Cynthesizer from Forte Design [2] or Agility Compiler from Celoxica [3]. Several tools use the C language as a starting point for both hardware and software with a custom application programming interface (API) for HW/SW interfaces. It is the case of DK Design Suite from Celoxica [3] with its DSM API and CatapultC from Mentor [4]. In SiliconC [5], structural VHDL is generated for the C functions. Prototypes of the functions become the entities. There are other variants which start from Matlab to produce both hardware and software like SPW from CoWare [6]. Many design methodologies exist for the design of embedded software [7–9]. Some are based on code generated from an abstract model (UML [10]), graphical finite state machine design environments (e.g., StateCharts [11]), DSP graphical programming environments (e.g., Ptolemy [8]), or from synchronous programming languages (e.g., Esterel [12]). A software generation from a high-level model of operating system is proposed by several authors [13–16]. In [15], a software generation from SystemC is based on the redefinition and overloading of SystemC class library elements. In [13], a software-software communication synthesis approach by substituting each SystemC module with an equivalent C structure is proposed. It requires special SystemC modeling styles (i.e., with macrodefinitions and preprocessing switches in addition to the original specification code). In [16], software is generated from SpecC with no restrictions on the description of the system model.

Several approaches have been developed to deal with IP integration. Fast prototyping enables the productive reuse of IPs [17]. It describes how to use an innovative system design flow that combines different technologies, such as C modelling, emulation, hard virtual component reuse, and CoWare tools [6]. Prosilog’s IP creator, as part of Magillem, aims to improve the integration and reuse of non-VCI compliant IPs by wrapping them into a compatible structure. This tool allows the generation of wrappers from a RTL VHDL description of the IP interface [18]. The Cosy approach is based on the infrastructure and concepts developed...
in the VCC framework [19]; it defines interfaces at multiple levels of abstraction. Most of those approaches deal with low-level protocol adaptation in order to integrate RTL, level IPs. A few approaches provide a ready network on Chip (NoC) to allow easy integration of communication. But these approaches require that the IPs have to be compliant with the NoC interface. Consequently, the designers have to modify the IPs codes. All these approaches deal with system-level synthesis which is widely considered as the solution for closing the productivity gap in system design. System-level models are developed for early design exploration. The system specification of an embedded system is made of a hierarchical set of modules (or processes) interconnected by channels. They are described in a system-level language as a set of behaviour, channel, and interface declarations. Those behaviours mapped onto general or application-specific microprocessors are then implemented as embedded software and hardware. The predominant system-level languages are C/C++ extensions [13, 20]. We consider here the SystemC language but another language can be used. SystemC is mainly used to model and to simulate designs at system level. However, dedicated powerful hardware description languages like VHDL and Verilog are used for RTL. Embedded software languages like C with static scheduling or POSIX RTOS are used for embedded processors. This leads to a decoupling of behavioral descriptions and implementable descriptions. This decoupling usually requires the recoding of the design from its specification simulation in order to meet the very different requirements of the final generated code. The recoding step often results in new coding errors that must be identified and debugged. The derivation of embedded software and hardware from system specifications described in a system-level language requires to implement all language elements (e.g., modules, processes, channels, and port mappings). It is known that SystemC allows the refinement for hardware synthesis, but up to now SystemC has not been used as an embedded software language. Considering the limited memory space and execution power of embedded processors, the SystemC overhead makes the direct compilation to produce the binary code for target embedded microprocessors highly inefficient. Obviously, it is due to the large SystemC kernel included in the compiled code. This kernel introduces an overhead to support the system-level features (e.g., hierarchy, concurrency, communication), but these features are not necessary to the target embedded software code. In addition to direct SystemC compilation inefficiency, some cross-compilers for embedded processors may only support the C language. Thus, SystemC has to be translated to C code.

To address system-level synthesis, we propose in this paper a top-down methodology. Our challenge is to automate the codesign flow generating the final code for both embedded processors and hardware from a unifying high-level language (SystemC). In our methodology, we have developed methods to make the codesign flow smooth, efficient, and automated. These methods allow two improvements: a rapid integration of communication and a fast software generation for embedded processors with an efficient interface synthesis. The proposed methodology includes several parsing steps and intermediate models. The first main step is the communication integration based on a custom library of interface adapters that uses the virtual component interface (VCI) standard from VSIA consortium [21]. This library aims to perform the interface synthesis. It allows heterogeneous IPs to communicate in a plug-and-play fashion in the same system. The second main step is the generation of embedded C code from the system specification written in SystemC. Our approach proposes the use of static scheduling and POSIX-based RTOS models. It enables also an automatic refinement, while [14] requires its own proprietary simulation engine and needs manual refinement to get the software code. Our method also differs from [13–16] in that our high-level SystemC code is translated to a C code with optimized interface synthesis. Optimization is performed according to the processors busses and the NoC as well as according to the SystemC parallel programming model. Other recent propositions have been published in that direction [22].

The paper is organized as follow. In Section 2 we describe the main features of our proposed design flow. The main innovative parts of the design flow are detailed in the next two sections. The first one presents our hardware interface library and our integration methodology of functional IPs, with implementation results from a simple design example. The second one describes the translation process of SystemC elements to C code. This C code targets either an RTOS for dynamic scheduling or a stand alone solution with a generated static scheduling. This translation process is validated in Section 5 with implementation results of a producer/consumer and a code-division multiple-access (CDMA) radio-communication applications. This work is the result of a project started in 2001 [23–25].

2. DESIGN FLOW

SoC design requires the elaboration and the use of radically new design methodologies. The main parts of a typical system-level design flow are the specification model, the partition into HW/SW elements, and the implementation of the models for each element. In Figure 1 we describe the proposed top-down methodology of automatic generation of binary files from SystemC to both embedded software and hardware. The design flow starts with a high-level model described in a high-level programming language (SystemC). The system is described either through direct programming or through IP reuse. We use Celoxica tools to develop, simulate, analyze, and validate the SystemC code (step (1)). The first SystemC description is at the functional level. The system is a set of functional IPs including functional models of architectural IPs for fast simulation. The communication between IPs uses SystemC channel mechanisms like sc_signal or sc_fifo with read() and write() primitive functions. From the Celoxica graphical tool, we select the IPs which are associated with the hardware side (the architectural IPs substituted by their already VCI-compliant version), and the IPs which are associated with the software side (the monitoring IPs, stimulating IPs, host IPs, etc.). The remaining IPs of the
system are targeted to the codesign side, as we need to optimize and well balance hardware and embedded software to meet several stringent design constraints simultaneously: hard real-time performance, low power consumption, and low resources.

Considering the software side (step (2)), the SystemC IPs are directly compiled to become binary files targeted to the host processor. This set of software tasks communicates with the remaining IPs contained in the FPGA platform through the PCI bus. Because software components run on processors, the SystemC abstract communication needed to describe the interconnection between the software and hardware components is totally different from the existing abstraction of wires between hardware components as well as the function calls abstraction that describes the software communication.

In this part, the communication is abstracted as an API which calls PCI bus drivers through an operating system layer. The API hides hardware details such as interrupt controllers or memory and input/output subsystems. We have implemented the message passing interface (MPI-2) library on the host processor and on the embedded processors of our platform [26]. MPI-2 is our HW/SW interface API.

Step (3) is the performing of our SCXML parser tool which allows to convert a given SystemC source code into an XML intermediate representation. The XML format is a subset of the standardized SPIRIT 2.0 format [27]. The system is interpreted as a set of XML files. Each XML file contains the most important characteristics of a SystemC IP, such as

(i) name, type, and size of each in/out ports, name and type of processes declared in the constructor, and also the sensitivity list of each process;
(ii) name and type of IPs building a hierarchical IP, the names of connections between the sub-IPs, and the binding with the IP ports.

Both XML files and profiling reports from Celoxica tool are treated by our HW/SW partitioning tool (step (4)) in order to partition IPs as hardware or software according to the architecture parameters and constraints. After this step we use SynDEx tool (step (5)) to perform an automatic mapping, routing, and static scheduling of IPs on the software and hardware architecture based on a predefined NoC topology [28]. The different SynDEx inputs are the following.

(i) A hierarchical conditioned data-flow graph of computing operations and input/output operations. The operations are just specified by the type and size of input/output data and execution time of the IPs. The XML files and profiling reports are parsed to produce these inputs. We need also to provide manually information on the nonexclusive execution of IPs in order to help SyndEx optimize parallelism.
Figure 2: VCI connections of non-VCI IPs through VCI adapters. (a) "Wire" point-to-point connection. (b) NoC connection.

(ii) Specification of the heterogeneous architecture as a graph composed of software processors and hardware processors, interconnected through communication medias. Processors characteristics are supported tasks, their execution duration, worst-case transfer duration for each type of data on the interconnect. The profiling reports and architecture parameters are parsed to produce these inputs.

SynDEx implements the IPs onto the multicomponent architecture through a heuristic mapping, routing, and scheduling. After the implementation, a timing diagram gives the mapping of the different IPs on the components and the real-time predicted behavior of the system. The communication links are represented in order to show all the exchanges between processors; they are taken into account in the execution time of IPs. The mapping/routing code generated by SynDEx tool is then parsed (step (6)) in order to manage the NoC configuration and to switch software IPs to the XML/C parser. This parser translates the XML markups to C code with either RTOS calls or a static scheduling provided by SynDEx tool. With our SCXML and XML/C parsers, we obtain an embedded C generation tool (SCEmbed) from SystemC. This SCEmbed tool has about 5000 C++ and JAVA code lines. This tool and its XML format can be easily adapted to a different RTOS.

The embedded C code is then treated in step (7) with the Gcc compiler in order to obtain binary executables for the embedded processors. As the C software IPs are mapped on several heterogeneous processors, they need to use a communication library (MPI-2).

In the communication integration (step (8)), the identified SystemC hardware IPs are completed with our SystemC VCI adapter library. This point is detailed in the section below. Then point-to-point communications are established between the new VCI-compliant IPs and the VCI hardware IPs through the VCI NoC. We use SynDEx configuration information to initialize the VCI adapters, plug the IPs on the NoC, and load the binary code of the software IPs on their corresponding processor memory. Once all the SystemC architecture is produced, we can either simulate it back in the Celoxica tool for evaluation. After validation, we continue with the implementation step.

The last hardware synthesis step plays a very important role in the methodology described above. There have been various research efforts to come up with a good hardware compiler which can generate a synthesizable HDL from high-level C/SystemC specifications. The Agility Compiler from Celoxica can help the generation of synthesizable VHDL from SystemC. The final product of the design flow is a set of binary files representing programs for the host processor, LEON and Microblaze (Xilinx) processors and FPGAs. These files can be loaded onto the respective components of the prototyping platform (FPGA boards) to build a prototype with a real-time communication system.

3. HW/SW INTERFACE CODESIGN

3.1. Introduction

An SoC can include specific hardware subsystems and one or several CPU subsystems to execute the software tasks. The SoC architecture includes hardware adapters (bridges or communication coprocessors) to connect the CPU subsystems to other subsystems. The HW/SW interface abstraction must hide the CPU. On the software side, the abstraction hides the CPU under a low-level software layer ranging from basic drivers and I/O functionality to sophisticated operating system. On the hardware side, the interface abstraction hides CPU bus details through a hardware abstraction layer generally called the CPU interface. This can range from simple registers to sophisticated I/O peripherals including direct memory access queues and complex data conversion and buffering systems.

3.2. Hardware-to-hardware interface synthesis: VCI adaptation methodology

We show in Figure 2 the way to establish a communication between IPs with different abstraction levels. We consider here functional IPs and architectural IPs.

The connection can be through wires or through an NoC. The VCI adapters library aims to simplify the (re)use of functional IPs (non-VCI compliant) in any SoC based on the VCI protocol. This adapter library is designed in order to change neither the IP cores nor their interface description.
The generic architecture shown in Figure 3 helps to clarify the relationship between two hardware IPs connected through a sophisticated VCI NoC. The communication between heterogeneous component interfaces imposes the existence of a wrapper on each side of the communication media (bus or NoC). This wrapper behaves like a bridge which translates the RTL interface between the media and the component. These wrappers (agents) have to be compatible with VCI interface to build a standard media. Thus, an initiator wrapper is connected to VCI initiator ports of a master IP and a target wrapper is connected to VCI target ports of a slave IP.

Considering that these two VCI wrappers are available, the interface synthesis of SystemC functional IPs is a set of steps to replace a primitive channel with a refined channel in order to connect it to the wrappers. A refined channel will often have a more complex interface (e.g., VCI) than the primitive channel previously used. The main step in refining the interfaces is to create adapters that connect the original modules to the refined channel. Adapters can help to convert the interfaces of the IPs instances into VCI interfaces. The interface refinement can be made more manageable if new interfaces are developed without making changes to their associated module. The adapter translates the transaction-oriented interface consisting of methods such as `write(data)` into VCI RTL-level interface for hardware IPs. Figure 3 depicts the use of adapters to connect functional IPs to the NoC VCI agents. Hook arrow boxes indicate the interface provided by the adapters while the rightleftarrows square boxes represent ports. Our contribution consists in the design of VCI master adapters and VCI slave adapters which manage the VCI initiator and VCI target interfaces, respectively. We have chosen a convention that each SystemC output port is an initiating port of transaction and each input port is a target port. Thus, the release of a transaction results in a nonblocking write of data on the output port for an `sc_signal` and in a blocking write for an `sc_fifo`. This corresponds to the semantics of the SystemC `sc_signal` and `sc_fifo` primitive channels. Thus, initiating ports of functional IPs are connected to a master adapter and target ports are connected to a slave adapter. In this case, several IPs may be connected to the same adapter.

The adaptation methodology approach is implemented using a micronetwork stack paradigm, which is an adaptation of the OSI protocol stack.

### 3.2.1. Application layer

This layer describes the functional behaviour of a complex system. A system is a set of functional IPs with behavioural models, not architectural IPs such as processors or memories. The communication mechanism is performed with classical `read(data)` and `write(data)` SystemC primitives without additional parameters and no protocol implementation.

### 3.2.2. VCI adapter layer

The VCI adapter layer is responsible for converting an IP interface towards a lower-level interface. A VCI adapter core can manage different ports of different non-VCI-compliant IPs. Functional hardware IP ports are implemented as a memory segment accessed through its VCI adapter. The VCI adapter layer is composed of the following sublayers.

(a) **Presentation layer**

This layer is responsible for translating an abstract data-type port towards a SystemC synthesizable data-type port.

(b) **Session layer**

The session layer generates a single VCI address between two ports connected to each other in the system-level description. This address is divided into two fields: the most significant bits (MSB) identify the destination wrapper and the least significant bits (LSB) identify the local offset at destination. Each agent of the NoC needs to be configured in order to know the separation position between MSB and LSB, and thus be able to perform address translation to correctly route the data to be sent.

The LSB field is itself divided first according to the target IP port addressed among the different IP ports connected to the same VCI adapter, and second according to the local address segment managed by the transport layer. VCI adapter address is finally divided into three fields.

(i) Field 1: agent number is the address field decoded/generated by the NoC agents and routed in the NoC.

(ii) Field 2: port number is the address field decoded/generated by the VCI adapter to switch data to the corresponding IP port.

(iii) Field 3: word number is the address field decoded/generated by the transport layer. It represents the address in the memory segment of the selected port.

The address translation of each VCI adapter is configured during its connection to the NoC with its NoC agent number and its port number. Already VCI-compliant IPs have to provide configurability of addresses in order to communicate to any IP on the NoC. This configuration of IP VCI adapters is performed during VCI adapter integration step based on SynDEx mapping/routing information. For already VCI-compliant IPs, addresses are provided manually as it is IP dependant. This is the second of the very few non-fully automated parts of the flow.
(c) **Transport layer**

The basic function of the transport layer is multiple: it accepts data from the IP ports, splits them into smaller units (segments) according to the VCI master adapter data bus size, passes them to the network layer, and ensures that the pieces all arrive correctly at the other end. In addition, the transport layer is responsible for the generation of the segment number which constitutes the third field of VCI address.

(d) **Network layer**

This layer is responsible for the identification of the initiating port. In the case of a multiport master adapter, the network layer launches an arbiter to solve the conflicts and ensures that only one port can have an access to the resource (media). The second treatment is the operation of transfers multiplexing and demultiplexing.

(e) **Datalink layer**

The datalink layer defines the format of data on the interface and the communication protocol. It is responsible for VCI transactions.

3.2.3. **Physical layer**

The physical layer is the physical way of communication. Wires are used for point-to-point connection between VCs. An NoC is used for sophisticated communications.

We have synthesized an example of a simple producer/consumer on the Xilinx FPGA technology. We have used the PVC/VC/VC/VC adapters with an 8-bit data bus and a 5-bit address bus on both IPs. Each adapter unit allows two IP data bus connections of 64-bit and 32-bit size, respectively, with a static IP port priority management. This implementation was performed with Xilinx Virtex II xc2v3000-6 technology. We present here the post placed/routed results. We have obtained a master adapter cost of 489 units of 4-entries logic and 136 flipflop units, with a 100 MHz clock frequency. So, it occupies 1.7% of the FPGA. The slave adapter requires 144 4-entries logic units and 204 flipflop units with the same clock frequency. It needs 0.46% of the FPGA resources. A master adapter is four times larger than a slave adapter.

3.3. **Software-to-software interface synthesis**

For embedded software, the SystemC read(data) and write(data) are implemented with POSIX elements in the case of dynamic scheduling with an RTOS and message passing interface (MPI) elements in the case of static scheduling. We have used the POSIX compliant real-time embedded multiprocessor scheduler (RTEMS) as RTOS.

For RTEMS, the read and write primitive functions are replaced with the rtems_message_queue_receive() function and the rtems_message_queue_send() function, respectively. The sc_fifo blocking read() function is implemented with the RTEMS_WAIT option set in rtems_message_queue_receive(). The nonblocking sc_signal functions are implemented for RTEMS through message queues which are flushed before each data write. The nonblocking read is implemented with the option RTEMS_NO_WAIT.

