Application Specific Processors for Flexible Receivers

Oliver Schliebusch, Gerd Ascheid,
Andreas Wieferink, Rainer Leupers, Heinrich Meyr
Institute for Integrated Signal Processing Systems
Aachen University of Technology, Germany
schliebusch,ascheid,wieferink,leupers,
meyr@iss.rwth-aachen.de

Abstract

In order to fulfill the ever increasing processing constraints in the wireless communication domain without sacrificing flexibility, the techniques to design today's and tomorrow's System-on-Chips (SoCs) need to be carefully chosen. After analyzing the requirements of the 3rd generation wireless systems, this article reviews why Application Specific Instruction set Processors (ASIPs) offer a good compromise between flexibility and performance for most system components. A design methodology for optimally tailoring such customized processors to the wireless application domain is presented.

1. Algorithmic Complexity

Wireless applications have experienced an immense growth in algorithmic complexity, in particular, to provide higher data rates and more system capacity. Processing in today's UMTS receivers is significantly more complex than in a GSM receiver.

These processing steps differ significantly with respect to the processing power they require. It is however not just the number of samples per second which determines the required amount of processing. Rather, the number of samples per second must be multiplied with the number of operations to be performed per sample to obtain the right measure. As shown in Fig. 5, the processing requirements for the various UMTS processing steps are in the range between 1 MOPS (Million Operations Per Second) and 1000 MOPS.

Ideally, in order to support the variety of today's standards as well as to stay flexible for future wireless algorithms, one would want to implement every component in the processing chain in software, running on standard processors. Unfortunately, the required processing power would require high gate count and prohibitive power consumption when using general purpose (digital signal) processors.

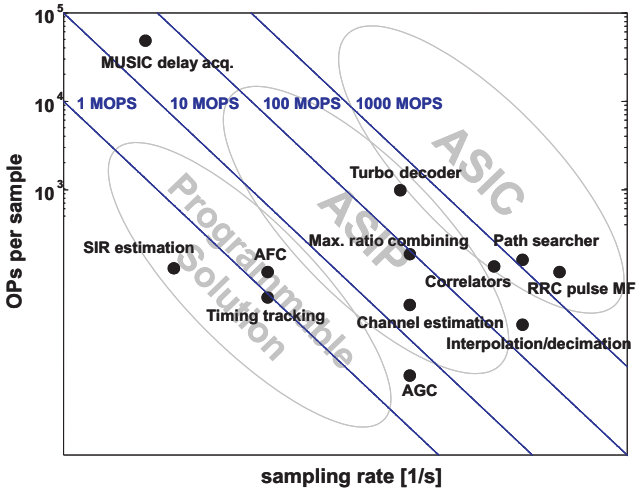

Figure 1. UMTS Receiver Complexity

Higher performance and better energy efficiency can only be obtained by customizing the hardware to the algorithm or at least the algorithm domain. One possibility is designing hardware specifically for a certain task. Such an Application Specific Integrated Circuit (ASIC) can be applied for tasks requiring more than 1000 MOPS, but once implemented in silicon, no flexibility is left to support additional or alternative algorithms.

Most of the UMTS baseband processing steps require a processing power of around 100 MOPS. This is an optimal range for execution on an Application Specific Instruction set Processor (ASIP), which is programmable, thus still offers the flexibility needed in today's fast-moving time, but more efficient because it is optimized for the processing task.

The next section introduces in more detail the techniques which can be applied to implement these different system components.

2. SoC Implementation Techniques

The ultimate goal is to optimally implement such complex wireless algorithms in a system on a single chip (System-on-Chip, SoC), leaving a maximum of flexibility for executing additional or modified wireless standards, yet being power and area efficient. For every component outlined in Fig. 5, an implementation technique exists which best fulfills the respective constraints.

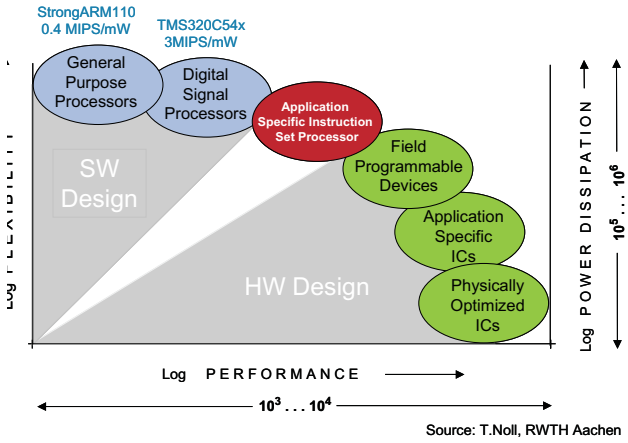

Figure 2. The Energy-Flexibility Gap

System modules which require lower processing power but should be kept flexible, are best implemented in software (Fig. 2, by [1]). Typically, this software runs on an embedded general purpose processor or microcontroller as provided, e.g., by ARM [2]. The first step in providing more processing power is moving to a domain specific processor. Digital signal processors (DSPs), like the Texas Instruments DSPs [3], have an architecture optimized for signal processing tasks and offer additional instructions for operations typically occurring in the signal processing domain, e.g. Multiply-Accumulate (MAC).

In contrast, system modules with very high processing requirements are usually fully implemented as hardware blocks. The highest performance is obtained by physically optimized ICs, but their design is a very tedious and error prone task, significantly increasing the development time. Even more, there is no flexibility left at all. A change in the specification may require a complete re-design of the entire system module.

The first step in providing more flexibility are Application Specific ICs (ASICs) which are automatically synthesized from a more abstract hardware model written using a textual HDL (Hardware Description Language) like VHDL and Verilog. By modifying this HDL model, changes can be put onto silicon much faster, but still, once the silicon is produced, subsequent modifications are not possible any more. Some flexibility can be obtained using configurable elements. Such a design style increases the verification effort and only offers limited flexibility.

Thus, both 'hardware' techniques do not offer sufficient opportunity of extending or just modifying the functionality of an existing chip. In order to solve this flexibility problem, field programmable devices like FPGAs (Field Programmable Gate Array) are offered by companies like Xilinx [4]. Since the functionality can be reconfigured anytime, this technique is much more flexible, but at the cost of performance. In order to obtain one reconfigurable gate, roughly ten real gates are necessary on the silicon, which leads to one order of magnitude worse values in chip area. In addition, the reconfigurable routing decreases the processing throughput due to its large delays.

As can be seen in Fig. 2, there is a relatively large gap left between the area covered by the software designer and that covered by the hardware designer. This is the field of the Application Specific Instruction set Processors (ASIPs). Since they can be optimally tailored for the considered wireless applications, they offer higher throughput compared to more general processors. On the other hand, since they are programmable rather than only configurable, they provide higher flexibility to the system designer than hardware solutions. Of course, designing and programming ASIPs requires a totally new design methodology, combining expertise in software as well as processor design. Recently, powerful tools have been developed, which enable the system designer to succeed doing this challenging task. The following section presents the LISA processor design environment [5], which has been developed at our Institute in Aachen and is now commercially available by CoWare.

3. LISA Processor Design Platform

The LISA processor design platform (LPDP) is an environment that allows the automatic generation of software development tools for architecture exploration, hardware implementation, software development tools for application design, and hardware-software co-simulation interfaces from one sole specification of the target architecture in the LISA language. Fig. 3 shows the components of the LPDP environment.