For an RTOS-less solution, the SystemC read(data) and write(data) are implemented as one-sided remote memory access (RMA) with the MPI_MPI_put(data) primitive only. The blocking mechanism for sc_fifo is implemented with the MPI_wait() primitive which waits for an acknowledgment.

3.4. **Software-to-hardware interface synthesis**

For software IP on embedded CPUs, communication with the NoC VCI agent is managed with dual-ported memory buffers and DMA from its VCI adapter (dedicated to the CPUs) directly connected to this dual-ported memory. The DMA is controlled by software driver subroutines overloading MPI or RTEMS message queues.

In the case of host processor, the read(data) and write(data) SystemC primitives are overloaded in order to call the PCI driver services through MPI calls. This software driver configures the hardware DMA which manages the data transactions between host memory and the NoC on the prototyping board through the VCI/PCI bridge.

Using one-sided RMA is an efficient implementation solution of MPI [26, 29] and the SystemC programming model is also very well suited to RMA implementation as sc_signal reads and writes are not correlated. In practice, efficiency of HW/SW interfaces is obtained with a direct integration of SystemC high-level communication library in hardware, that is, by a joint optimization of the implementation of the SystemC programming model with the MPI_MPI_put() and MPI_MPI_wait() primitives (RMA model) as well as with the underlying NoC design. The RMA mechanism is limited to write-only transfers between IPs allowing the design of a specific NoC optimized for those transfers with DMA. This approach is similar to the joint optimization of compilers and microarchitectures of microprocessors.

We have designed optimized network interfaces for two custom NoC [30] with write-only communications, connected to Microblazes, LEONs, and PowerPCs processors through their dedicated ports. The MPI_MPI_put() primitive needs two I/O access to configure the DMA of the network interface and to launch the DMA transfer in the NoC. Thus the MPI_MPI_put() takes only 8 processor clock cycles: 6 clock cycles to prepare the DMA configuration and 2 clock cycles for I/O access. In that case, the result for the SystemC sc_signal write() primitive is 25 clock cycles of overhead comprising two MPI_MPI_put() executions (one for the control and one for the data), that is, 16 clock cycles, and 9 clock cycles to prepare the data to be transferred. Also there is no overhead for the SystemC sc_signal read() which is only a local variable access due to the RMA mechanism.

For comparison, the main difference between MPI RMA subset and DSM API from Celoxica presented in Table 1 is that the MPI_MPI_put is a nonblocking mechanism which in conjunction with MPI_MPI_wait can implement a blocking
TABLE 1: DSM and RMA MPI subset comparison.

| DSM              | MPI          |
|------------------|--------------|
| DsmInit()        | MPI_Init()   |
| DsmExit()        | MPI_Finalize()|
| DsmWrite() & DsmRead() | MPI_Put() & MPI_Wait() |
| DsmPortS2HOpen() | —            |
| —                | MPI_BARRIER  |

mechanism, compared to the DsmWrite and DsmRead which are only blocking mechanisms. Also the DSM API is a two-sided communication compared to the one-sided RMA subset.

4. GENERATION OF EMBEDDED C CODE

4.1. Generation process

In modern complex SoCs, the software as an integral part of the SoC is gaining more and more importance. At the system level, the system is composed of a set of hierarchical behaviors connected together through channels. However, for the implementation, many designers use a task-based approach, where the tasks are scheduled by a real-time kernel. A whole system design is composed of a set of globally asynchronous/locally synchronous reactive processes that concurrently perform the system functionalities.

Inside the SystemC process code, only wait() primitives are allowed and processes lack a sensitivity list except for one signal which is considered as a clock. Therefore, a process will only block when it reaches a wait(). These restrictions that we have required are only for the code involved in the embedded HW/SW partitioning process. They help our SCEmbed tool to generate the embedded C code [30]. These restrictions on SystemC coding are also required by Celoxica tools for the SystemC synthesis.

The XML format used by the XML/C parser is easily adaptable for a new target RTOS. The main idea behind is to redefine the SystemC class library elements for the new target RTOS. The original code of these IPs calls the SystemC kernel functions to support process concurrency and communication. The new code calls the embedded RTOS functions that implement the equivalent functionality. Thus, SystemC kernel functions are replaced either by typical RTOS functions or through direct generation of a statically scheduled code. The functional behavior is not modified during the hardware, software, and interfaces generation.

We illustrate the C generation process for the RTOS target with a producer/consumer example. The SystemC main code named sc_main() is converted to the RTEMS RTOS main code “init.” The channels are implemented with message queues for blocking sc_fifo channels and shared variables for nonblocking sc_signal channels. The clock in the SystemC code is converted into a task sending an event value broadcasted on a message queue. All the tasks read this clock message queue for there synchronization.

SystemC concurrent processes need to be converted into RTOS-based tasks. We instantiate the child tasks in a parent one corresponding to the SC_MODULE in the system specification. This step is illustrated by our example in Figure 4. The producer and the consumer instances are converted into Tprod and Tcons parent tasks. In RTEMS, each parent task (SC_MODULE in SystemC) launches the child tasks (processes in SystemC) and an additional task which is responsible for interprocess communication. This task is created to manage sc_out ports writing delay corresponding to the behavioral delay of the SystemC write function (the data are validated after the wait event). The RTEMS equivalent code of the SystemC Producer is shown in Figure 5.

At the system level, synchronization is implemented using channels or SystemC events. During the generation process, the RTOS model provides routines to replace the SystemC synchronization primitives.

In the case of POSIX generation, synchronization between tasks is managed by semaphores for sc_signal implementation with global shared variables. A special clock management task is generated which schedules the two-step signal assignment process in order to respect the semantic of sc_signal. All the signal assignments are performed simultaneously after all the processes are stopped on a wait() instruction. The wait() instruction is implemented by a semaphore synchronization. The clock task is waiting for all the tasks which are sensitive to the same clock to stop on a wait instruction. Then the second step is performed by this clock management task, which corresponds to the assignment of all the shared global variables with the temporary variables assigned by the different blocked tasks. These blocked tasks are then freed and can read the shared global variables which are now updated. This mechanism is generated for each independent clock in the whole system. When different tasks are mapped on different processors, we assume that they communicate through asynchronous sc_fifo channels. Otherwise, the clock management tasks of the different processors have to be synchronized before the assignment of the shared global variables.

The second approach uses an RTOS-less static scheduling. In this solution, the SystemC scheduler is replaced by our custom simulation engine optimized for embedded applications. This scheduler is called from each wait() instruction or from sc_fifo blocking read() or write() functions. This scheduler also manages the synchronization of clock sensitive tasks with barrier primitives.

A channel implementation library is provided for all the solutions. Up to now, only primitive channels are available (sc_signal, sc_fifo). There are three versions of implementation for each channel: SW/SW, HW/HW, and SW/HW. SW/SW channels are direct shared variables or message queues implementation. HW/HW channels are RTL-level NoC wrappers. SW/HW are C drivers for embedded processors connected to the NoC.

4.2. Application example

In order to evaluate the proposed technique, two designs have been experimented for the SW part in this section.
The first one is a simple consumer/producer case with two SystemC components linked together. The second one, a more realistic case, is a CDMA radiocommunication example. The consumer/producer system description has about 86 SystemC code lines and the CDMA system description has about 976 SystemC code lines. The CDMA includes 7 modules with 8 concurrent processes. Both examples have been implemented in a SPARC-based platform that includes 1 MB SDRAM and a LEON2 processor synthesized on one 4Mgate Xilinx FPGA with 128 KB of RAM. The open source POSIX-compliant RTEMS operating system has been selected as the target embedded RTOS.

The CDMA system has 7 modules: the top (CDMA), one module that generates samples, three modules that compute the QPSK modulation, the THR and the interleaving, one that models the real environment channel behavior by introducing noise, and the last ones that do the reverse treatment that is deinterleaving, ITHR and demodulation. All the modules work in a pipelined dataflow way. Several channel models have been implemented with our design flow. The CDMA application example uses one of them: a nonblocking channel (the sc_signal channel). The proposed channel models have different implementations depending on the HW/SW partition. Several experiments have been performed with semaphores, mutex condition variables, and signals in order to synchronize threads with RTEMS.

Table 2 shows the code size of the different codes on the different operating systems. Table 3 presents their binary size and Table 4 their average execution time per treatment iteration.
/* ==*/Myproducer.h File=*/

class prod : public sc_module
{
  public:
    sc_out<uchar> out;
    sc_in<bool> clk;

  int i;

  void main();

  SC_HAS_PROCESS(prod);

  prod(···): sc_module(name){
    SC_THREAD(main);
    sensitive_pos ≪ clk;
  }

};

rtems_task prod(rtems_task_argument * port) {
  Tsend = rtems_build_name("f","c","o","m");
  Tmain = rtems_build_name("m","a","i","n");

  rtems_task_create(Tsend[0], ... , & TsendID);
  rtems_task_create(Tmain[1], ... , & TmainID);
  rtems_task_start(TsendID, ComTask, & port);
  rtems_task_start(TmainID, main, & port);

  rtems_task_delete(RTEMS_SELF);
}

rtems_task main(rtems_task_argument * port){
  // main code
  rtems_task_delete(RTEMS_SELF);
}

// Communication task
rtems_task ComTask (rtems_task_argument * port) {
  // task code
}

Figure 5: Producer RTEMS code.

Table 2: Line number of prod/cons and CDMA source code.

|            | SystemC | POSIX | RTEMS | POSIX | Static C | Static C |
|------------|---------|-------|-------|-------|----------|----------|
| Prod/Cons  | 86      | 130   | 203   | 161   | —        | —        |
| CDMA       | 976     | 1350  | 1479  | 1387  | 950      | 950      |

Table 3: Binary code size of prod/cons and CDMA.

|            | SystemC | POSIX | RTEMS | POSIX | Static C | Static C |
|------------|---------|-------|-------|-------|----------|----------|
| Prod/Cons  | 592 K   | 14 K  | 106 K | 83 K  | —        | —        |
| CDMA       | 1.8 M   | 32 K  | 119 K | 97 K  | 188 K    | 12 K     |

Table 4: Execution time of prod/cons and CDMA.

|            | SystemC | POSIX | RTEMS | POSIX | Static C | Static C |
|------------|---------|-------|-------|-------|----------|----------|
| Prod/Cons  | 43 μs   | 81 μs | 2.43 ms | 1.85 ms | —        | —        |
| CDMA       | 170 μs  | 310 μs | 12.5 ms | 9.2 ms | 17 μs    | 153 μs   |

In Table 2, the number of lines of the generated embedded C code is nearly the double for the first simple case which includes 27% of SystemC primitives. For the CDMA, the generated code size is nearly half more important with only 13% of SystemC primitives. The size of the embedded C generated code is directly linked to the number of SystemC elements included in the original code. As each SystemC primitive is translated with a set of embedded C instructions, a large proportion of read(), write(), wait(), and others primitives can result in an important size. However, the increase of the generated code size remains low. Moreover, this generated C code is entirely “readable” and can be completed or optimized manually.

In Table 3, the size of the statically scheduled code for embedded processor is nearly ten times lower than the RTOS one. Thus, when it is possible, it is more interesting to use an RTOS-less solution for embedded processors. For the Linux implementation, the kernel is not included in the code, thus its size is lower than for the standalone one which includes its own kernel.

In Table 4, the code execution time on the embedded processor with a static scheduling is nearly 60 times faster than the RTOS one. We have to consider here that the CDMA application highly communicates and thus highly requests RTOS services with context switching for each communication. In addition, the validation of the embedded software can be operated on the host processor, through direct POSIX execution, as it obtains comparable execution times compared to SystemC execution. We obtain also better execution times for a dedicated static scheduling that is nearly ten times faster than pure SystemC execution times.

It is thus possible to evaluate more rapidly the whole SystemC model by parsing it in C and execute it on the user computer instead of using pure SystemC simulations. The results collected in Tables 2, 3 and 4 show the feasibility of our SystemC parsing process to POSIX or static scheduling C code.

We have also experimented different CDMA multiprocessor implementations with static scheduling in order to...
evaluate the impact of HW/SW communication in term of time overhead. This overhead includes software delays from device drivers and hardware delays due to the NoC crossing. We have experimented several configurations with 1, 2, 4, and 7 processors connected with a one-dimension linear NoC with two processors per node. The speedup obtained is presented on Figure 6. The overhead of HW/SW communication (25 clock cycles for a write) added to the NoC crossing time (almost one clock per NoC node crossed) makes the impact of communication low compared to the execution time of the CDMA functions on the different processors. We obtain a speedup of 1.8 with 2 processors and 5 with 7 processors.

These results show the low implications of such a higher-level interface approach.

These previous design experiments conducted to measures of design cycle times used for time-to-market evaluation. Several groups of students have implemented these examples with the SystemC code as a starting point. As we evaluate here only the mapping of software IPs on embedded processors, we have provided also the VHDL code of the hardware platform. This platform is composed of LEON-2 processors and an NoC. We have compared the implementation time between our automated flow and two manual solutions. These experiment results presented in Table 5 show consequent implementation time differences. Even for a simple case, the manual design of a multithreaded (POSIX/MPI) code needs a debugging step which is time consuming. This is of course even more important for a fully static code which needs to design the scheduling of IPs and their communication (MPI). This debugging step needs to be conducted first as pure C/MPI code on the user computer and then on the VHDL platform. Moreover, even if no VHDL is designed here, they need at least to configure the platform in terms of addresses, and thus validate the entire system. A more complex system with VHDL integration would conduct to even more time-consuming implementation.

Table 5: Average design time of prod/cons and CDMA.

| Solutions  | Automated | Manual |
|------------|-----------|--------|
|            |           | POSIX C | Static C |
| Prod/Cons  | 1.5 H     | 6 H     | 1.5 days |
| CDMA       | 3 H       | 2 days  | 4 days   |

5. CONCLUSION

This paper deals with the idea of unifying the use of SystemC to implement both hardware and embedded software. We propose an automated HW/SW codesign methodology which reduces the design time of embedded systems. The proposed methodology uses the redefinition of SystemC elements to generate the embedded software. A first solution is to replace each SystemC element by typical RTOS functions and MPI primitives. This solution provides a significantly smaller code size than the equivalent SystemC code size. A second complementary solution is to generate a statically scheduled stand alone C code which exhibits better results in code size and execution time.

The main advantage of this methodology relies on the optimization of the different steps of the flow. These steps are jointly designed and optimized by integrating the features of the SystemC programming model. Actually, the NoC architecture is jointly designed with the MPI software of the communication layer. This approach is similar to the joint optimization of compilers and microarchitectures of microprocessors. According to the previous experiments, our optimized flow exhibits efficient results in terms of execution times and hardware resources. Finally, this automated flow can help to obtain fast design cycle times.

Future works concern the full support of the SPIRIT standard as well as the improvement of SynDEx mapping and routing solution.

REFERENCES

[1] P. Sánchez, “Embedded SW and RTOS,” in Design of HW/SW Embedded Systems, E. Villar, Ed., University of Cantabria, Santander, Spain, 2001.
[2] Forte Design Systems, “Cynthesizer 3.0,” http://www.forteds.com/.
[3] Celoxica, “Agility compiler user guide,” Celoxica, 2005.
[4] Mentor Graphics, “CatapultC,” http://www.mentor.com/.
[5] C. S. Ananian, “SiliconC: a hardware backend for SUIF”.
[6] CoWare, “SPW and Platform Architect,” http://www.coware.com/.
[7] R. K. Gupta, Co-Synthesis of Hardware and Software for Digital Embedded Systems, Kluwer Academic, Norwell, Mass, USA, 1995.
[8] J. L. Pino, S. Ha, E. A. Lee, and J. T. Buck, “Software synthesis for DSP using Ptolemy,” Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, vol. 9, no. 1–2, pp. 7–21, 1995.
[9] F. Baladin, M. Chiodo, P. Giusto, et al., Hardware-Software Codesign of Embedded Systems: The POLIS Approach, Kluwer Academic, Norwell, Mass, USA, 1997.
[10] Rational, http://www.rational.com/uml/index.html.
[11] D. Harel, H. Lachover, A. Naamad, et al., “STATEMATE: a working environment for the development of complex reactive systems,” IEEE Transactions on Software Engineering, vol. 16, no. 4, pp. 403–414, 1990.
[12] F. Boussinot and R. de Simone, “The ESTEREL language,” Proceedings of the IEEE, vol. 79, no. 9, pp. 1293–1304, 1991.
[13] T. Grötker, S. Liao, G. Martin, and S. Swan, System Design with SystemC, Kluwer Academic, Norwell, Mass, USA, 2002.
[14] D. Desmet, D. Verkest, and H. De Man, “Operating system based software generation for systems-on-chip,” in Proceedings of 37th Design Automation Conference, pp. 396–401, Los Angeles, Calif, USA, June 2000.

[15] F. Herrera, H. Posadas, P. Sanchez, and E. Villar, “Systemic embedded software generation from systemC,” in Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE ’03), pp. 10142–10149, Munich, Germany, March 2003.

[16] H. Yu, R. Dömer, and D. Gajski, “Embedded software generation from system level design languages,” in Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC ’04), pp. 463–468, Yokohama, Japan, January 2004.

[17] F. Pogodalla, R. Hersemeule, and P. Coulomb, “Fast Prototyping: a system design flow for fast design, prototyping and efficient IP reuse,” in Proceedings of the 7th International Conference on Hardware/Software Codesign (CODES ’99), pp. 69–73, Rome, Italy, May 1999.

[18] OCP Adoption Adds Value to Prosilog. www.prosilog.com/news/press/documents/.