3.1. Hardware Designer Platform - for Exploration and Processor Generation

As indicated before, architecture design requires the designer to work in two fields (see Fig. 4): on one hand the de-

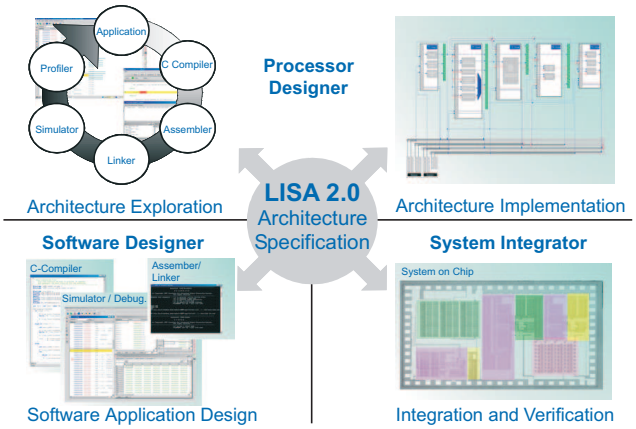

Figure 3. LISA based Processor Design

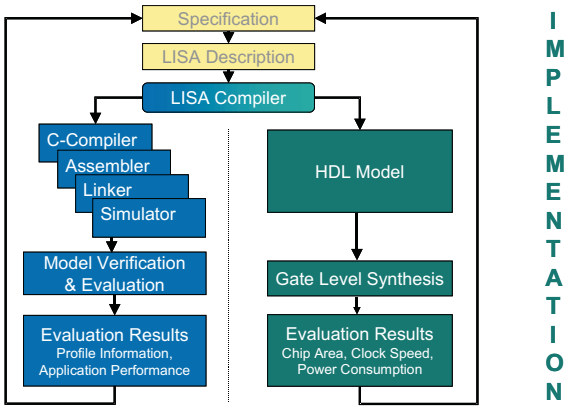

Figure 4. LISA Architecture Exploration Flow

velopment of the software part including compiler, assembler, linker, and simulator and on the other hand the development of the target architecture itself. The software simulator produces profiling data and thus support optimization of the instruction set, the performance of an algorithm and the required size of memories and registers. The required silicon area or power consumption can only be determined in conjunction with a synthesizable HDL model. To accommodate these requirements, the LISA hardware designer platform can generate the following tools:

- LISA language debugger for debugging the instruction-set with a graphical debugger frontend;
- Exploration C-compiler for the non-critical parts of the application;
- Exploration assembler which translates text-based instructions into object code for the respective programmable architecture;
- Exploration linker which is controlled by a dedicated linker command file;
- Instruction-set architecture (ISA) simulator providing extensive profiling capabilities, such as instruction execution statistics and resource utilization.

Besides the ability to generate a set of software development tools, synthesizable HDL code (both VHDL and Verilog) can be generated automatically from the LISA processor description [6]. This comprieses both, the control path as well as the data path. It is obvious that deriving both software tools and hardware implementation model from one single specification of the architecture in the LISA language has significant advantages: only one model needs to be maintained, changes on the architecture are applied automatically to the software tools and the implementation model, and the consistency problem among the software tools and between software tools and implementation model is avoided.

3.2. Software Designer Platform - for Software Application Design

To cope with the requirements of functionality and speed in the software design phase, the tools generated for this purpose are an enhanced version of the tools generated during the architecture exploration phase. The generated simulation tools are enhanced in speed by using the compiled simulation principle - where applicable - and are faster than the interpretive simulators often provided by architecture vendors by one to two orders of magnitude. The Just-In-Time Cache-Compiled (JIT-CC) Simulation principle [7] automatically uses already compiled simulation tables where possible and only switches back to interpretive simulation when necessary.

Besides the architecture specific assembler and linker, a fully featured C-Compiler can be generated automatically as well [8].

3.3. System Integrator Platform - for System Integration and Verification

Once the processor software simulator is available, it must be integrated and verified in the context of the whole system (SoC), which can include a mixture of different processors, memories, and interconnect components. In order to support the system integration and verification, the LPDP system integrator platform provides a well-defined application programmer interface (API) to interconnect the instruction-set simulator generated from the LISA specification with other simulators. The API allows to control the simulator by stepping, running, and setting breakpoints in the application code and by providing access to the processor resources [9]. Automatically generated wrappers allow integrating the fast LISA processor simulators into the final system context for a variety of SoC system simulation environments and multiple abstraction levels [10].

4 Example: an ASIP for DVB-T

DVB (digital video broadcasting) is an international standard for digital television. Terrestrial DVB (DVB-T) uses conventional terrestrial TV radio frequencies to distribute the digital video and audio signals. DVB-T enables mobile TV reception with high quality and can also be used for distribution of data broadcasts e. g. for low-cost mobile internet applications. The goal was the implementation of a flexible and power efficient digital receiver chip for DVB-T.

The ICORE architecture [11] is initially based on a mainly conventional DSP instruction set of a typical load/store Harvard-DSP architecture. This instruction set includes instructions for arithmetic and logical operations, data moves, and program flow control. This basic architecture has been implemented using the LISA design environment in order to tailor the architecture to the application. Certainly, the first ASIP specification in general as well as the the first ICORE realization are not optimal solutions with respect to execution time and power consumption and may violate given constraints.

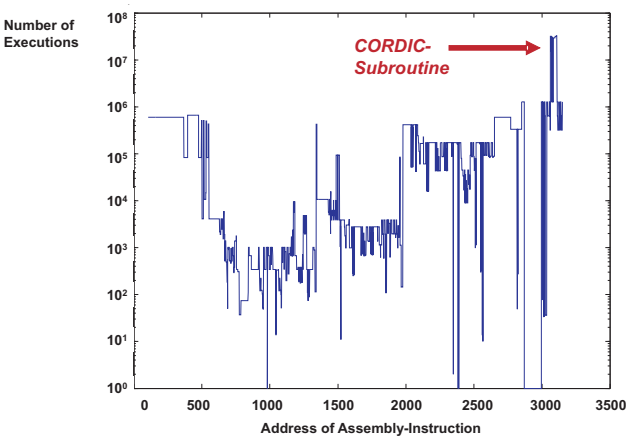

Figure 5. ICORE Application Profiling

When profiling the first version of the architecture and application, one measure is the execution count for the implemented algorithm. The execution count per assembly address is given in figure 5. The profiling clearly indicates the *CORDIC* subroutine as hot-spot of the application.

Power efficiency was one of the primary goals in the ICORE project. Generally speaking the following power optimization steps are applicable:

- high instruction coding density / application specific instruction encoding
- data-path optimization using application specific instructions and functional units
- local/global clock gating
- blocking logic to suppress wasteful logic and wire activity
- sleep-/doze modes of the processor
- software optimization

During the development of the ICORE architecture, it turned out, that application specific instruction set optimizations are most effective for the overall reduction in power consumption. According to figure 6 and to [12] the overall power consumption can be reduced by concurrently executing arithmetic operations. The energy consumed by the useful arithmetic operations is referred to as *intrinsic energy*. Obviously, this kind of energy consumption is independent from the operation schedule. The energy required to control the arithmetic operations, such as to load instructions, to decode instructions and to steer the control path is called *overhead energy*. This kind of energy is certainly strongly dependent on the instruction schedule.

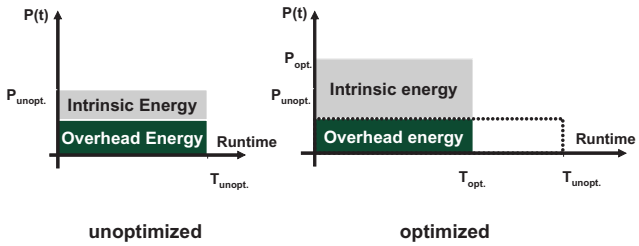

Figure 6. Power Optimization Principle

The concept of concurrent execution of heterogenous data-independent arithmetic operations evolved from an already known technique to increase throughput, the SIMD (Single Instruction Multiple Data) architectures, which perform the same arithmetic operations on multiple data values. This concept is especially applicable to matrices and vector calculations and therefore e.g. strongly used in Intel's Pentium MMX extension. The instruction set optimization targeted in ASIP design breaks with the necessity to perform the same arithmetic operations on multiple data, but executes arbitrary arithmetic operations on multiple data, so called *fused instructions*.