[19] J.-Y. Brunel, W. M. Kruijftzer, H. J. H. N. Kenter, et al., “COSY communication IP’s,” in Proceedings of 37th Design Automation Conference, pp. 406–409, Los Angeles, Calif, USA, June 2000.

[20] D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao, SpecC: Specification Language and Methodology, Kluwer Academic, Norwell, Mass, USA, 2000.

[21] “Virtual Component Interface Standard (OCB 2.1.0),” VSIA on-Chip Bus Development Working Group, March 14, 2000.

[22] W. Klingauf, “Systematic transaction level modeling of embedded systems with systemC,” in Proceedings of Design, Automation and Test in Europe (DATE ’05), vol. 1, pp. 566–567, Munich, Germany, March 2005.

[23] W. Gropp, E. Lusk, and R. Thakur, Using MPI-2 Advanced Features of the Message Passing Interface, MIT Press, Cambridge, Mass, USA, 1999.

[24] S. Ouadjaout and D. Houzet, “Easy SoC design with VCI systemC adapters,” in Proceedings of the EUROMICRO Systems on Digital System Design (DSD ’04), pp. 316–323, Rennes, France, August-September 2004.

[25] S. Ouadjaout, M.-F. Albenge, and D. Houzet, “VSIA interface cosynthesis,” in Proceedings of the 1st IEEE International Workshop on Electronic Design, Test and Applications (DELTA ’02), pp. 43–46, Christchurch, New Zealand, January 2002.

[26] SPIRIT Consortium, “SPIRIT V2.0 Alpha release,” 2006.

[27] T. Grandpierre and Y. Sorel, “From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations,” in Proceedings of 1st ACM and IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE ’03), pp. 123–132, Mont Saint-Michel, France, June 2003.

[28] S. G. Ziavras, A. V. Gerbessiotis, and R. Bafta, “Coprocessor design to support MPI primitives in configurable multiprocessors,” to appear in Integration, the VLSI Journal.

[29] S. Evain, J.-P. Diguet, and D. Houzet, “μSpider: a CAD tool for efficient NoC design,” in Proceedings of 22nd Norchip Conference, pp. 218–221, Oslo, Norway, November 2004.

[30] S. Ouadjaout and D. Houzet, “Embedded hardware/software generation from high level design languages,” in Proceedings of IEEE International Computer Systems & Information Technology Conference (ICSIT ’05), Algiers, Algeria, July 2005.

Salim Ouadjaout received the M.S. degree in computer science from the National Institute of Computers (INI), Algeria, in 2000, and the M.S. degree from INP, ENSEEIHT, Toulouse, France, in 2001. He is a Ph.D. candidate in the Electrical and Computer Engineering Department at the Institute of Electronics and Telecommunication, Rennes, France. He is also working as a Research Engineer at M3Systems, Inc. He has been an ACM Student Member. His research interests include design methodologies, interface synthesis, micronetworks for SoC, and embedded multiprocessors SoC.

Dominique Houzet received the M.S. degree in computer sciences in 1989 from Paul Sabatier University, Toulouse, France, and the Ph.D. degree and HDR degree in computer architecture in 1992 and 1999, both from INPT, ENSEEIHT, Toulouse, France. He worked at IRIT Laboratory and ENSEEIHT Engineering School from 1992 to 2002 as an Assistant Professor and at IETR Laboratory INSa Engineering School in Rennes from 2002 to 2006 and also as a Digital Design Consultant with SME and large companies. He is now a Professor at LIS-INPG, Grenoble. He has published a number of research papers in the area of parallel computer architecture and SoC design and a book on VHDL principles. His research interests include codesign and SoC design methodologies applied to image processing and radiocommunications. He is a Member of the IEEE Computer Society.
Efficient Design Methods for Embedded Communication Systems

M. Holzer, B. Knerr, P. Belanović, and M. Rupp

Institute for Communications and Radio Frequency Engineering, Vienna University of Technology,
Gußhausstraße 25/389, 1040 Vienna, Austria

Received 1 December 2005; Revised 11 April 2006; Accepted 24 April 2006

Nowadays, design of embedded systems is confronted with complex signal processing algorithms and a multitude of computational intensive multimedia applications, while time to product launch has been extremely reduced. Especially in the wireless domain, those challenges are stacked with tough requirements on power consumption and chip size. Unfortunately, design productivity did not undergo a similar progression, and therefore fails to cope with the heterogeneity of modern architectures. Electronic design automation tools exhibit deep gaps in the design flow like high-level characterization of algorithms, floating-point to fixed-point conversion, hardware/software partitioning, and virtual prototyping. This tutorial paper surveys several promising approaches to solve the widespread design problems in this field. An overview over consistent design methodologies that establish a framework for connecting the different design tasks is given. This is followed by a discussion of solutions for the integrated automation of specific design tasks.

Copyright © 2006 M. Holzer et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. INTRODUCTION

Over the past 25 years, the field of wireless communications has experienced a rampant growth, in both popularity and complexity. It is expected that the global number of mobile subscribers will reach more than three billion in the year 2008 [1]. Also, the complexity of the modern communication systems is growing so rapidly, that the next generation of mobile devices for 3G UMTS systems is expected to be based on processors containing more than 40 million transistors [2]. Hence, during this relatively short period of time, a staggering increase in complexity of more than six orders of magnitude has taken place [3].

In comparison to this extremely fast-paced growth in algorithmic complexity, the concurrent increase in the complexity of silicon-integrated circuits proceeds according to the well-known Moore law [4], famously predicting the doubling of the number of transistors integrated onto a single integrated circuit every 18 months. Hence, it can be concluded that the growth in silicon complexity lags behind the extreme growth in the algorithmic complexity of wireless communication systems. This is also known as the algorithmic complexity gap.

At the same time, the International Technology Roadmap for Semiconductors [5] reported a growth in design productivity, expressed in terms of designed transistors per staff-month, of approximately 21% compounded annual growth rate (CAGR), which lags behind the growth in silicon complexity. This is known as the design gap or productivity gap.

The existence of both the algorithmic and the productivity gaps points to inefficiencies in the design process. At various stages in the process, these inefficiencies form bottlenecks, impeding increased productivity which is needed to keep up with the mentioned algorithmic demand.

In order to clearly identify these bottlenecks in the design process, we classify them into internal and external barriers. Many potential barriers to design productivity arise from the design teams themselves, their organisation, and interaction. The traditional team structure [6] consists of the research (or algorithmic), the architectural, and the implementation teams. Hence, it is clear that the efficiency of the design process, in terms of both time and cost, depends not only on the forward communication structures between teams, but also on the feedback structures (i.e., bug reporting) in the design process. Furthermore, the design teams use separate system descriptions. Additionally, these descriptions are very likely written in different design languages.

In addition to these internal barriers, there exist several external factors which negatively affect the efficiency of the design process. Firstly, the work of separate design teams is
supported by a wide array of different EDA software tools. Thus, each team uses a completely separate set of tools to any other team in the design process. Moreover, these tools are almost always incompatible, preventing any direct and/or automated cooperation between teams.

Also, EDA tool support exhibits several “gaps,” that is, parts of the design process which are critical, yet for which no automated tools are available. Although they have high impact on the rest of the design process, these steps typically have to be performed manually, due to their relatively large complexity, thus requiring designer intervention and effort. Designers typically leverage their previous experience to a large extent when dealing with these complex issues.

In Figure 1 a design flow is shown, which identifies several intermediate design steps (abstraction levels) that have to be covered during the refinement process. This starts with an algorithm that is described and verified, for example, in a graphical environment with SystemC [7]. Usually in the wireless domain algorithms are described by a synchronous data flow graph (SDFG), where functions (A, B, C, D, E) communicate with fixed data rates to each other. An intermediate design step is shown, where already hardware/software partitioning has been accomplished, but the high abstraction of the signal processing functions is still preserved. Finally the algorithm is implemented utilising a heterogenous architecture that consists of processing elements (DSPs, ASICs), memory, and a bus system.

Also some design tasks are mentioned, which promise high potential for decreasing design time by its automation. This paper discusses the requirements and solutions for an integrated design methodology in Section 2. Section 3 reports on high-level characterisation techniques in order to have early estimations of the final system properties and allows to make first design decisions. Section 4 presents environments for the conversion of data from floating-point to fixed-point representation. Approaches for automated hardware/software partitioning are shown in Section 5. The decrease of design time by virtual prototyping is presented in Section 6. Finally, conclusions end the paper.

2. CONSISTENT DESIGN FLOW

2.1. Solution requirements

In the previous section, a number of acute bottlenecks in the design process have been identified. In essence, an environment is needed, which transcends the interoperability problems of modern EDA tools. To achieve this, the environment has to be flexible in several key aspects.

Firstly, the environment has to be modular in nature. This is required to allow expansion to include new tools as
they become available, as well as to enable the designer to build a custom design flow only from those tools which are needed.

Also, the environment has to be independent from any particular vendor’s tools or formats. Hence, the environment will be able to integrate tools from various vendors, as well as academic/research projects, and any in-house developed automation, such as scripts, templates, or similar.

To allow unobstructed communication between teams, the environment should eliminate the need for separate system descriptions. Hence, the single system description, used by all the teams simultaneously, would provide the ultimate means of cooperative refinement of a design, from the initial concept to the final implementation. Such a single system description should also be flexible through having a modular structure, accommodating equally all the teams. Thus, the structure of the single system description is a superset of all the constructs required by all the teams, and the contents of the single system description is a superset of all the separate system descriptions used by the teams currently.

2.2. Survey of industrial and university approaches

Several research initiatives, both in the commercial and academic arenas, are currently striving to close the design and productivity gaps. This section presents a comparative survey of these efforts.

A notable approach to EDA tool integration is provided by the model integrated computing (MIC) community [8]. This academic concept of model development gave rise to an environment for tool integration [9]. In this environment, the need for centering the design process on a single description of the system is also identified, and the authors present an implementation in the form of an integrated model server (IMS), based on a database system. The structure of the entire environment is expandable and modular in structure, with each new tool introduced into the environment requiring a new interface. The major shortcoming of this environment is its dedication to development of software components only. As such, this approach addresses solely the algorithmic modelling of the system, resulting in software at the application level. Thus, this environment does not support architectural and implementation levels of the design process.

Synopsys is one of the major EDA tool vendors offering automated support for many parts of the design process. Recognising the increasing need for efficiency in the design process and integration of various EDA tools, Synopsys developed a commercial environment for tool integration, the Galaxy Design Platform [10]. This environment is also based on a single description of the system, implemented as a database and referred to as the open Milkyway database. Thus, this environment eliminates the need for rewriting system descriptions at various stages of the design process. It also covers both the design and the verification processes and is capable of integrating a wide range of Synopsys commercial EDA tools. An added bonus of this approach is the open nature of the interface format to the Milkyway database, allowing third-party EDA tools to be integrated into the tool chain, if these adhere to the interface standard. However, this environment is essentially a proprietary scheme for integrating existing Synopsys products, and as such lacks any support from other parties.

The SPIRIT consortium [11] acknowledges the inherent inefficiency of interfacing incompatible EDA tools from various vendors. The work of this international body focuses on creating interoperability between different EDA tool vendors from the point of view of their customers, the product developers. Hence, the solution offered by the SPIRIT consortium [12] is a standard for packaging and interfacing of IP blocks used during system development. The existence and adoption of this standard ensures interoperability between EDA tools of various vendors as well as the possibility for integration of IP blocks which conform to the standard. However, this approach requires widest possible support from the EDA industry, which is currently lacking. Also, even the full adoption of this IP interchange format does not eliminate the need for multiple system descriptions over the entire design process. Finally, the most serious shortcoming of this methodology is that it provides support only for the lower levels of the design process, namely, the lower part of the architecture level (component assembly) and the implementation level.

In the paper of Posadas et al. [13] a single source design environment based on SystemC is proposed. Within this environment analysis tools are provided for time estimations for either hardware or software implementations. After this performance evaluation, it is possible to insert hardware/software partitioning information directly in the SystemC source code. Further, the generation of software for real-time application is addressed by a SystemC-to-eCos library, which replaces the SystemC kernel by real-time operating system functions. Despite being capable of describing a system consistently on different abstraction levels based on a single SystemC description, this does not offer a concrete and general basis for integration of design tools at all abstraction levels.

Rault et al. [14] present a rapid prototyping environment based on a single tool called SynDex. Within this environment the user starts by defining an algorithm graph, an architecture graph, and constraints. Further executables for special kernels are automatically generated, while heuristics are used to minimize the total execution time of the algorithm. Those kernels provide the functionality of implementations in software and hardware, as well as models for communication.

The open tool integration environment (OTIE) [15] is a consistent design environment, aimed at fulfilling the requirements set out in Section 2.1. This environment is based on the single system description (SSD), a central repository for all the refinement information during the entire design process. As such, the SSD is used simultaneously by all the design teams. In the OTIE, each tool in the design process still performs its customary function, as in the traditional tool chain, but the design refinements from all the tools are now stored in just one system descriptions (the SSD) and thus no longer subject to constant rewriting. Hence, the SSD is a
superset of all the system descriptions present in the traditional tool chain.

The SSD is implemented as a MySQL [16] database, which brings several benefits. Firstly, the database implementation of the SSD supports virtually unlimited expandability, in terms of both structure and volume. As new refinement information arrives to be stored in the SSD, either it can be stored within the existing structure, or it may require an extension to the entity-relationship structure of the SSD, which can easily be achieved through addition of new tables or links between tables. Also, the database, on which this implementation of the SSD is based, is inherently a multiuser system, allowing transparent and uninterrupted access to the contents of the SSD to all the designers simultaneously. Furthermore, the security of the database implementation of the SSD is assured through detailed setting of access privileges of each team member and integrated EDA design tool to each part of the SSD, as well as the seamless integration of a version control system, to automatically maintain revision history of all the information in the SSD. Finally, accessing the refinement information (both manually and through automated tools) is greatly simplified in the database implementation of the SSD by its structured query language (SQL) interface.

Several EDA tool chains have been integrated into the OTIE, including environments for virtual prototyping [17, 18], hardware/software partitioning [19], high-level system characterisation [20], and floating-point to fixed-point conversion [21]. The deployment of these environments has shown the ability of the OTIE concept to reduce the design effort drastically through increased automation, as well as close the existing gaps in the automation coverage, by integrating novel EDA tools as they become available.

3. SYSTEM ANALYSIS

For the design of a signal processing system consisting of hardware and software many different programming languages have been introduced like VHDL, Verilog, or SystemC. During the refinement process it is of paramount importance to assure the quality of the written code and to base the design decisions on reliable characteristics. Those characteristics of the code are called metrics and can be identified on the different levels of abstraction.

The terms metric and measure are used as synonyms in literature, whereas a metric is in general a measurement, which maps an empirical object to a numerical object. This function should preserve all relations and structures. In other words, a quality characteristic should be linearly related to a measure, which is a basic concept of measurement at all. Those metrics can be software related or hardware related.

3.1. Software-related metrics

In the area of software engineering the interest in the measurement of software properties is ongoing since the first programming languages appeared [22]. One of the earliest software measures is the lines of code [23], which is still used today.

In general the algorithm inside a function, written in the form of sequential code can be decomposed into its control flow graph (CFG), built up of interconnected basic blocks (BB). Each basic block contains a sequence of data operations ending in a control flow statement as a last instruction. A control flow graph is a directed graph with only one root and one exit. A root defines a vertex with no incoming edge and the exit defines a vertex with no outgoing edge. Due to programming constructs like loops those graphs are not cycle-free. The sequence of data operations inside of one BB forms itself a data flow graph (DFG) or equivalently one or more expression trees. Figure 2 shows an example of a function and its graph descriptions.

For the generation of DFG and CFG a parsing procedure of the source code has to be accomplished. This task is usually performed by a compiler. The step of compilation is separated into two steps, firstly, a front end transforms the source code into an intermediate representation (abstract syntax tree). At this step target independent optimizations are already applied, like dead code elimination or constant propagation. In a second step, the internal representation is mapped to a target architecture.

The analysis of a CFG can have different scopes: a small number of adjacent instructions, a single basic block, across several basic blocks (intraprocedural), across procedures (interprocedural), or a complete program.

For the CFG and DFG some common basic properties can be identified as follows.

(i) For each graph type $G$, a set of vertices $V$, and edges $E$ can be defined, where the value $|V|$ denotes the number of vertices and $|E|$ denotes the number of edges.

(ii) A path of $G$ is defined as an ordered sequence $S = (v_{root}, v_{x}, v_{y}, \ldots, v_{exit})$ of vertices starting at the root and ending at the exit vertex.

![Figure 2: Control flow graph (CFG) and expression tree of one basic block.](image-url)
In Figure 3 it can be seen that for a $\gamma$ value of 1, the graph is sequential and for $\gamma > 1$ the graph has many vertices in parallel, which offers possibilities for the reuse of resources.

In order to render the CFG context more precisely, we can apply these properties and define some important metrics to characterise the algorithm.

**Definition 1** (longest path weight for operation $j$). Every vertex of a CFG can be annotated with a set of different weights $w(v_i) = (w_{i1}, w_{i2}, \ldots, w_{im})^T, i = 1 \cdots |V|$, that describes the occurrences of its internal operations (e.g., $w_i$ number of ADD operations in vertex $v_i$). Accordingly, a specific longest path with respect to the $j$th distinct weight, $S_{LP}^j$, can be defined as the sequence of vertices $(V_{\text{root}} v_1 \cdots v_{\text{exit}})$, which yields a maximum path weight $\text{PW}_j$ by summing up all the weights $w_{\text{root}}^j, w_{j1}^j, \ldots, w_{\text{exit}}^j$ of the vertices that belong to this path as in

$$\text{PW}_j = \sum_{v_i \in S_{LP}^j} w(v_i) \mathbf{d}_j.$$  

Here the selection of the weight with the type $j$ is accomplished by multiplication with a vector $\mathbf{d}_j = (\delta_{i1}, \ldots, \delta_{mj})^T$ defined with the Kronecker-delta $\delta_{ij}$.

**Definition 2** (degree of parallelism for operation $j$). Similar to the path weight $\text{PW}_j$, a global weight $\text{GW}_j$ can be defined as

$$\text{GW}_j = \sum_{v_i \in V} w(v_i) \mathbf{d}_j,$$

which represents the operation-specific weight of the whole CFG. Accordingly an operation-specific $\gamma_j$ is defined as follows:

$$\gamma_j = \frac{\text{GW}_j}{\text{PW}_j}$$  

(iii) The path with the maximum number of vertices is called the longest path or critical path and consists of $|V_{LP}|$ vertices.

(iv) The degree of parallelism $\gamma$ can be defined as the number of all vertices $|V|$ divided by the number of vertices in the longest path $|V_{LP}|$ of the algorithm

$$\gamma = \frac{|V|}{|V_{LP}|}. $$

In Figure 3 it can be seen that for a $\gamma$ value of 1, the graph is sequential and for $\gamma > 1$ the graph has many vertices in parallel, which offers possibilities for the reuse of resources.

In order to render the CFG context more precisely, we can apply these properties and define some important metrics to characterise the algorithm.

**Definition 3** (cyclomatic complexity). The cyclomatic complexity, as defined by McCabe [25], states the theoretical number (see (5)) of required test cases in order to achieve the structural testing criteria of a full path coverage:

$$V(G) = |E| - |V| + 2.$$  

The generation of the verification paths is presented by Poole [26] based on a modified depth-first search through the CFG.

**Definition 4** (control orientation metrics). The control orientation metrics (COM) identifies whether a function is dominated by control operations,

$$\text{COM} = \frac{N_{\text{cop}}}{N_{\text{op}} + N_{\text{cop}} + N_{\text{mac}}}.$$  

Here $N_{\text{cop}}$ defines the number of control statements (if, for, while), $N_{\text{op}}$ defines the number of arithmetic and logic operations, and $N_{\text{mac}}$ the number of memory accesses. When the COM value tends to be 1 the function is dominated by control operations. This is usually an indicator that an implementation of a control-oriented algorithm is more suited for running on a controller than to be implemented as dedicated hardware.

### 3.2 Hardware-related metrics

Early estimates of area, execution time, and power consumption of a specific algorithm implemented in hardware are crucial for design decisions like hardware/software partitioning (Section 5) and architecture exploration (Section 6.1). The effort of elaborating different implementations is usually not feasible in order to find optimal solutions. Therefore, only critical parts are modelled (rapid prototyping [6]) in order to measure worst-case scenarios, with the disadvantage that side effects on the rest of the system are neglected. According to Gajski et al. [27] those estimates must satisfy three criteria: accuracy, fidelity, and simplicity.

The estimation of area is based on an area characterization of the available operations and on an estimation of the needed number of operations (e.g., ADD, MUL). The area consumption of an operation is usually estimated by a function dependent on the number of inputs/outputs and their bit widths [28]. Further, the number of operations, for example, in Boolean expressions can be estimated by the number of nodes in the corresponding Boolean network [29]. Area estimation for design descriptions higher than register transfer level, like SystemC, try to identify a simple model for the high-level synthesis process [30].
The estimation of execution time of a hardware implementation requires the estimation of scheduling and resource allocation, which are two interdependent tasks. Path-based techniques transform an algorithm description from its CFG and DFG representation into a directed acyclic graph. Within this acyclic graph worst-case paths can be investigated by static analysis [31]. In simulation-based approaches the algorithm is enriched with functionality for tracing the execution paths during the simulation. This technique is, for example, described for SystemC [32] and MATLAB [33]. Additionally, a characterization of the operations regarding their timing (delay) has to be performed.

Power dissipation in CMOS is separated into two components, the static and the dominant dynamic parts. Static power dissipation is mainly caused by leakage currents, whereas the dynamic part is caused by charging/discharging capacitances and the short circuit during the switching. Charging accounts for over 90% of the overall power dissipation [34]. Assuming that capacitance is related to area, area estimation techniques, as discussed before, have to be applied. Fornaciari et al. [35] present power models for different functional units like registers and multiplexers. Several techniques for predicting the switching activity of a circuit are presented by Landman [36].

### 3.3. Cost function and affinity

Usually the design target is the minimization of a cost or objective function with inequality constraints [37]. This cost function $c$ depends on $x = (x_1, \ldots, x_n)^T$, where the elements $x_i$ represent normalized and weighted values of timing, area, and power but also economical aspects (e.g., cyclomatic complexity relates to verification effort) could be addressed. This leads to the minimization problem

$$\min \; c(x). \quad (7)$$

Additionally those metrics have a set of constraints $b_i$ like maximum area, maximum response time, or maximum power consumption given by the requirements of the system. Those constraints, which can be grouped to a vector $b = (b_1, \ldots, b_n)^T$ define a set of inequalities,

$$x \leq b. \quad (8)$$

There is another iteration of the presented metrics is its usage for the hardware/software partitioning process. Here a huge search space demands for heuristics that allows for partitioning within reasonable time. Nevertheless, a reduction of the search space can be achieved by assigning certain functions to hardware or software beforehand. This can be accomplished by an affinity metric [38]. Such an affinity can be expressed in the following way:

$$A = \frac{1}{\text{COM}} + \sum_{j \in f} y_j. \quad (9)$$

A high value $A$ and thus a high affinity of an algorithm to a hardware implementation are caused by less control operations and high parallelism of the operations that are used in the algorithm. Thus an algorithm with an affinity value higher than a certain threshold can be selected directly to be implemented in hardware.

### 4. Floating-Point to Fixed-Point Conversion

Design of embedded systems typically starts with the conversion of the initial concept of the system into an executable algorithmic model, on which high-level specifications of the system are verified. At this level of abstraction, models invariably use floating-point formats, for several reasons. Firstly, while the algorithm itself is undergoing changes, it is necessary to disburden the designer from having to take numeric effects into account. Hence, using floating-point formats, the designer is free to modify the algorithm itself, without any consideration of overflow and quantization effects. Also, floating-point formats are highly suitable for algorithmic modeling because they are natively supported on PC or workstation platforms, where algorithmic modeling usually takes place.

However, at the end of the design process lies the implementation stage, where all the hardware and software components of the system are fully implemented in the chosen target technologies. Both the software and hardware components of the system at this stage use only fixed-point numeric formats, because the use of fixed-point formats allows drastic savings in all traditional cost metrics: the required silicon area, power consumption, and latency/throughput (i.e., performance) of the final implementation.

Thus, during the design process it is necessary to perform the conversion from floating-point to suitable fixed-point numeric formats, for all data channels in the system. This transition necessitates careful consideration of the ranges and precision required for each channel, the overflow and quantisation effects created by the introduction of the fixed-point formats, as well as a possible instability which these formats may introduce. A trade-off optimization is hence formed, between minimising introduced quantisation noise and minimising the overall bitwidths in the system, so as to minimise the total system implementation cost. The level of introduced quantisation noise is typically measured in terms of the signal to quantisation noise ratio (SQNR), as defined in (10), where $v$ is the original (floating-point) value of the signal and $\hat{v}$ is the quantized (fixed-point) value of the signal:

$$\text{SQNR} = 20 \times \log \left| \frac{v}{|v - \hat{v}|} \right|. \quad (10)$$

The performance/cost tradeoff is traditionally performed manually, with the designer estimating the effects of fixed-point formats through system simulation and determining the required bitwidths and rounding/overflow modes through previous experience or given knowledge of the system architecture (such as predetermined bus or memory interface bitwidths). This iterative procedure is very time consuming and can sometimes account for up to 50% of the total design effort [39]. Hence, a number of initiatives to automate the conversion from floating-point to fixed-point formats have been set up.
In general, the problem of automating the conversion from floating-point to fixed-point formats can be based on either an analytical (static) or statistical (dynamic) approach. Each of these approaches has its benefits and drawbacks.

### 4.1. Analytical approaches

All the analytical approaches to automate the conversion from floating-point to fixed-point numeric formats find their roots in the static analysis of the algorithm in question. The algorithm, represented as a control and data flow graph (CDFG), is statically analysed, propagating the bitwidth requirements through the graph, until the range, precision, and sign mode of each signal are determined.

As such, analytical approaches do not require any simulations of the system to perform the conversion. This typically results in significantly improved runtime performance, which is the main benefit of employing such a scheme. Also, analytical approaches do not make use of any input data for the system. This relieves the designer from having to provide any data sets with the original floating-point model and makes the results of the optimisation dependent only on the algorithm itself and completely independent of any data which may eventually be used in the system.

However, analytical approaches suffer from a number of critical drawbacks in the general case. Firstly, analytical approaches are inherently only suitable for finding the upper bound on the required precision, and are unable to perform the essential trade-off between system performance and implementation cost. Hence, the results of analytical optimisations are excessively conservative, and cannot be used to replace the designer’s fine manual control over the trade-off. Furthermore, analytical approaches are not suitable for use on all classes of algorithms. It is in general not possible to process nonlinear, time-variant, or recursive systems with these approaches.

FRIDGE [39] is one of the earliest environments for floating-point to fixed-point conversion and is based on an analytical approach. This environment has high runtime performance, due to its analytical nature, and wide applicability, due to the presence of various back-end extensions to the core engine, including the VHDL back end (for hardware component synthesis) and ANSI-C and assembly back ends (for DSP software components). However, the core engine relies fully on the designer to preassign fixed-point formats to a sufficient portion of the signals, so that the optimisation engine may propagate these to the rest of the CDFG structure of the algorithm. This environment is based on fixed-C, a proprietary extension to the ANSI-C core language and is hence not directly compatible with standard design flows. The FRIDGE environment forms the basis of the commercial Synopsys CoCentric Fixed-Point Designer [40] tool.

Another analytical approach, Bitwise [41], implements both forward and backward propagations of bitwidth requirements through the graph representation of the system, thus making more efficient use of the available range and precision information. Furthermore, this environment is capable of tackling complex loop structures in the algorithm by calculating their closed-form solutions and using these to propagate the range and precision requirements. However, this environment, like all analytical approaches, is not capable of carrying out the performance-cost trade-off and results in very conservative fixed-point formats.

An environment for automated floating-point to fixed-point conversion for DSP code generation [42] has also been presented, minimising the execution time of DSP code through the reduction of variable bitwidths. However, this approach is only suitable for software components and disregards the level of introduced quantisation noise as a system-level performance metric in the trade-off.

An analytical approach based on affine arithmetic [43] presents another fast, but conservative, environment for automated floating-point to fixed-point conversion. The unique feature of this approach is the use of probabilistic bounds on the distribution of values of a data channel. The authors introduce the probability factor \( \lambda \), which in a normal hard upper-bound analysis equals 1. Through this probabilistic relaxation scheme, the authors set \( \lambda = 0.999999 \) and thereby achieve significantly more realistic optimisation results, that is to say, closer to those achievable by the designer through system simulations. While this scheme provides a method of relaxing the conservative nature of its core analytical approach, the mechanism of controlling this separation (namely, the trial-and-error search by varying the \( \lambda \) factor) does not provide a means of controlling the performance-cost tradeoff itself and thus replacing the designer.

### 4.2. Statistical approaches

The statistical approaches to perform the conversion from floating-point to fixed-point numeric formats are based on system simulations and use the resulting information to carry out the performance-cost tradeoff, much like the designer does during the manual conversion.

Due to the fact that these methods employ system simulations, they may require extended runtimes, especially in the presence of complex systems and large volumes of input data. Hence, care has to be taken in the design of these optimisation schemes to limit the number of required system simulations.

The advantages of employing a statistical approach to automate the floating-point to fixed-point conversion are numerous. Most importantly, statistical algorithms are inherently capable of carrying out the performance-cost trade-off, seamlessly replacing the designer in this design step. Also, all classes of algorithms can be optimised using statistical approaches, including nonlinear, time-variant, or recursive systems.

One of the earliest research efforts to implement a statistical floating-point to fixed-point conversion scheme concentrates on DSP designs represented in C/C++ [44]. This approach shows high flexibility, characteristic to statistical approaches, being applicable to nonlinear, recursive, and time-variant systems.

However, while this environment is able to explore the performance-cost tradeoff, it requires manual intervention.
by the designer to do so. The authors employ two optimisation algorithms to perform the trade-off: full search and a heuristic with linear complexity. The high complexity of the full search optimisation is reduced by grouping signals into clusters, and assigning the same fixed-point format to all the signals in one cluster. While this can reduce the search space significantly, it is an unrealistic assumption, especially for custom hardware implementations, where all signals in the system have different optimal fixed-point formats.

QDDV [45] is an environment for floating-point to fixed-point conversion, aimed specifically at video applications. The unique feature of this approach is the use of two performance metrics. In addition to the widely used objective metric, the SQNR, the authors also use a subjective metric, the mean opinion score (MOS) taken from ten observers.

While this environment does employ a statistical framework for measuring the cost and performance of a given fixed-point format, no automation is implemented and no optimisation algorithms are presented. Rather, the environment is available as a tool for the designer to perform manual “tuning” of the fixed-point formats to achieve acceptable subjective and objective performance of the video processing algorithm in question. Additionally, this environment is based on Valen-C, a custom extension to the ANSI-C language, thus making it incompatible with other EDA tools.

A further environment for floating-point to fixed-point conversion based on a statistical approach [46] is aimed at optimising models in the MathWorks Simulink [47] environment. This approach derives an optimisation framework for the performance-cost trade-off, but provides no optimisation algorithms to actually carry out the trade-off, thus leaving the conversion to be performed by the designer manually.

A fully automated environment for floating-point to fixed-point conversion called fixify [21] has been presented, based on a statistical approach. While this results in fine control over the performance-cost trade-off, fixify at the same time dispenses with the need for exhaustive search optimisations and thus drastically reduces the required runtimes. This environment fully replaces the designer in making the performance-cost trade-off by providing a palette of optimisation algorithms for different implementation scenarios.

For designs that are to be mapped to software running on a standard processor core, restricted-set full search is the best choice of optimisation technique, since it offers guaranteed optimal results and optimises the design directly to the set of fixed-point bitwidths that are native to the processor core in question. For custom hardware implementations, the best choice of optimisation option is the branch-and-bound algorithm [48], offering guaranteed optimal results. However, for high-complexity designs with relatively long simulation times, the greedy search algorithm is an excellent alternative, offering significantly reduced optimisation runtimes, with little sacrifice in the quality of results.

Figure 4 shows the results of optimising a multiple-input multiple-output (MIMO) receiver design by all three optimisation algorithms in the fixify environment. The results are presented as a trade-off between the implementation cost c (on the vertical axis) and the SQNR, as defined in (10) (on the horizontal axis). It can immediately be noted from Figure 4 that all three optimisation methods generally require increased implementation cost with increasing SQNR requirements, as is intuitive. In other words, the optimisation algorithms are able to find fixed-point configurations with lower implementation costs when more degradation of numeric performance is allowed.

It can also be noted from Figure 4 that the optimisation results of the restricted-set full search algorithm consistently (i.e., over the entire examined range [5 dB, 100 dB]) require higher implementation costs for the same level of numeric performance than both the greedy and the branch-and-bound optimisation algorithms. The reason for this effect is the restricted set of possible bitwidths that the full search algorithm can assign to each data channel. In this example, the restricted-set full search algorithm uses the word length set of \{16, 32, 64\}, corresponding to the available set of fixed-point formats on the TIC6416 DSP which is used in the original implementation [49]. The full search algorithm can only move through the solution space in large quantum steps, thus not being able to fine tune the fixed-point format of each channel. On the other hand, greedy and branch-and-bound algorithms both have full freedom to assign any positive integer (strictly greater than zero) as the word length of the fixed-point format for each channel in the design, thus consistently being able to extract fixed-point configurations with lower implementation costs for the same SQNR levels.

Also, Figure 4 shows that, though the branch-and-bound algorithm consistently finds the fixed-point configuration with the lowest implementation cost for a given level of SQNR, the greedy algorithm performs only slightly worse. In 13 out of the 20 optimisations, the greedy algorithm returned the same fixed-point configuration as the branch-and-bound algorithm. In the other seven cases, the subtree relaxation routine of the branch-and-bound algorithm discovered a superior fixed-point configuration. In these cases, the relative improvement of using the branch-and-bound algorithm ranged between 1.02% and 3.82%.

Furthermore, it can be noted that the fixed-point configuration found by the designer manually can be improved...
for both the DSP implementation (i.e., with the restricted-set full search algorithms) and the custom hardware implementation (i.e., with the greedy and/or branch-and-bound algorithms). The designer optimized the design to the fixed-point configuration where all the word lengths are set to 16 bits by manual trial and error, as is traditionally the case. After confirming that the design has satisfactory performance with all word lengths set to 32 bits, the designer assigned all the word lengths to 16 bits and found that this configuration also performs satisfactorily. However, it is possible to obtain lower implementation cost for the same SQNR level, as well as superior numeric performance (i.e., higher SQNR) for the same implementation cost, as can be seen in Figure 4.

It is important to note that fixify is based entirely on the SystemC language, thus making it compatible with other EDA tools and easier to integrate into existing design flows. Also, the fixify environment requires no change to the original floating-point code in order to perform the optimisation.

5. HARDWARE/SOFTWARE PARTITIONING

Hardware/software partitioning can in general be described as the mapping of the interconnected functional objects that constitute the behavioural model of the system onto a chosen architecture model. The task of partitioning has been thoroughly researched and enhanced during the last 15 years and produced a number of feasible solutions, which depend heavily on their prerequisites:

(i) the underlying system description;
(ii) the architecture and communication model;
(iii) the granularity of the functional objects;
(iv) the objective or cost function.

The manifold formulations entail numerous very different approaches to tackle this problem. The following subsection arranges the most fundamental terms and definitions that are common in this field and shall prepare the ground for a more detailed discussion of the sophisticated strategies in use.