As already indicated before, the *CORDIC* subroutine has been identified as hot-spot in the ICORE application. The arithmetic instructions, which perform the *CORDIC* angle calculation hasve been reduced from five to only two instructions. Therefore, this optimization strongly reduces the energy consumption due to the reduced instruction-fetch and -decoding. Certainly, the arithmetic calculation remains unchanged, only the scheduling of arithmetic operations changed.

The overall power consumption was reduced by a factor of six, as shown in figure 7. This figure also clearly indicates, that a much higher reduction is achieved by the

instruction set optimization compared to traditional VLSI optimization techniques. In the ICORE project instruction set optimization has been performed manually, while current research activities focus on an automatic C-code analysis and instruction set generation.

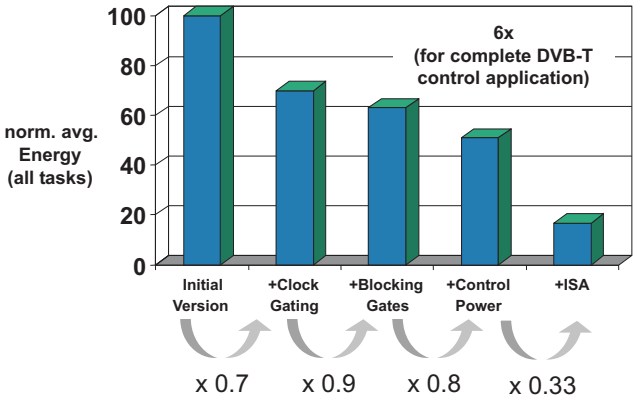

Figure 7. Overall Energy Reduction

The final ICORE is a typical 32 bit load-store Harvard processor architecture with single instruction issue rate. The core currently implements about 60 DSP instructions. These instructions are subdivided into 18 arithmetic instructions, 14 instructions for program flow control including zero overhead loop instructions. The remaining instructions are used for memory and interface I/O operations, bit manipulations and logical operations etc. Furthermore, highly optimized instructions support a fast *CORDIC* angle calculation, which have been derived from the optimization process described above.

5. Reconfigurable ASIPs

The computational building blocks in SoC design have been already elaborated in section 2. However, a new architecture type is recently evolving, combining the benefits of ASIPs and FPGAs, namely reconfigurable ASIPs (rASIPs). The flexibility of ASIPs is guaranteed, as different programs can be executed on the same hardware. Unfortunately, this flexibility mainly concerns the control flow, in fact the schedule of (arithmetic-)operations. Therefore, the data-path is fixed and does not provide any flexibility. Unfortunately, today's development in wireless communications does not only require flexibility in the domain of control-flow, but also in data-flow. FPGAs provide a high efficiency in realizing regular data-path structures. Therefore, the evolution of combining ASIPs with reconfigurable portions is quite natural.

The solutions currently existing in this field (Tab. 1) all are based on a fixed programmable architecture loosely or tightly coupled with a reconfigurable block, often called embedded FPGA (eFPGA). The eFPGA portion of the architectures represents the data-path of the architecture. Often this re-configurable data-path extends the also existing regular data-path. These solutions definitely provide flexibility in the domain of control-flow (via software) as well as data-path (via eFPGA). Unfortunately, these solutions do not use application specific optimizations, as the programmable architectures are often only domain specific or - even worse - general purpose solutions. In particular, the FPGA is based on very fine grained building blocks which leads to low throughput and low energy efficiency. Currently, there is no tool flow or methodology available which combines ASIPs and eFPGAs.

Bringing the advantages of ASIPs and re-configurable solutions together leads to SoC building blocks combining flexibility and performance in an unprecedented way in SoC design. The possibilities of developing new architectures are hughe. Three possibilities are shown in figure 8, demonstrating several eFPGA integration possibilities. Currently, there is no tool flow or methodology available which simultaneously covers the aspect of ASIP design and tightly coupled reconfigurable building blocks. Our future work will focus on developing an unified reconfigurable ASIP development methodology.