5.1. Common terms

The functionality can be implemented with a set of interconnected system components, such as general-purpose CPUs, DSPs, ASICS, ASIPs, memories, and buses. The designer’s task is in general twofold: selection of a set of system components or, in other words, the determination of the architecture, and the mapping of the system’s functionality among these components. The term partitioning, originally describing only the latter, is usually adopted for a combination of both tasks, since these are closely interlocked. The level, on which partitioning is performed, varies from group to group, as well as the expressions to describe these levels. The term system level has always been referring to the highest level of abstraction. But in the early nineties the system level identified VHDL designs composed of several functional objects in the size of an FIR or LUT. Nowadays the term system level describes functional objects of the size of a Viterbi or a Huffman decoder. The complexity differs by one order of magnitude. In the following the granularity of the system partitioning is labelled decreasingly as follows: system level (e.g., Viterbi, UMTS Slot Synchronisation, Huffman, Quicksort, etc.), process level (FIR, LUT, Gold code generator, etc.), and operational level (MAC, ADD, NAND, etc). The final implementation has to satisfy a set of design constraints, such as cost, silicon area, power consumption, and execution time. Measures for these values, obtained by high-level estimation, simulation, or static analysis, which characterize a given solution quantitatively are usually called metrics; see Section 3. Depending on the specific problem formulation a selection of metrics comprises an objective function, which captures the overall quality of a certain partitioning as described in detail in Section 3.3.

5.2. Partitioning approaches

Ernst et al. [50] published an early work on the partitioning problem starting from an all-software solution within the COSYMA system. The underlying architecture model is composed of a programmable processor core, memory, and customised hardware (Figure 5).

The general strategy of this approach is the hardware extraction of the computational intensive parts of the design, especially loops, on a fine-grained basic block level (CDFG), until all timing constraints are met. These computation intensive parts are identified by simulation and profiling. User interaction is demanded since the system description language is C, a superset of ANSI-C. Not all C constructs have valid counterparts in a hardware implementation, such as dynamic data structures, and pointers. Internally simulated annealing (SA) [51] is utilized to generate different partitioning solutions. In 1994 the authors introduced an optional programmable coprocessor in case the timing constraints could not be met by hardware extraction [52]. The scheduling of the basic blocks is identified to be as soon as possible.
(ASAP) driven, in other words, it is the simplest list scheduling technique also known as earliest task first. A further improvement of this approach is the usage of a dynamically adjustable granularity [53] which allows for restructuring of the system’s functionality on basic block level (see Section 3.1) into larger partitioning objects.

In 1994, the authors Kalavade and Lee [54] published a fast algorithm for the partitioning problem. They addressed the coarse-grained mapping of processes onto an identical architecture (Figure 5) starting from a directed acyclic graph (DAG). The objective function incorporates several constraints on available silicon area (hardware capacity), memory (software capacity), and latency as a timing constraint. The global criticality/local phase (GCLP) algorithm is a greedy approach, which visits every process node once and is directed by a dynamic decision considering several cost functions.

The partitioning engine is part of the signal processing work suite Ptolemy [55] firstly distributed in the same year. This algorithm is compared to simulated annealing and a classical Kernighan-Lin implementation [56]. Its tremendous speed with reasonably good results is mentionable but in fact only a single partitioning solution is calculated in a vast search space of often a billion solutions. This work has been improved by the introduction of an embedded implementation bin selection (IBS) [57].

In the paper of Eles et al. [58] a tabu search algorithm is presented and compared to simulated annealing and Kernighan-Lin (KL). The target architecture does not differ from the previous ones. The objective function concentrates more on a trade-off between the communication overhead between processes mapped to different resources and reduction of execution time gained by parallelism. The most important contribution is the preanalysis before the actual partitioning starts. For the first time static code analysis techniques are combined with profiling and simulation to identify the computation intensive parts of the functional code. The static analysis is performed on operation level within the basic blocks. A suitability metric is derived from the occurrence of distinct operation types and their distribution within a process, which is later on used to guide the mapping to a specific implementation technology.

The paper of Vahid and Le [59] opened a different perspective in this research area. With respect to the architecture model a continuity can be stated as it does not deviate from the discussed models. The innovation in this work is the decomposition of the system into an access graph (AG), or call graph. From a software engineering point of view a system’s functionality is often described with hierarchical structures, memory (software capacity), and latency as a timing constraint. The partitioning algorithm mirrors exactly the concept given in Figure 5.

The authors extend the Kernighan-Lin heuristic to be applicable to this problem instance and put much effort in the exploitation of the access graph structure to greatly reduce the runtime of the algorithm. Indeed their approach yields good results on the examined real and random designs in comparison with other algorithms, like SA, greedy search, hierarchical clustering, and so forth. Nevertheless, the assignment of function nodes to the programmable component lacks a proper scheduling technique, and the decomposition of a usually block-based signal processing system into an access graph representation is in most cases very time consuming.

5.3. Combined partitioning and scheduling approaches

In the later nineties research groups started to put more effort into combined partitioning and scheduling techniques. The first approach of Chatha and Vemuri [60] can be seen as a further development of Kalavade’s work. The architecture consists of a programmable processor and a custom hardware unit, for example, an FPGA. The communication model consists of a RAM for hardware-software communication connected by a system bus, and both processors accommodate local memory units for internal communication. Partitioning is performed in an iterative manner on system level with the objective of the minimization of execution time while maintaining the area constraint.

The partitioning algorithm mirrors exactly the control structure of a classical Kernighan-Lin implementation adapted to more than two implementation techniques. Every time a node is tentatively moved to another kind of implementation, the scheduler estimates the change in the overall execution time instead of rescheduling the task subgraph. By this means a low runtime is preserved by paying reliability of their objective function. This work has been further extended for combined retiming, scheduling, and partitioning of transformative applications, that is, JPEG or MPEG decoder [61].

A very mature combined partitioning and scheduling approach for DAGs has been published by Wiangtong et al. [62]. The target architecture, which establishes the fundament of their work, adheres to the concept given in Figure 5.

![Figure 6: Code segment and corresponding access graph.](image-url)
The work compares three heuristic methods to traverse the search space of the partitioning problem: simulated annealing, genetic algorithm, and tabu search. Additionally, the most promising technique of this evaluation, tabu search, is further improved by a so-called penalty reward mechanism. This mechanism modifies the long-term memory, in which the information about most/least frequently visited neighbourhood solutions is stored. This solution yields the best results in terms of least processing time, shortest runtime of the algorithm, while meeting all the hardware area constraints.

The applied technique that is utilised to schedule a visited partitioning solution avoids any resource conflicts and is very fast. Not surprisingly, the technique is essentially a list scheduling technique. The process nodes are grouped a priori in a so-called precedence list, which is a rank-ordered sequence, where one sequence element, or one precedence level, contains all nodes with the same rank. As the nodes’ ranks remain always the same in the DAG, independent from the current partitioning, only the ordering of the processes within one precedence level has to be calculated for every new partitioning solution. An example of this approach can be seen in Figure 7. The nodes of the DAG are ordered according to its rank and their different color identifies a certain mapping to either software or hardware. The scheduling of these processes is shown on the right side of Figure 7. Most notably, this approach returns an exact system execution time to the partitioning engine in opposition to the estimation-based techniques described before. The scheduling is reasonably fast and collisions are avoided completely. However, the list scheduling fails to recognize situations in which one software process would enable many hardware processes running in parallel, whereas the instead preferred software process with the same rank does not have a single hardware successor, as the decision is based on the larger bus utilization.

The inspiration for the architecture model in the papers of Knerr et al. [18, 63] and the paper [62] originates from an industry-designed UMTS baseband receiver chip. Its abstraction (see Figure 8(a)) has been developed to provide a maximum degree of generality while being along the lines of the industry-designed SoCs in use. It consists of several (here two) DSPs handling the control-oriented functions, for instance, an ARM for the signalling part and a StarCore for the multimedia part, several hardware accelerating units (ASICs), for the data oriented and computation intensive signal processing, one system bus to a shared RAM for mixed resource communication, and optionally direct I/O to peripheral subsystems. In Figure 8(b) the simple modification towards the platform concept with one hardware processing unit (e.g., FPGA) has been established (cf. Figure 5).

Table 1 lists the access times for reading and writing bits via the different resources of the platform in Figure 8(b).

The graph representation of the system, which should be mapped onto this platform, is generally given in the form of a task graph. It is usually assumed to be a DAG describing the dependencies between the components of the system. The authors base their work on a graph representation for multirate systems, also known as synchronous data flow graphs [64]. This representation accomplished the backbone of renowned signal processing work suites, for example, Ptolemy [55] or SPW [65]. In Figure 9, a simple example of an SDF graph $G = (V, E)$ is depicted on the left,
shared system memory (RAM)

Local SW memory

DMA

DSP (ARM)

DSP (StarCore)

ASIC

ASIC

Direct I/O

System bus

 ASIC

 ASIC

 Local SW memory

 Shared system memory (RAM)

 DSP

 System bus

 Figure 8: Origin and modification of a general SoC platform abstraction.

Table 1: Maximum delays for read/write accesses to the communication resources.

| Communication                  | Read (bits/cycle) | Write (bits/cycle) |
|-------------------------------|-------------------|--------------------|
| Local software memory         | 128               | 256                |
| Local hardware memory         | 64                | 128                |
| Shared system bus             | 256               | 512                |
| Direct I/O                    | 1024              | 1024               |

Figure 9: Simple SDFG (left) and the decomposition into its SAG (right).

Note that all of the known approaches discuss task graph sets, which are homogeneous SDF graphs. This assumption leads to the very convenient situation of single invocations of every task. In this work, general SDF graphs with different input and output rates (see edge $e_2$ in Figure 9) are considered. A mapping of a task in the SDF graph from hardware to software causes a more complex situation since certainly all invocations of this task have to be moved to the DSP.

It has to be stated that the hardware/software partitioning process itself relies heavily on the metrics and measurements generated with methods in Section 3. Only in combination with a high fidelity of values like execution time, power consumption, and chip area, the partitioning algorithm is capable to return useful decisions very early in the design process.

In 2004 the first work of this group has been published solely regarding the high-level metrics generation and the partitioning problem [19]. The objective function for the hardware/software partitioning included estimations of execution time for software and hardware implementation and gate counts for the hardware implementation. The core algorithm to examine the search space was an adaptation of the Kernighan-Lin min-cut heuristic [56] for process graphs. Mainly because of the advancements of the underlying architecture abstraction and the more and more realistic communication model, scheduling issues came to the fore. The generalization to SDF graph systems with multiple invocation of a single process has been shown [63]. This work focused on a fast rescheduling method, which returns exact execution times for every single partitioning solution, which a partitioning algorithm evaluates during its run. The performance
of the so-called *local exploitation of parallelism (LEP)* algorithm has shown to be better than the aforementioned popular list scheduling techniques, like Hu-level scheduling [66] and earliest task first [67]. Most importantly LEP has been developed to preserve linear complexity, since it is aimed to be applied within partitioning algorithms that move incrementally through the search space (direct neighbourhood searches). This rescheduling method has been enhanced towards multicore architectures with many ASICs and *several* DSPs [63].

## 6. Virtual Prototyping

One of the main difficulties in the design of an embedded system, which consists of software and hardware parts, is that usually the design and testing of the strong related software parts have to wait until the hardware has been manufactured. Whereas hardware development, and especially its testing, is rather independent from the software development. Thus the design of an embedded system is a sequential process (Figure 10).

![Figure 10: Decrease of design time by virtual prototyping and automatic generation of virtual prototypes.](image)

The application of a so-called virtual prototype (VP), which is a software model of the hardware, allows for earlier start of the software development process and provides a platform for hardware architecture evaluation. In this technique, software reflects the behavior of the hardware and implements the software interface to the software, as it will be realized later in hardware. Such a VP can be implemented faster than the hardware itself, because all the hardware implementation details specific to the chosen technology can be neglected and high-level description languages can be used instead of hardware description languages (HDLs).

Generally, a complex SoC reflects a platform-based design (PBD), typically one or more DSPs surrounded by multiple hardware accelerators (HA). Those HAs are called VP components if they are used inside a VP simulation. The hardware/software partitioning process transforms a system-level specification into a heterogeneous architecture composed of hardware and software modules. This partitioning can be performed by a tool supported way (Section 5) or manually based on the experience of the designer.

Additionally different abstraction levels of a VP support a refinement process, which is especially needed for systems with high complexity being too large to support a consistent refinement in one major step. Several properties of abstraction layers are proposed for a VP, as they can be time related (e.g., untimed, timed functional, bus cycle accurate, and cycle true), data related (e.g., floating-point and fixed-point representation), and communication related (e.g., synchronous data flow, transaction level modeling (TLM) [68], and open core protocol international partnership OCP [69]).

In Figure 11 three different abstraction levels for a VP are shown: one VP (Figure 11(a)) for a first architecture evaluation, which is characterized by its properties (e.g., data rates, execution time, and power consumption); another one (Figure 11(b)) for software development, which achieves fast simulation performance by using a synchronous data flow description; and a third one (Figure 11(c)) for the cycle true cosimulation on register transfer level (RTL). The following sections explain those VP models in more detail.

### 6.1. Virtual Prototype for Architecture Exploration

A first evaluation of the system performance is achieved by a high-level characterisation of a VP component regarding only its features like, for example, input/output data rates, worst-case execution time (WCET), best-case execution time (BCET), and power consumption. Those properties can be combined to a cost function as shown in Section 3.3. Such a model provides a base for investigating communication bottlenecks on the bus system, power consumption of the system, and the meeting of real-time constraints.
6.2. Virtual prototyping for software development

In order to have high simulation speed together with a certain accuracy of the model, a VP component can have a cycle true interface to the bus system, whereas the implementation of the VP component is a high-level description (e.g., synchronous data flow description in COSSAP). An automatic generation method for a VP tailored for platform-based designs allows for a further decrease of development time [17, Figure 10]. Within such a method the algorithmic description is reused for the VP component (Figure 12).

Usually at algorithmic level the design information is free of communication details. Thus, in order to achieve communication of the VP components via the chosen platform, an object-oriented environment in C++ provides the functionality of functional blocks, ports, FIFOs, and scheduling. While this implementation implies a certain hardware platform, much emphasis is put on the fact that this platform is very general, a DSP with a common bus structure for its hardware accelerator units. The automatism is implemented for COSSAP designs based on GenericC descriptions only. However, the methodology is left open for supporting other descriptions, like SystemC.

The implementation of such a VP representation needs a simulation environment that allows for simulation of parallel processes. This is provided by a simulation interface, which is proposed by the virtual socket interface association (VSIA) [70]. In this approach a static scheduling is used, achieving faster simulation compared to the event-based simulation of SystemC and VHDL. Even compared to a plain C++ implementation, the VSIA implementation introduces negligible overhead.

The evaluation of a hardware-software system in real-time constraints additionally needs for an accurate description environment of software and hardware. Software and hardware need to be annotated with execution time estimations. Especially the software parts need to take into account the effects of interrupts, which can be modelled with TIPSY (TImed Parallel SYstem Modeling) C++ [71].

6.3. Virtual prototype for hardware development

As a last step, the internal behavior of the hardware accelerators has to be transformed to a cycle true model. This step is usually called high-level synthesis, investigated by many research projects [72], and also adopted to commercially available tools like CatapultC [73]. In that sense VP also supports a refinement-step-based design, which allows a much more consistent forgoing than switching between description languages.

A semiautomatic synthesis is achieved by the MASIC (MATH to ASIC) environment allowing for describing the control part of a system with the global control, configuration, and timing language (GLOCCT). Within this language the FIFOs have to be defined and connected manually. Functions, which are described in C, are used for a bit true and cycle true implementation. Afterwards the RTL code is generated automatically. A speedup in the order of 5 to 8 times compared to manually creation is achieved [13].

The paper of Valderrama [74] describes communication structures that are provided in a library. Such communication libraries implement simple handshake mechanisms up to layered networks. Nevertheless, a focus is needed on the hardware/software cosimulation process in order to increase efficiency and quality of the design process.

7. CONCLUSIONS

This paper presents an overview of modern techniques and methodologies for increasing the efficiency of the design process of embedded systems, especially in the wireless communications domain. The key factor influencing efficiency is the organization and structure of the overall design process. In an effort to increase efficiency, modern design methodologies tend towards unified descriptions of the system, with flexible and generalized tool integration schemes. Such environments can save most of the design effort invested in rewriting system descriptions, thus resulting in a streamlined design process. However, the most substantial increase in
efficiency comes from the automation of all individual steps in the design process through dedicated tools which are integrated into the design methodologies.

Firstly, design decisions at all levels of the design process are based on characteristics of the system, also called metrics. Hence, reliable, fast, and accurate analysis of the system is of paramount importance. The required metrics are efficiently obtained through static code analysis techniques, which offer increased speed by avoiding lengthy simulations, as well as the capability to estimate a wide range of required system properties.

Floating-point to fixed-point conversion is a critical step in the design process whose automation offers significant savings in design effort. Automation through dynamic (data-driven) techniques is most promising, allowing for complete replacement of the designer’s manual effort, while achieving the same quality of conversion results. Modern automation techniques offer optimization algorithms specifically suited for the conversion towards a particular implementation option, such as DSP code or custom hardware.

Hardware/software partitioning is another key step in the design process, for which a variety of automated techniques exists. The practical use and applicability of these implementations to industrial projects hinges heavily on the strength of the underlying algorithms, the degree to which the environment is tailored to the application domain, as well as the integration of the environment into an overall design methodology covering the entire design process.

Finally, virtual prototyping is a promising design technique for speeding up the design process, by allowing parallel development of both hardware and software components in the system. Modern design techniques for automated generation of virtual prototypes also exist, thus boosting the design productivity substantially.