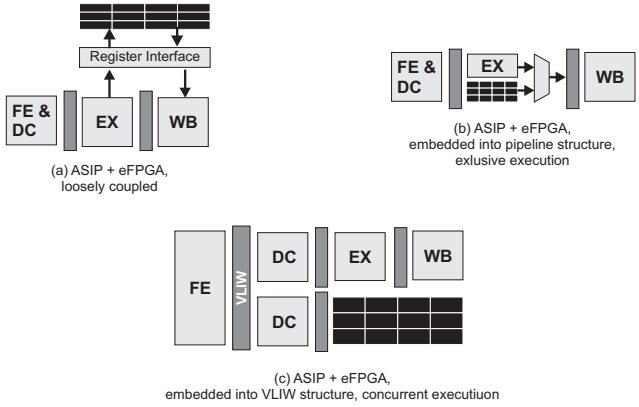

Figure 8. Combinations of ASIPs and eFPGAs

References

[1] Tobias Noll. *EECS, RWTH Aachen, http://www.eecs.rwth-aachen.de/.*

[2] ARM Processors. *ARM, http://www.arm.com/.*

[3] DSP Products. *Texas Instruments, http://www.ti.com/.*

[4] Xilinx Virtex FPGA Platforms. *Xilinx, http://www.xilinx.com/.*

[5] Hoffmann, A and Meyr, H. and Leupers, R. Architecture Exploration for Embedded Processors with LISA. In *Academic Publishers*. Kluwer Academic Publishers ISBN 1-4020-7338-0, Dec 2002.

[6] Schliebusch, O. and Chattopadhyay, A. and Kammler, D. and Ascheid, G. and Leupers, R. and Meyr, H. and Kogel, T. A Frame-

Name	Reference	Processor	Application	Coupling	reconfigurable block	dynamic
Splash	[13]	Sparc	Streaming	loosely	Xilinx 4010	no
Prism I	[14]	M68010	general	loosely	Xilinx 3090	no
Prism II	[15]	ARM29050	general	loosely	Xilinx 4010	no
Garp	[16]	MIPS	general	loosely	applic.specific	yes
Remarc	[17]	RISC	Multimedia	loosely	applic.specific	yes
PipeRench	[18]	Generic	Multimedia	loosely	applic.specific	yes
Pleiades	[19]	ARM	Multimedia	loosely	applic.specific	yes
Nano Processor	[20]	RISC	general	tightly	Xilinx 3000	no
PRISC	[21]	R2000	general	tightly	applic.specific	yes
DISC	[22]	CISC	general	tightly	National Clay31	partially
OneChip	[23]	DLX	general	tightly	Xilinx 4010	no
Chimaera	[24]	MIPS	Multimedia	tightly	applic.specific	yes
OneChip98	[25]	S-DLX	general	tightly	applic.specific	yes
Hybrid	[26]	SHARC	general	tightly	Xilinx 4000	no
Stretch 5000	[27]	XTensa Core	general	tightly	ISEF	no

Table 1. Coupling Processors with Reconfigurable Units

work for Automated and Optimized ASIP Implementation Supporting Multiple Hardware Description Languages. In *ASP-DAC*, Shanghai, China, Jan 2005.

[7] Nohl, Achim and Braun, Gunnar and Hoffmann, Andreas and Schliebusch, Oliver and Meyr, Heinrich and Leupers, Rainer. A Universal Technique for Fast and Flexible Instruction-Set Architecture Simulation. In *Proceedings of the Design Automation Conference (DAC)*, New Orleans, June 2002. ACM.

[8] Hohenauer, M. and Scharwaechter, H. and Karuri, K. and Wahlen, O. and Kogel, T. and Leupers, R. and Ascheid, G. and Meyr, H. and Braun, G. and van Someren, H. A Methodology and Tool Suite for C Compiler Generation from ADL Processor Models. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*, Paris, France, Feb 2004.

[9] Wieferink, A. and Kogel, T. and Nohl, A. and Hoffmann, A. and Leupers, R. and Meyr, H. A Generic Toolset for SoC Multiprocessor Debugging and Synchronisation. In *IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, The Hague (Netherlands), June 2003.