ACKNOWLEDGMENT

This work has been funded by the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms.

REFERENCES

[1] Y. Neuvo, “Cellular phones as embedded systems,” in Proceedings of IEEE International Solid-State Circuits Conference (ISSCC ’04), vol. 1, pp. 32–37, San Francisco, Calif, USA, February 2004.
[2] J. Hausner and R. Denk, “Implementation of signal processing algorithms for 3G and beyond,” IEEE Microwave and Wireless Components Letters, vol. 13, no. 8, pp. 302–304, 2003.
[3] R. Subramanian, “Shannon vs. Moore: driving the evolution of signal processing platforms in wireless communications,” in Proceedings of IEEE Workshop on Signal Processing Systems (SIPS ’02), p. 2, San Diego, Calif, USA, October 2002.
[4] G. Moore, “Cramming more components onto integrated circuits,” Electronics Magazine, vol. 38, no. 8, pp. 114–117, 1965.
[5] International SEMATECH, “International Technology Roadmap for Semiconductors,” 1999, http://www.sematech.org.
[6] M. Rupp, A. Burg, and E. Beck, “Rapid prototyping for wireless designs: the five-ones approach,” Signal Processing, vol. 83, no. 7, pp. 1427–1444, 2003.
[7] R. L. Moigne, O. Pasquier, and J.–P. Calvez, “A graphical tool for system-level modeling and simulation with systemC,” in Proceedings of the Forum on Specification & Design Languages (FDL ’03), Frankfurt, Germany, September 2003.
[41] M. Stephenson, J. Babh, and S. Amarasinghe, “Bitwidth analysis with application to silicon compilation,” in Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI ’00), pp. 108–120, Vancouver, BC, Canada, June 2000.

[42] D. Menard, D. Chillet, F. Charot, and O. Sentieys, “Automatic floating-point to fixed-point conversion for DSP code generation,” in Proceedings of International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES ‘02), pp. 270–276, Grenoble, France, October 2002.

[43] C. F. Fang, R. A. Rutenbar, and T. Chen, “Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs,” in Proceedings of IEEE/ACM International Conference on Computer-Aided Design, pp. 275–282, San Jose, Calif, USA, November 2003.

[44] S. Kim, K.-I. Kum, and W. Sung, “Fixed-point optimization utility for C and C++ based digital signal processing programs,” IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 45, no. 11, pp. 1455–1464, 1998.

[45] Y. Cao and H. Yasuura, “Quality-driven design by bitwidth optimization for video applications,” in Proceedings of IEEE/ACM Asia and South Pacific Design Automation Conference, pp. 532–537, Kitakyushu, Japan, January 2003.

[46] C. Shi and R. W. Brodersen, “An automated floating-point to fixed-point conversion methodology,” in Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP ’03), vol. 2, pp. 529–532, Hong Kong, April 2003.

[47] MathWorks Simulink, http://www.mathworks.com/products/simulink.

[48] J. Hromkovic, Algorithms for Hard Problems, Springer, New York, NY, USA, 2nd edition, 2003.

[49] C. Mehlhüffer, F. Kaltenberger, M. Rupp, and G. Humer, “A scalable rapid prototyping system for real-time MIMO OFDM transmission,” in Proceedings of the 2nd IEEE/EURASIP Conference on DSP Enabled Radio, Southampton, UK, September 2005.

[50] R. Ernst, J. Henkel, and T. Benner, “Hardware-software cosynthesis for microcontrollers,” IEEE Design & Test, vol. 10, no. 4, pp. 64–75, 1993.

[51] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” Science, vol. 220, no. 4598, pp. 671–680, 1983.

[52] D. Henkel, J. Herrman, and R. Ernst, “An approach to the adaption of estimated cost parameters in the COSYMA system,” in Proceedings of the 3rd International Workshop on Hardware/Software Codesign (CODES ’94), pp. 100–107, Grenoble, France, September 1994.

[53] J. Henkel and R. Ernst, “Hardware/software partitioner using a dynamically determined granularity,” in Proceedings of the 34th Annual Conference on Design Automation (DAC ’97), pp. 691–696, Anaheim, Calif, USA, June 1997.

[54] A. Kalavade and E. A. Lee, “Global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem,” in Proceedings of the 3rd International Workshop on Hardware/Software Codesign (CODES ’94), pp. 42–48, Grenoble, France, September 1994.

[55] E. A. Lee, “Overview of the ptolemy project,” Tech. Rep., University of Berkeley, Berkeley, Calif, USA, March 2001. http://ptolemy.eecs.berkeley.edu.

[56] B. Kernighan and S. Lin, “An efficient heuristic procedure in partitioning graphs,” Bell System Technical Journal, vol. 49, no. 2, pp. 291–307, 1970.
M. Holzer received his Dipl.-Ing. degree in electrical engineering from the Vienna University of Technology, Austria, in 1999. During his diploma studies, he worked on the hardware implementation of the LonTalk protocol for Motorola. From 1999 to 2001, he worked at Frequentis in the area of automated testing of TETRA systems and afterwards until 2002 at Infineon Technologies on ASIC design for UMTS mobiles. Since 2002, he has a research position at the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms at the Vienna University of Technology.

B. Knerr studied communications engineering at the Saarland University in Saarbrücken and at the University of Technology in Hamburg, respectively. He finished the Diploma thesis about OFDM communication systems and graduated with honours in 2002. He worked for one year as a Software Engineer at the UZR GmbH & Co KG, Hamburg, on image processing and 3D computer vision. In June 2003, he joined the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms at the Vienna University of Technology as a Ph.D. candidate. His research interests are HW/SW partitioning, multicore task scheduling, static code analysis, and platform-based design.

P. Belanović received his Dr. Tech. degree in 2006 from the Vienna University of Technology, Austria, where his research focused on the design methodologies for embedded systems in wireless communications, virtual prototyping, and automated floating-point to fixed-point conversion. He received his M.S. and B.E. degrees from Northeastern University, Boston, and the University of Auckland, New Zealand, in 2002 and 2000, respectively. His research focused on the acceleration of image processing algorithms with reconfigurable platforms, both in remote sensing and biomedical domains, as well as custom-format floating-point arithmetic.

M. Rupp received his Dipl.-Ing. degree in 1988 at the University of Saarbrücken, Germany, and his Dr. Ing. degree in 1993 at the Technische Universität Darmstadt, Germany, where he worked with Eberhardt Häusler on designing new algorithms for acoustical and electrical echo compensation. From November 1993 until July 1995, he had a postdoctoral position at the University of Santa Barbara, California with Sanjit Mitra, where he worked with Ali H. Sayed on a robustness description of adaptive filters with impacts on neural networks and active noise control. From October 1995 until August 2001, he has been a member of the technical staff in the Wireless Technology Research Department of Bell Labs where he has been working on various topics related to adaptive equalization and rapid implementation for IS-136, 802.11, and UMTS. He is presently a Full Professor for digital signal processing in mobile communications at the Technical University of Vienna. He was an Associate Editor of IEEE Transactions on Signal Processing from 2002 to 2005, he is currently an Associate Editor of JASP EURASIP Journal of Applied Signal Processing, and of JES EURASIP Journal on Embedded Systems, and he is elected as AdCom Member of EURASIP. He authored and coauthored more than 200 papers and patents on adaptive filtering, wireless communications and rapid prototyping, as well as automatic design methods.
Fixed-Point Configurable Hardware Components

Romuald Rocher, Daniel Menard, Nicolas Herve, and Olivier Sentieys

ENSSAT, Université de Rennes 1, 6 rue de Kerampont, 22305 Lannion; IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes, France

Received 1 December 2005; Revised 4 April 2006; Accepted 8 May 2006

To reduce the gap between the VLSI technology capability and the designer productivity, design reuse based on IP (intellectual properties) is commonly used. In terms of arithmetic accuracy, the generated architecture can generally only be configured through the input and output word lengths. In this paper, a new kind of method to optimize fixed-point arithmetic IP has been proposed. The architecture cost is minimized under accuracy constraints defined by the user. Our approach allows exploring the fixed-point search space and the algorithm-level search space to select the optimized structure and fixed-point specification. To significantly reduce the optimization and design times, analytical models are used for the fixed-point optimization process.

Copyright © 2006 Romuald Rocher et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. INTRODUCTION

Advances in VLSI technology offer the opportunity to integrate hardware accelerators and heterogeneous processors in a single chip (system-on-chip) or to obtain FPGAs with several millions of gate equivalent. Thus, complex signal processing applications can be now implemented in embedded systems. For example, the third generation of mobile communication system requires implementing in a digital platform the wide band code division multiple access (WCDMA) transmitter/receiver, a turbo-decoder, and different codecs for voice (AMR), image (JPEG), and video (MPEG4). The application time-to-market requires reducing the system development time and thus, high-level design tools are needed. To bridge the gap between the available gate count and the designer productivity, design reuse approaches [1] based on intellectual properties (IP) blocks have to be used. The designer assembles predesigned and verified block to realize the architecture.

To reduce the cost and the power consumption, the fixed-point arithmetic has to be used. Nevertheless, the application fixed-point specification has to be determined. This specification defines the integer and fractional word length for each data. The data dynamic range has to be estimated for computing the data binary-point position corresponding to the data integer word length. The fractional part word length depends on the operators word length. For efficient hardware implementation, the chip size and the power consumption have to be minimized. Thus, the goal of this hardware implementation is to minimize the operator word length as long as the desired accuracy constraint is respected.

From an arithmetic point of view, the available IP blocks are limited. In general, the IP user can only configure the input and output word length and sometimes the word length of some specific operators. Thus, the fixed-point conversion has to be done manually by the IP user. This manual fixed-point conversion is a tedious, time-consuming, and error-prone task. Moreover, the fixed-point design search space cannot be explored easily with this approach.

Algorithm level optimization is an interesting and promising opportunity in terms of computation quality. For a specific application, like a linear time-invariant filter, different structures can be tested. These structures lead to different computation accuracy. As shown in the experiment presented in Section 5, for a same architecture the signal-to-quantization-noise ratio (SQNR) can vary from 30 dB to 62 dB for different structures. Thus, this search space must be explored and the adequate structure must be chosen to reduce the chip size and the power consumption. This algorithm level search space cannot be explored easily with available IPs without a huge exploration time. Indeed, the computation accuracy evaluation is based on fixed-point simulations.

In this paper, a new kind of IP optimized in terms of fixed-point arithmetic is presented. The fixed-point conversion is automatically achieved through the determination of the integer and fractional part word lengths. These IPs are configurable according to accuracy constraints influencing
the algorithm quality. The IP user specifies the accuracy constraint and the operator word lengths are automatically optimized. The optimal operator word lengths which minimize the architecture cost and respect the accuracy constraint are then searched. The accuracy constraint can be determined from the application performances through the technique presented in [2].

The computation accuracy is evaluated with analytical approaches to reduce dramatically the optimization time compared to simulation-based approach. Moreover, our analytical approach allows exploring the algorithm level search space in reasonable time.

In this paper, our method is explained through the least mean square (LMS), delayed-LMS (DLMS) applications, and infinite impulse response (IIR) filter. The paper is organized as follows. After a review of the available IP generators, our approach is presented in Section 3. The fixed-point optimization process is detailed in Section 4. Finally, the interest of our approach is underlined with several experiments in Section 5. In each section, the LMS/DLMS application case is developed and the experiments are detailed with IIR applications also.

2. RELATED WORKS

To provide various levels of flexibility, IP cores can be classified into three categories corresponding to hard, soft, or firm cores [1]. Hard IP cores correspond to blocks defined at the layout level and mapped to a specific technology. They are often delivered in masked-level designed blocks (e.g., GDSII format). These cores are optimized for power, size, or performance and are much more predictable. But, they depend on the technology and lead to minimum flexibility. Soft IP cores are delivered in the form of synthesizable register transfer (RT) or behavioral levels hardware description languages (e.g., VHDL, Verilog, or SystemC) code and correspond to the IP functional descriptions. These cores offer maximum flexibility and reconfigurability to match the IP user requirements. Firm IP cores are a tradeoff between the soft and hard IP cores. They combine the high performances of hard cores and the flexibility of soft cores but are restricted in terms of genericity.

To obtain a sufficient level of flexibility, only soft cores are considered in this paper. For soft cores, FPGA vendors often provide a library of classical DSP functions. For most of these blocks, different parameters can be set to customize the block to the specific application [3]. Especially, the data word length can be configured. The user sets these different IP parameters, and the complete RTL code is generated for this configuration. Nevertheless, the link between the application performances and the data word length is not immediate. To help the user to set the IP parameters, some IP providers supply a configuration wizard (Xilinx generator, Altera Mega-Function). The different data word lengths for the IP can be restricted to specific values and all the word lengths cannot be tested. In these approaches, the determination of the binary-point position is not automated and must be done manually by the IP user. This task is tedious, time consuming, and error prone.

The different tools provided by AccelChip integrate an IP generator core (AccelWare) [4] and assist the user to achieve the floating-point to fixed-point conversion [5, 6]. The effect of finite word length arithmetic can be evaluated with Matlab fixed-point simulations. The data dynamic range is automatically evaluated by using the interval arithmetic and the binary-point positions are computed from these information. Then, a fixed-point Matlab code is generated to evaluate the application performances. Thus, the user sets manually the data word length with general rules and modifies them to explore the fixed-point design space. This approach helps the user to convert into fixed-point but does not allow exploring the design space by minimizing the architecture cost under accuracy constraint.

This approach has been extended in [7, 8] to minimize the hardware resources by constraining the quantization error into a specified limit. This optimization is based on an iterative process made up of data word length setting and fixed-point simulations with Matlab. First of all, a coarse grain optimization is applied. In this case, all the data have the same word length. When the obtained solution is closed to the objective, a fine grain optimization is achieved to get a better solution. The different data can have their own word length. This fine grain optimization cannot be applied directly because it will take a long time to converge.

This accuracy evaluation approach suffers from a major drawback which is the time required for the simulation [9]. The simulations are made on floating-point machines, and the extra-code used for emulating the fixed-point mechanisms increases the execution time between one and two orders of magnitude compared to a traditional simulation with floating-point data types [10]. For obtaining an accurate estimation of the noise statistic parameters, a great number of samples must be taken for the simulation. This great number of samples, combined with the increase of execution time due to the fixed-point mechanisms emulation, leads to long simulation time.

This approach becomes a severe limitation when these methods are used in the process of data word length optimization where multiple simulations are needed to explore the fixed-point design space. To obtain reasonable optimization times, heuristic search algorithms like the coarse-grain/fine-grain optimization are used to limit this design space.

Moreover, these approaches test a unique structure for an application. This tool does not explore the algorithm level search space to find the adequate structure which minimizes the chip size or the power consumption for a given accuracy constraint.

3. IP GENERATION METHODOLOGY

3.1. IP generation flow

The aim of our IP generator is to provide an RTL-level VHDL code for an IP with a minimal architecture cost. The architecture cost corresponds to the architecture area, the energy consumption, or the power consumption. This IP generator,
presented in Figure 1, is made up of three modules corresponding to the algorithm level exploration, the fixed-point conversion, and the back end which generates the RTL level VHDL code.

The aim of the algorithm level exploration module is to find the structure which leads to minimal architecture cost and fulfils the computation accuracy constraints. This module tests the different structures for a given application, to select the best one in terms of architecture cost. For each structure, the fixed-point conversion process searches the specification which minimizes the architecture cost $C(\vec{b})$ under an accuracy constraint where $\vec{b}$ is the vector containing the data word lengths of all variables. The conversion process returns the minimal cost $C_{\text{min}}(\vec{b})$ for the structure which is selected.

The main part of the IP generator corresponds to the fixed-point conversion process. The aim of this module is to explore the fixed-point search space to find the fixed-point specification which minimizes the architecture cost under accuracy constraints. The first stage corresponds to the data dynamic range determination. Then, the binary-point position is deduced from the dynamic range to ensure that all data values can be coded to prevent overflow. The third stage is the data word length optimization. The architecture cost $C(\vec{b})$ (area, energy consumption) is minimized under an accuracy constraint as expressed in the following expression:

$$\min (C(\vec{b})) \quad \text{with} \quad \text{SQNR}(\vec{b}) \geq \text{SQNR}_{\text{min}}, \quad (1)$$

where $\vec{b}$ represents all data word length and $\text{SQNR}_{\text{min}}$ the accuracy constraint. The optimization process requires evaluating the architecture cost $C(\vec{b})$ and the computation accuracy SQNR($\vec{b}$) defined through the signal-to-quantization-noise ratio (SQNR) metric. This metric corresponds to the ratio between the signal power and the quantization noise power due to finite word length effect. These two processes are detailed in Sections 4.1 and 4.2. To determine the parallelism level $K$ which allows respecting the throughput constraint, the architecture execution time is evaluated as explained in Section 3.3.2. Once the different operator word lengths and the parallelism level are defined, the VHDL code representing the architecture at the RTL level is generated.

### 3.2. User interface

The user interface allows setting the different IP parameters and constraints. The user defines the different parameters associated with the application. For example, for linear-time-invariant filters, the user specifies the transfer function. For the least-mean-square (LMS) adaptive filter, the filter size or the adaptation step can be specified.

For the fixed-point conversion, the dynamic range evaluation and the computation accuracy require different information on the input signal. The user gives the dynamic range and test vectors for the input signals.

For generating the optimized architecture, the user defines the throughput and the computation accuracy constraints. The throughput constraint defines the output sample frequency and is linked to the application sample frequency. Different computation accuracy constraints can be considered according to the application. For the LMS, the output SQNR is used. For linear-time-invariant filters, three
constraints are defined. They correspond to the maximal frequency response deviation $|\Delta H_{\text{max}}(\omega)|$ due to finite word length coefficient, the maximal value of the power spectrum for the output quantization noise $|B_{\text{max}}(\omega)|$, and the SQNR minimal value $\text{SQNR}_{\text{min}}$.

### 3.3. Architecture model

#### 3.3.1. Generic architecture model

Architecture performances depend on algorithm structure. Thus, a generic architecture model is defined for each kind of structure associated with the targeted algorithm. This model can be configured according to the parameters set by the IP user. This architecture model defines the processing and control units, the memory, and the input and output interfaces. The processing unit corresponds to a collection of arithmetic operators, registers, and multiplexors which are inter-connected. These operators and the memory are extracted from a library associated with a given technology. The control unit is defined with a finite state machine. To explore the search space in reasonable time, analytical models are used for evaluating the architecture cost, the architecture latency, and the parallelism level.

**LMS/DLMS architecture**

In this part, the architecture of the IP LMS example is detailed. The least-mean-square (LMS) adaptive algorithm, presented in Figure 2, estimates a sequence of scalars $y_n$ from a sequence of $N$-length input sample vectors $x_n$ [11]. The linear estimate of $y_n$ is $w_n^T x_n$, where $w_n$ is an $N$-length weight vector which converges to the optimal vector $w_{\text{opt}}$. The vector $w_n$ is updated according to the following equation:

$$w_{n+1} = w_n + \mu x_n e_{n-D} \quad \text{with} \quad e_n = y_n - w_n^T x_n,$$

where $\mu$ is a positive constant representing the adaptation step. The delay $D$ is null for the LMS algorithm and different from zero for the delayed-LMS (DLMS).

The architecture model presented in Figure 3 consists of a filter part and an adaptation part to compute the new coefficient value. To satisfy the throughput constraint, the filter part and the adaptation part can be parallelized. For the filter part, $K$ multiplications are used in parallel and for the adaptation part $K$ multiply-add (MAD) patterns are used in parallel. The different data word lengths $b$ in this architecture are $b_f$ for the input filter, $b_m$ for the filter multiplier output, $b_h$ for the filter coefficient, and $b_e$ for the filter output.

To accelerate the computation, the processing is pipelined and the operators work in parallel. Let $T_{\text{cycle}}$ be the cycle-time corresponding to the clock period. This cycle-time is equal to the maximum value between multiplier and adder latency. The filter part is divided into several pipeline stages. The first stage corresponds to the multiply operation. To add the different multiplication results, an adder based on a tree structure is used. This tree is made up of $\log_2(K)$ levels. This global addition execution is pipelined. Let $L_{\text{ADD}}$ be the number of additions which can be executed in one cycle-time. Thus, the number of pipeline stages for the global addition is given by the following expression:

$$M_{\text{ADD}} = \left\lceil \frac{\log_2(K)}{L_{\text{ADD}}} \right\rceil \quad \text{with} \quad L_{\text{ADD}} = \left\lceil \frac{T_{\text{cycle}}}{L_{\text{ADD}}^1} \right\rceil,$$

where $L_{\text{ADD}}^1$ is the 2-input adder latency. The last pipelined stage for the filter part corresponds to the final accumulation.

The adaptive part is divided into three pipeline stages. The first one is for the subtraction. The second stage corresponds to the multiplication and the final addition completes the last stage.

#### 3.3.2. Parallelism level determination

To satisfy the throughput constraint specified by the IP user, several operations have to be executed in parallel. The
The parallelism level is determined such that the architecture latency is lower than the throughput constraint. To solve this inequality, the operator latency has to be known and this latency depends on the operator word length. Firstly, the operator word lengths are optimized with no parallelism. The obtained operator word lengths allow determining the operator latency. Secondly, the parallelism level is computed from the throughput constraint, and then the operator word lengths are optimized with the parallelism level real value.

**LMS/DLMS architecture**

In this part, the architecture of the IP LMS example is detailed. The LMS architecture is divided into two parts corresponding to the filter part and the adaptation part. The execution time of the filter part is obtained with the following expression:

\[
T_{\text{FIR}} = \frac{N}{K} T_{\text{cycle}} + M_{\text{ADD}} T_{\text{cycle}} + T_{\text{cycle}}.
\]  

The execution time of the adaptation part is given by

\[
T_{\text{Adapt}} = T_{\text{cycle}} + \frac{N}{K} (T_{\text{cycle}}) + T_{\text{cycle}}.
\]

The system throughput constraint depends on the chosen algorithm. For the LMS algorithm, the sampling period \( T_e \) must satisfy the following expression:

\[
T_{\text{FIR}} + T_{\text{Adapt}} < T_e.
\]

Even if the delayed-LMS algorithm has a slower convergence speed compared to the LMS Algorithm, as the error is delayed, the filter part and the adaptation part can be computed in parallel which gives to the DLMS a potentially higher execution frequency. The constraints become

\[
T_{\text{FIR}} < T_e, \quad T_{\text{Adapt}} < T_e.
\]

The parallelism level is obtained by solving analytically expressions (6) and (7).

### 3.4. Dynamic range evaluation

Two kinds of method can be used for evaluating the data dynamic range of an application. The dynamic range of a data can be computed from its statistical parameters obtained by a floating-point simulation. This approach estimates accurately the dynamic range with the signal characteristics. Nevertheless, overflow can occur for signals with different statistics. The second method corresponds to analytical approaches which allow computing the dynamic range from input data dynamic range. These types of methods guarantee that no overflow occurs but lead to more conservative results. Indeed, the dynamic range expression is computed in the worst case. The determination of the data dynamic range is obtained by the interval arithmetic theory [12]. The operator output data dynamic range is determined by its input dynamic using propagation rules. For linear time-invariant systems, the data dynamic range can be computed from the \( L_1 \) or Chebychev norms [13] according to the frequency characteristics of the input signal. These norms allow computing the dynamic range of a data in the case of nonrecursive and recursive structures with the help of the computation of the transfer function between the data and each input. For an adaptive filter like the LMS/DLMS, a floating-point simulation is used to evaluate the data dynamic range.

To determine the binary-point position of a data, an arithmetic rule is supplied. The binary-point position \( m_x \) of a data \( x \) is referenced from the most significant bit as presented in Figure 4. For a data \( x \), the binary-point position is obtained from its dynamic range \( D_x \) with the following relation:

\[
m_x = \lceil \log 2(D_x) \rceil \quad \text{with} \quad D_x = \max \left( \| x(n) \| \right).
\]

A binary-point position is assigned to each operator input and output and a propagation rule is applied for each kind of operators (adder, multiplier, etc.) [14]. Scaling operations are inserted in the graph to align the binary point position in the case of addition or to adapt the binary-point position to the data dynamic range.
4. FIXED-POINT OPTIMIZATION

The fixed-point specification is optimized through the architecture cost minimization under a computation accuracy constraint. In this section, the architecture cost and the computation accuracy evaluation are detailed and then, the algorithm used for the minimization process is presented.

4.1. Computation accuracy evaluation

The computation accuracy evaluation based on analytical approach is developed in this part. Quantization noises are defined and modeled, and their propagation through an operator is studied. Then, the expression of the output quantization noise power is detailed for the different kinds of systems.

4.1.1. Noise models

The use of fixed-point arithmetic introduces an unavoidable quantization error when a signal is quantized. A well-known model has been proposed by Widrow [15] for the quantization of a continuous-amplitude signal like in the process of analog-to-digital conversion. The quantization of a signal \( x \) is modeled by the sum of this signal and a random variable \( b_x \). This additive noise \( b_x \) is a stationary and uniformly distributed white noise that is not correlated with the signal \( x \) and the other quantization noises. This model has been extended for modeling the computation noise in a system resulting from the elimination of some bits during a cast operation (fixed-point format conversion), if the number of bits eliminated \( k \) is sufficiently high [16, 17].

These noises are propagated in the system through operators. These models define the operator output noise as a function of the operator inputs. An operator with two inputs \( X \) and \( Y \) and one output \( Z \) is under consideration. The inputs \( X \) and \( Y \) and the output \( Z \) are made up, respectively, of a signal \( x \), \( y \), and \( z \) and a quantization noise \( b_x \), \( b_y \) and \( b_z \). The operator output noise \( b_z \) is the weighted sum of the input noises \( b_x \) and \( b_y \) associated, respectively, with the first and second inputs of the operation. Thus, the function \( f_y \) expressing the output noise \( b_z \) from the input noises is defined as follows for each kind of operation \( y \in \{+,-,\times,\div\} \) [18]:

\[
b_z = f_y(x, y) = a^{(1)} \cdot b_x + a^{(2)} \cdot b_y.
\]

The terms \( a^{(1)} \) and \( a^{(2)} \) are associated with the noise located, respectively, on the first and second inputs of the operation. They are obtained only from the signal \( x \) and \( y \) and include no noise term. They are represented on Table 1.

| Operator | Value of \( a^{(1)} \) | Value of \( a^{(2)} \) |
|----------|------------------------|------------------------|
| \( Z = X \pm Y \) | 1                      | 1                      |
| \( Z = X \times Y \) | \( y \)               | \( x \)               |
| \( Z = X \div Y \) | 1                      | \( -\frac{x}{y^2} \)  |

4.1.2. Output quantization noise power

Let us consider, a nonrecursive system made up of \( N_e \) inputs \( x_j \) and one output \( y \). For multiple-output system, the approach is applied for each output. Let \( \hat{y} \) be the fixed-point version of the system output. The use of fixed-point arithmetic gives rise to an output computation error \( b_y \) which is defined as the difference between \( \hat{y} \) and \( y \). This error is due to two types of noise sources. An input quantization noise is associated with each input \( x_j \). When a cast operation occurs, some bits are eliminated and a quantization noise is generated. Each noise source is a stationary and uniformly distributed white noise that is uncorrelated with the signals and the other noise sources. Thus, no distinction between these two types of noise sources is done. Let \( N_q \) be the number of noise sources. Each quantization noise source \( b_{q_j} \) is propagated inside the system and contributes to the output quantization noise \( b_y \) through the gain \( v_j \) as presented in Figure 5. The analytical approach goal is to define the power expression of the output noise \( b_y \) according to the noise source \( b_{q_j} \) parameters and the gains \( v_j \) between the output and the different noise sources.

**Linear time-invariant system**

For linear time-invariant (LTI) systems, the gain \( a_i \) is obtained from the transfer function \( H_i(z) \) between the system output and the noise source \( b_{q_j} \). Let \( m_{v_j} \) and \( \sigma_{b_{q_j}}^2 \) be, respectively, the mean and the variance of the noise source \( b_{q_j} \). Thus, the output noise power \( P_{b_y} \) corresponding to the second-order moment is obtained with the following expression [13]:

\[
P_{b_y} = \sum_{i=0}^{N_q} \sigma_{b_{q_j}}^2 \cdot \frac{1}{2 \pi} \int_{-\pi}^{\pi} |H_i(e^{i\Omega})|^2 d\Omega + (m_{v_j} H_i(1))^2.
\]

This equation is applied to compute the output noise power of the IIR applications.
Nonlinear and nonrecursive systems

For the nonrecursive system, each noise \( b_{qi} \) is propagated through \( K_i \) operations \( a_k \), and leads to the \( b'_{qi} \) noise at the system output. This noise is the product of the \( b_{qi} \) input quantization noise source and the different \( a_k \) signals associated with each \( a_k \) operation involved in the propagation of the \( b_{qi} \) noise source.

\[
b'_{qi} = b_{qi} \prod_{k=1}^{K_i} a_k = b_{qi} v_i \quad \text{with} \quad v_i = \prod_{k=1}^{K_i} a_k.
\]  

(11)

For a system made up of \( N_q \) quantization noise sources, the output noise \( b_y \) can be expressed as follows:

\[
b_y = \sum_{i=0}^{N_q-1} b'_{qi} = \sum_{i=0}^{N_q-1} b_{qi} v_i.
\]  

(12)

Given that the \( b_{qi} \) noise source is not correlated with any \( v_i \) signal and with the other \( b_{qi} \) noise sources, the output noise power is obtained with the following expression [18]:

\[
P_{b_y} = \sum_{i=0}^{N_q} E(b'_{qi}^2)E(v_i^2) + 2 \sum_{i=0}^{N_q} \sum_{j>i} E(b_{qi})E(b_{qj})E(v_i v_j).
\]  

(13)

The computation of the noise power expression presented in (13) requires the knowledge of the statistical parameters associated with the noise sources \( b_{qi} \) and the signal \( v_i \).

Adaptive systems

For each kind of adaptive filter, an analytical expression of the global noise power can be determined. This expression is established using algorithm characteristics. For gradient-based algorithms, an analytical expression has been developed to compute the output noise power for the LMS/NLMS in [19] and for the affine projection algorithms (APA).

The LMS/DLMS algorithm noise model is presented in the rest of this part. The different noises are presented in Figure 2. With fixed-point arithmetic, the updated coefficient expression (2) becomes

\[
w'_{n+1} = w'_n + \mu e'_n \chi'_n + \gamma_n,
\]  

(14)

where \( \gamma_n \) is the noise associated with the term \( \mu e'_n \chi'_n \) and depends on the way the filter is computed. The error in finite precision is given by

\[
e'_n = y_n - w'_n \chi'_n - \eta_n
\]  

(15)

with \( \eta_n \) the global noise in the inner product \( w'_n \chi'_n \). This global noise is the sum of each multiplication output noise and output accumulation noise:

\[
\eta_n = \sum_{i=0}^{N-1} v_n(i) + u_n.
\]  

(16)

Moreover, a new term \( \rho_n \) is introduced:

\[
\rho_n = w'_n - w_n,
\]  

(17)

which is the \( N \)-length error vector due to the quantization effects on coefficients. This noise cannot be considered as the noise due to a signal quantization. The mean of each term is represented by \( m \) whereas \( \sigma^2 \) represents its variance and can be determined as explained in [17].

The study is made at steady-state, once the filter coefficients have converged. The noise is evaluated at the filter output. The power of the error between filter output in finite precision and in infinite precision is determined. It is composed of three terms:

\[
E(b_y^2) = E(\alpha_n^2 w_n^2) + E(\rho_n^2 \chi_n^2) + E(\eta_n^2).
\]  

(18)

At the steady state, the vector \( w_n \) can be approximated by the optimum vector \( w_{opt} \). So the term \( E(\alpha_n^2 w_n^2) \) is equal to \( |w_{opt}|^2 (m_n^2 + \sigma_n^2) \) with \( |w_{opt}|^2 = \sum w_{opt}^2 \).

The second term \( E(\rho_n^2 \chi_n^2) \) depends on the specific implementation chosen for the filter output computation (filtered data).

The last term is detailed in [19] and is equal to

\[
E(\rho_n \chi_n) = m_n^2 \sum_{i=1}^{N} \sum_{k=1}^{N} \frac{(R_{ki}^{-1})}{\mu^2} + \frac{N(\sigma^2 - m_n^2)}{2\mu}.
\]  

(19)

4.2. Architecture cost evaluation

The IP processing unit is based on a collection of operators extracted from a library. This library contains the arithmetic operators, the registers, the multiplexors, and memory banks for the different possible word lengths. Each library element \( l_i \) is automatically generated and characterized in terms of area \( A_l \) and energy consumption \( E_n \) using scripts for the synopsys tools. The IP architecture area \( (A_{IP}) \) is the sum of the different IP basic element area and the IP memory as explained.
Table 2: Different structure complexity for the 8th-order IIR filter.

| Kinds of structure | Cell order | Number of cells | Filter complexity |
|--------------------|------------|----------------|--------------------|
|                    |            |                | Addition | Multiplication | Storage | Coefficients |
| Direct form I      | 8          | 1              | 16       | 17            | 15      | 17           |
|                    | 4          | 2              | 16       | 18            | 15      | 18           |
|                    | 2          | 4              | 16       | 20            | 15      | 20           |
| Direct form II     | 8          | 1              | 16       | 17            | 12      | 17           |
|                    | 4          | 2              | 16       | 18            | 12      | 18           |
|                    | 2          | 4              | 16       | 20            | 12      | 20           |
| Transposed form II | 8          | 1              | 16       | 17            | 12      | 17           |
|                    | 4          | 2              | 16       | 18            | 12      | 18           |
|                    | 2          | 4              | 16       | 20            | 12      | 20           |

in expression (20). Let \( IP_{arch}\) be the set of all elements setting up the IP architecture. The different element area \( A_r_i \) depends on the element word length \( b_i \):

\[
A_{IP}(\vec{b}) = \sum_{i \in IP_{arch}} A_r_i(b_i). \tag{20}
\]

The IP energy consumption \( (\text{En}_{IP}) \) is the sum of different operation energy consumption executed to compute the IP algorithm output as explained in expression (21). These operations include the arithmetic operations, the data transfer between the processing unit and the memory (read/write). Let \( IP_{ops} \) be the set of all operations executed to compute the output. The different \( \text{En}_j \) operation energy consumption depends on the operation \( b_j \) word length

\[
\text{En}_{IP}(\vec{b}) = \sum_{j \in IP_{ops}} \text{En}_j(b_j). \tag{21}
\]

The \( \text{En}_j \) operation energy consumption is evaluated through simulations with synopsys tool. The mean energy is computed from the energy obtained for 10 000 random input data.

### 4.3. Optimization algorithm

For the optimization algorithm, operations are classified into different groups. A group contains the operations executed on the same operator, and thus these operations will have the same word length corresponding to the operator word length. All group word lengths are initially set to their maximum value. So the accuracy constraint must be satisfied. Then, for each group, the minimum value still verifying the accuracy constraint is determined, whereas all other group word lengths keep their maximum value. Next, all groups are set to their minimum value. The group for which the word length increases gives the highest ratio between accuracy constraint and the cost has its word length incremented until satisfying the accuracy constraint. Finally, all word lengths are optimized under the accuracy constraint.

### 5. EXPERIMENTS AND RESULTS

Some experiments have been made to illustrate our methodology and to underline our approach efficiency. Two applications have been tested, an 8th-order IIR filter and a 128-tap LMS/DLMS algorithm. The operator library has been generated from 0.18 \( \mu \)m CMOS technology. Each library element is automatically generated and characterized in terms of area and energy consumption using scripts for the synopsys tools.