[10] Wieferink, A. and Kogel, T. and Braun, G. and Nohl, A. and Leupers, R. and Ascheid, G. and Meyr, H. A System Level Processor/Communication Co-Exploration Methodology for Multi-Processor System-on-Chip Platforms. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*, Paris, France, Feb 2004.

[11] T. Glökler and S. Bitterlich and H. Meyr. ICORE: A Low-Power Application Specific Instruction Set Processor for DVB-T Acquisition and Tracking. *13th IEEE workshop on Signal Processing Systems (ASIC/SOC'2000)*, September 2000.

[12] T. Glöckler and H. Meyr. *Design of Energy-Efficient Application-Specific Instruction Set Processors*. Kluwer Academic Publishers, 2004.

[13] Duncan A. Buell, Jeffrey M. Arnold, and Walter J. Kleinfelder. *Splash 2: FPGAs in a Custom Computing Machine*. IEEE Computer Society Press, 1996.

[14] Peter M. Athanas and Harvey F. Silverman. Processor reconfiguration through instruction-set metamorphosis. *IEEE Computer*, 26(3):11–18, 1993.

[15] M. Wazlowski, L. Agarwal, T. Lee, A. Smith, E. Lam, P. Athanas, H. Silverman, and S. Ghosh. PRISM-II compiler and architecture. In Duncan A. Buell and Kenneth L. Pocek, editors, *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 9–16, Los Alamitos, CA, 1993. IEEE Computer Society Press.

[16] J. R. Hauser and J. Wawrzynek. Garp: a mips processor with a reconfigurable coprocessor. In *Proceedings of the 5th IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM '97)*, page 12. IEEE Computer Society, 1997.

[17] Takashi Miyamori and Kunle Olukotun. Remarc (abstract): reconfigurable multimedia array coprocessor. In *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, page 261. ACM Press, 1998.

[18] Seth Copen Goldstein, Herman Schmit, Matthew Moe, Mihai Budiu, Srihari Cadambi, R. Reed Taylor, and Ronald Laufer. Piperench: A coprocessor for streaming multimedia acceleration. In *ISCA*, pages 28–39, 1999.

[19] Marlene Wan, Hui Zhang, Varghese George, Martin Benes, Arthur Abnous, Vandana Prabhu, and Jan Rabaey. Design methodology of a low-energy reconfigurable single-chip dsp system. *J. VLSI Signal Process. Syst.*, 28(1-2):47–61, 2001.

[20] Michael J. Wirthlin, Brad L. Hutchings, and Kent L. Gilson. The nano processor: A low resource reconfigurable processor. In Duncan A. Buell and Kenneth L. Pocek, editors, *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 23–30, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[21] R. Razdan and M. D. Smith. A high-performance microarchitecture with hardware-programmable functional units. In *Proceedings of the 27th Annual International Symposium on Microarchitecture*, pages 172–80, 1994.

[22] M. J. Wirthlin. A dynamic instruction set computer. In *Proceedings of the IEEE Symposium on FPGA's for Custom Computing Machines*, page 99. IEEE Computer Society, 1995.

[23] R. Wittig and P. Chow. OneChip: An FPGA processor with reconfigurable logic. In Kenneth L. Pocek and Jeffrey Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 126–135, Los Alamitos, CA, 1996. IEEE Computer Society Press.

[24] Scott Hauck, Thomas W. Fry, Matthew M. Hosler, and Jeffrey P. Kao. The chimaera reconfigurable functional unit. *IEEE Trans. Very Large Scale Integr. Syst.*, 12(2):206–217, 2004.

[25] Jeffrey A. Jacob and Paul Chow. Memory interfacing and instruction specification for reconfigurable processors. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, pages 145–154. ACM Press, 1999.

[26] Paul Graham and Brent E. Nelson. Reconfigurable processors for high-performance, embedded digital signal processing. In *Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications*, pages 1–10. Springer-Verlag, 1999.

[27] Stretch. *http://www.stretchinc.com*.