#### 5.1. IIR filter

**5.1.1. IIR IP description**

In this part, an infinite impulse response filter (IIR) IP is under consideration. Let \( N_{IIR} \) be the filter order. Three types of structure corresponding to direct form I, direct form II, and transposed form II can be used [13]. For high-order filter, cascaded versions have to be tested. The cell order \( (N_{cell}) \) can be set from 2 to \( N_{IIR} \) if \( N_{IIR} \) is even or from 2 to \( \frac{(N_{IIR} - 1)}{2} \) if \( N_{IIR} \) is odd. The cell transfer functions are obtained with the factorization of the numerator and denominator polynomials. The complexity of the different IIR filter configurations are presented in Table 2 for an 8th-order IIR filter.

For a cascaded version of the IIR filter, the way that the different cells are organized is important. Thus, different cell permutations must be tested. For the 4th-order cell three different couples of cell transfer functions can be obtained and for each couple, two cell permutations can be tested. For the 2nd-order cell, 24 cell permutations are available. For this 8th IIR filter, the three different types of structure, the different cell orders, the different factorization cases, and the different cell permutations have been tested. It leads to 93 different structures for the same application.

**5.1.2. Fixed-point optimization**

**Coefficient word length optimization**

The fixed-point optimization process for the IIR filter is achieved in two steps. First, the coefficient word length \( b_h \) is optimized to limit the frequency response deviation \( |\Delta H(\omega)| \)
due to the finite word length coefficients as in the following equation:

$$\min (b_n) \quad \text{with} \quad \left| \Delta H(\omega) \right| \leq \left| \Delta H_{\max}(\omega) \right|. \quad (22)$$

The maximal frequency response deviation $\left| \Delta H_{\max}(\omega) \right|$ has been chosen such that the frequency response obtained with the fixed-point coefficient remains in the filter template. Moreover, the filter stability is verified with the fixed-point coefficient values. The results obtained for the 8th-order filter obtained with the cascaded and the noncascaded version are presented in Table 3. For high-order cell, the coefficients have greater value, so, more bits are needed to code the integer part. Thus, to obtain the same precision for the frequency response, the coefficient word length must be more important for high-order cell. To simplify, a single coefficient word length is under consideration. Nevertheless, to optimize the implementation, the coefficients associated with the same multiplication operator can have their own word length.

**Signal word length optimization**

The second step of the fixed-point optimization process corresponds to the optimization of the signal word length. The goal is to minimize the architecture cost under computation accuracy constraints. With this filter IP, two accuracy constraints are taken into account. They correspond to the power spectrum maximal value for the output quantization noise $|B_{\max}(\omega)|$ and the SQNR minimal value $\text{SQNR}_{\min}$

$$\min (C(\vec{b})) \quad \text{with} \quad \text{SQNR}(\vec{b}) \geq \text{SQNR}_{\min},$$

$$\left| B(\omega) \right| \leq \left| B_{\max}(\omega) \right|. \quad (23)$$

The computation accuracy has been evaluated through the SQNR for the 93 structures to analyze the difference between these structures. This accuracy has been evaluated with a classical implementation based on $16 \times 16 \rightarrow 32$-bit multiplications and 32-bit additions. For the noncascaded filter the quantization noise is important and leads to a nonstable filter. The results are presented in Figure 6 for the filter based on 2nd-order cells and 4th-order cells.

The results obtained for the transposed form II are those obtained with the direct form I with an offset. This offset is equal to 7 dB for the filter based on 2nd-order cells, and 9 dB for the filter based on 4th-order cells. These two filter types have the same structure except that the adder results are stored in memory for the transposed form II. This memory storage adds a supplementary noise source. Indeed, in the memory the data word lengths are reduced.

The analysis of the results obtained that for the direct form I and the direct form II, none of these two forms is always better. The results depend on the cell permutations. In the case of filter based on 2nd-order cells, the SQNR varies from 42 dB to 57 dB for the direct form I and from 50 dB to 61 dB for the direct form II. In the case of filter based on 4th-order cells, the SQNR varies from 30 dB to 45 dB for the direct form I and from 26 dB to 49.5 dB for the direct form II. Thus, the choice of filter form cannot be done initially and all the structures and permutations have to be tested.

| Cell order | Optimized coefficient word length |
|------------|----------------------------------|
| 8          | 24                               |
| 4          | 15                               |
| 2          | 13                               |

The IP architecture area and energy consumption have been evaluated for the different structures and for two accuracy constraints corresponding to 40 dB and 90 dB. The results are presented in Figure 7 for the power consumption and in Figure 8 for the IP architecture area. To underline the IP architecture area variation due to operator word length changes, the throughput constraint is not taken into account in these experiments in the case of IIR filter. Thus, the number of operators for the IP architecture is identical for the different tested structures.

As shown in Figure 6, the filters based on 4th-order cells lead to SQNR with lower values compared to the filters based on 2nd-order cells. Thus, these filters require operator with greater word length to fulfill the accuracy constraint. This phenomenon increases the IP architecture area as shown in Figure 8. Nevertheless, these filters require less operations to compute the filter output. This reduces the power consumption compared to the filters based on 2nd-order cells. Thus, the energy consumption is slightly greater for the filters based on 4th-order as shown in Figure 8.

The energy consumption is more important for the direct form I because this structure requires more memory accesses to compute each filter cell output. For the transposed form II and direct form II, the results are closed. The best solution is obtained for the transposed form II with 2nd-order cells and leads to an energy consumption of 1.6 nJ for the 40 dB accuracy constraint and to 2.7 nJ for the 90 dB accuracy constraint. As shown in Figure 6, this structure gives the lowest SQNR, thus, the operator word length is greater than that for the other forms. Nevertheless, this form consumes less energy because it requires less memory accesses than the direct form II. In the direct form II the memory transfers correspond to the read of the signal to compute the products with the coefficients and the memory write to update the delay taps. In the transposed form II, the memory accesses correspond only to the storage of the adder output.

Compared to the best solution the other structures based on 2nd-order cells lead to a maximal energy over cost of 36% for the 40 dB accuracy constraint and to 53% for the 90 dB accuracy constraint. For the structures based on 4th-order cells the maximal energy over cost is equal to 48% for the 40 dB accuracy constraint and to 71% for the 90 dB accuracy constraint.

The architecture area is more important for the filters based on 4th-order cells. As explained before, these filters lead to SQNR lower value compared to the filters based on 2nd-order cells. Thus, they require operators with greater word length to fulfill the accuracy constraints. The best solution obtained for the direct form II with 2nd-order cells leads to an architecture area of 0.3 mm$^2$ for the 40 dB accuracy.
constraint and to 0.12 mm$^2$ for the 90 dB accuracy constraint. Compared to this best solution the other structures based on 2nd-order cells lead to a maximal area over cost of 100% for the 40 dB accuracy constraint and to 40% for the 90 dB accuracy constraint. For the structures based on 4th-order cells the maximal energy over cost is equal to 225% for the 40 dB accuracy constraint and to 74% for the 90 dB accuracy constraint.

The best structure depends on the kind of architecture cost. The results are different for the architecture area and for the energy consumption. These results underline the opportunities offered by the algorithm level optimization to
optimize the architecture cost and the necessity to test the different structures and to select the best one.

For a given structure, the IP architecture area and power consumption evolve linearly according to the SQNR constraint. Between a 40 dB and 90 dB constraints, the energy varies from a factor 1.68 and the area from a factor 4. These results underline the necessity to choose the adapted accuracy constraint in order not to waste energy or area.

5.2. LMS and DLMS algorithms

The LMS and DLMS IP blocks have been used for different experiments to underline the necessity to optimize the operator and memory word lengths under an accuracy constraint. The IP users have to supply the reference and input signal. For the architecture generation, the throughput constraint $T_T$ and the accuracy constraint $\text{SQNR}_{\text{min}}$ must be defined.

The LMS and DLMS IP blocks have been tested for different values of the throughput constraint $T_T$ and the accuracy constraint $\text{SQNR}_{\text{min}}$. For each $T_T$ and $\text{SQNR}_{\text{min}}$ value, the operator and memory word lengths are optimized under the accuracy constraint. Then, the architecture is generated. The architecture area, the parallelism level, and the energy consumption are calculated by simulation and the results are presented, respectively, in Figures 9(a), 9(b), and 9(c) for a timing constraint between 60 ns and 170 ns and for an accuracy constraint between 30 dB and 90 dB.

The evolution of the energy consumption according to the accuracy constraint is presented in Figure 9(c). In our model, the energy consumption is independent of the throughput constraint and the power can be estimated by dividing the energy by the throughput period. The energy consumption varies from 4 nJ to 8.1 nJ for an accuracy constraint going from 30 dB to 90 dB. The energy is multiplied by two between these two accuracy constraints. This energy consumption increase is only due to the growth of the architecture element word length. To fulfill the accuracy constraint, the operator word length has to be increased.

The evolution of the IP architecture area according to the accuracy and throughput constraints are presented in Figures 9(a) and 9(b). For the minimal accuracy and throughput constraints, the architecture area is equal to 0.3 mm$^2$ with a parallelism level of $K = 4$. The architecture area climbs to 9 mm$^2$ with a parallelism level of $K = 20$ for the maximal accuracy and throughput constraints. The architecture area increases when the timing constraint decreases. Indeed, to respect this constraint, the parallelism level $K$ must be more important. More operators are needed and thus the processing unit area is increased. The architecture area increases with the accuracy constraint. High values of accuracy constraint require using operators and data with a greater word length. Thus, it increases the energy consumption and the area of the processing and memory units, and moreover, the operator latency. Thus to respect the timing constraint, the parallelism level $K$ must be more important and the processing unit area is increased.

Our results have been compared to a classical solution based on $16 \times 16 \rightarrow 32$-bit multiplications and 32-bit additions. This solution leads to an SQNR of 52 dB. The cost has been evaluated for the classical approach and our optimized approach for an accuracy constraint of 52 dB and with different timing constraints. The results are presented.
Architecture area $S = f(T_e, SQNR_{\text{min}})$

Parallelism level $K = f(T_e, SQNR_{\text{min}})$

Architecture area versus timing constraint

Figure 9: Experiment results: architecture area, energy consumption, and parallelism level for different values of accuracy and timing constraints.

In Figure 9(d). For the same computation accuracy our approach reduces the architecture area by 30% and the power consumption by 23%.

6. CONCLUSION

In this paper, a new kind of fixed-point arithmetic IP has been proposed. The architecture cost is minimized under one or several accuracy constraints. This IP is based on a library of operators and a generic architecture. The parallelism level is adapted to the timing and the computation accuracy constraints. The architecture cost and, especially, the computation accuracy are evaluated analytically to reduce dramatically the evaluation time. This technique allows exploring the fixed-point search space and thus to find the fixed-point specification which optimizes the implementation. Moreover, this analytical approach offers the opportunity to explore the algorithm level search space to find the optimal structure. The results presented for the 8th-order filter underline the interest of algorithm level optimization. The best structure can reduce significantly the IP area and the energy consumption compared to some inefficient structures. For a 128-tap LMS filter, compared to a classical approach, and for the same computation accuracy, the architecture area and the energy consumption are reduced, respectively, by 30% and 23%. With our approach, the user can optimize the tradeoff between the architecture cost, the accuracy, and the execution time.

REFERENCES

[1] M. Keating and P. Bricaud, Reuse Methodology Manual, Kluwer Academic, Norwell, Mass, USA, 3rd edition, 2000.
[2] D. Menard, M. Guitton, S. Pillement, and O. Sentieys, “Design and implementation of WCDMA platforms: challenges and trade-offs,” in Proceedings of the International Signal Processing Conference (ISPC ’03), pp. 1–6, Dallas, Tex, USA, April 2003.
[3] R. Seepold and A. Kunzmann, Reuse Techniques for VLSI Design, Kluwer Academic, Boston, Mass, USA, 1999.
[4] AccelChip., “Creating IP for system generator for DSP using matlab,” Tech. Rep. 95035, AccelChip, Milpitas, Calif, USA, 2004.
[5] P. Banerjee, D. Bagchi, M. Haldar, A. Nayak, V. Kim, and R. Uribe, “Automatic conversion of floating point MATLAB programs into fixed point FPGA based hardware design,” in Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM ’03), pp. 263–264, Napa Valley, Calif, USA, April 2003.

[6] R. Uribe and T. Cesar, “A methodology for exploring finite-precision effects when solving linear systems of equations with least-squares techniques in fixed-point hardware,” in Proceedings of the 9th Annual Workshop on High Performance Embedded Computing (HPEC ’05), Lincoln, Neb, USA, September 2005.

[7] S. Roy and P. Banerjee, “An algorithm for converting floating-point computations to fixed-point in MATLAB based FPGA design,” in Proceedings of the 41st Design Automation Conference (DAC ’04), pp. 484–487, San Diego, Calif, USA, June 2004.

[8] S. Roy and P. Banerjee, “An algorithm for trading off quantization error with hardware resources for MATLAB-based FPGA design,” IEEE Transactions on Computers, vol. 54, no. 7, pp. 886–896, 2005.

[9] L. De Coster, M. Adè, R. Lauwereins, and J. A. Peperstraete, “Code generation for compiled bit-true simulation of DSP applications,” in Proceedings of the 11th IEEE International Symposium on System Synthesis (ISSS ’98), pp. 9–14, Hsinchu, Taiwan, December 1998.

[10] H. Keding, M. Willems, M. Coors, and H. Meyr, “FRIDGE: a fixed-point design and simulation environment,” in Proceedings of the IEEE/ACM conference on Design, Automation and Test in Europe (DATE ’98), pp. 429–435, Paris, France, February 1998.

[11] S. Haykin, Adaptive Filter Theory, Prentice-Hall, Upper Saddle River, NJ, USA, 2nd edition, 1991.

[12] R. B. Kearfott, “Interval computations: introduction, uses, and resources,” Euromath Bulletin, vol. 2, no. 1, pp. 95–112, 1996.

[13] A. V. Oppenheim and R. W. Schafer, Discrete-Time Signal Processing, Prentice-Hall Signal Processing Series, Prentice-Hall, Upper Saddle River, NJ, USA, 2nd edition, 1999.

[14] D. Menard, D. Chillet, and O. Sentieys, “Floating-to-fixed-point conversion for digital signal processors,” EURASIP Journal on Applied Signal Processing, vol. 2006, Article ID 96421, 19 pages, 2006, special issue design methods for DSP systems.

[15] B. Widrow, “Statistical analysis of amplitude quantized sampled-data systems,” Transactions of the American Institute of Electrical Engineer: Part II: Applications and Industry, vol. 79, pp. 555–568, 1960.

[16] C. W. Barnes, B. N. Tran, and S. H. Leung, “On the statistics of fixed-point round off error,” IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 33, no. 3, pp. 595–606, 1985.

[17] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, “Truncation noise in fixed-point SFGs [digital filters],” IEEE Electronics Letters, vol. 35, no. 23, pp. 2012–2014, 1999.

[18] D. Menard, R. Rocher, P. Scalart, and O. Sentieys, “SQNR determination in non-linear and non-recursive fixed-point systems,” in Proceedings of the 12th European Signal Processing Conference (EUSIPCO ’04), pp. 1349–1352, Vienna, Austria, September 2004.

[19] R. Rocher, D. Menard, O. Sentieys, and P. Scalart, “Accuracy evaluation of fixed-point LMS algorithm,” in Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP ’04), vol. 5, pp. 237–240, Montreal, Quebec, Canada, May 2004.

Romuald Rocher received the Engineering degree and M.S. degree in electronics and signal processing engineering from ENSSAT, University of Rennes, in 2003. In 2003, he received the Ph.D. degree in signal processing and telecommunications from the University of Rennes. He is a member of the R2D2 (Reconfigurable Retargetable Digital Devices) Research Team at the IRISA Laboratory. His research interests include floating-to-fixed-point conversion and adaptive filters.

Daniel Menard received the Engineering degree and M.S. degree in electronics and signal processing engineering from the University of Nantes, Polytechnic School, in 1996, and the Ph.D. degree in signal processing and telecommunications from the University of Rennes, in 2002. From 1996 to 2000, he was a Research Engineer at the University of Rennes. He is currently an Associate Professor of electrical engineering at the University of Rennes (ENSSAT) and a member of the R2D2 (Reconfigurable Retargetable Digital Devices) Research Team at the IRISA Laboratory. His research interests include implementation of signal processing and mobile communication applications in embedded systems and floating-to-fixed-point conversion.

Nicolas Herve received the Engineering degree and M.S. degree in signal processing engineering and telecommunications from IFSIC, University of Rennes, in 2002. In 2002, he received the Ph.D. degree in signal processing and telecommunications from the University of Rennes. He is a member of the R2D2 (Reconfigurable Retargetable Digital Devices) Research Team at the IRISA Laboratory. His research interests include floating-to-fixed-point conversion, FPGA architecture, and high-level synthesis.

Olivier Sentieys received the Engineering degree and M.S. degree in electronics and signal processing engineering from ENSSAT, University of Rennes, in 1990, and the Ph.D. degree in signal processing and telecommunications from the University of Rennes, in 1993, and the Habilitation à Diriger des Recherches degree in 1999. He is currently a Professor of electrical engineering at the University of Rennes (ENSSAT). He is the Cohead of the R2D2 (Reconfigurable Retargetable Digital Devices) Research Team at the IRISA Laboratory and is a Cofounder of Aphycare Technologies, a company developing smart sensors for biomedical applications. His research interests include VLSI integrated systems for mobile communications, finite arithmetic effects, low-power and reconfigurable architectures, and multiple-valued logic circuits. He is the author or coauthor of more than 70 journal publications or published conference papers and holds 4 patents.