

Design of a Low Power Pre-Synchronization ASIP for Multimode SDR Terminals

Thomas Schuster, Bruno Bougard, Praveen Raghavan, Robert Priewasser,
David Novo, Liesbet Van der Perre, Francky Catthoor

IMEC
Kapeldreef 75
3000 Leuven, Belgium
{schuster,bougardb,ragha,priewasr,novo,vdperre,catthoor}@imec.be

Abstract. SDR enables cost-effective multi-mode terminals but still suffers from significant energy penalty when compared to dedicated hardware solutions. At system level, this energy bottleneck can be leveraged capitalizing on heterogeneous MPSOC platforms where specific engines are dedicated to classes of functions with similar computation characteristics and duty cycle. In burst-based communication as in IEEE802.11 or IEEE802.16, burst detection functions have high duty cycle and hence need an ultra low power implementation. Besides, programmability must be preserved to support multiple modes.

An ultra low-power pre-synchronization ASIP is designed targeting the IEEE802.11a/g/n and IEEE802.16e synchronization at 20MHz input rate. An IEEE802.16e synchronization (20MHz) can be carried out with an average power of 15.86mW. This corresponds to an effective energy efficiency of 115.89MOPS/mW (32-bit equivalent operations).

1 Introduction

The combination of the continuously growing variety of wireless standards and the increasing costs related to IC design and handset integration make implementation of wireless standards on reconfigurable radio platforms the only viable option in the near future. The Software Defined Radio (SDR) paradigm, according to which digital radio functionality is executed on widely reusable programmable platforms, is an effective way to provide the therefore necessary performance and flexibility.

If programmable from a high-level language (such as C), SDR enables cost-effective multi-mode terminals but still suffers from a significant energy penalty compared to dedicated hardware solutions. Hence, programmability and energy efficiency must be carefully balanced. To maintain energy efficiency at the level required for mobile device integration, abstraction may only be introduced where its impact on the total average power is sufficiently low or at those places where the resulting extra flexibility can be exploited by improved energy management (Targeted Flexibility, [3]). Many different architecture styles have already been proposed for SDR. Most of them are designed keeping in mind the important

characteristics of wireless physical layer processing: high data level parallelism (DLP) and data flow dominance [10, 14]. Targeted Flexibility and the fact that in wireless systems area can partly be traded for energy efficiency ask for heterogeneous MPSOC architectures [?, 6], in which the different tasks of a transmission scheme are implemented on specific engines providing just the necessary performance at minimum cost.

In practice, a radio standard implementation contains, next to modulation and demodulation, functionality for Medium Access Control (MAC) and, in case of burst-based communication, signal detection and time synchronization. The high DLP does not hold for the MAC processing which is, by definition, control dominated and should be implemented separately (e.g. on a RISC). Moreover, packet detection and coarse time synchronization have a significantly higher duty cycle than packet modulation and demodulation. They, therefore require a different flexibility/efficiency tradeoff.

In this work, we develop an instruction set processor specialized for signal detection and coarse time synchronization that will be part of a heterogeneous MPSOC platform for SDR [?]. We focus on the IEEE 802.11a/g/n and IEEE 802.16e standards, where packet-based radio transmission is implemented based on Orthogonal Frequency Division Multiplexing or Multiple-Access (OFDM(A)). The main design target is energy efficiency. Performance must be just sufficient to enable real time processing at the rates defined by the standards. In our reference implementations, packet detection and coarse synchronization account for less than 5% of the total code size. Hence, the effort for assembly programming is reasonable and compiler support not critical.

The remainder of the paper is structured as follows. In section 2, we analyze the targeted functionality and define the processor architecture. Section 3 focuses on design flow and implementation. Power estimation and optimization are discussed in section 4. Finally, conclusions are drawn in section 5.

2 Architecture Definition

Specific target applications for our design are signal detection and time synchronizations for IEEE 802.11a/g/n and IEEE 802.16e. Reason is that those functions have the highest duty cycle and dominate the standby power consumption.

Because one also wants to take provision for future standards such as 3GPP-LTE, an Application Specific Instruction-set Processor (ASIP) approach is preferred [9, 10]. For energy-aware implementation, special attention must be paid to the selection of the instruction-set, parallelization, storage elements (register files, memories) and interconnect. In this section, the most important architectural decisions are motivated.

2.1 Instruction-Set Selection

Usually, ASIP design starts with a careful analysis of the targeted algorithms. We applied a flow where profiling is performed on the application to obtain the

parts of the data flow graph which are activated often [2, 15]. Therefore, code for the targeted modes was written in Matlab and evaluated. Figure 1 illustrates the typical structure of a synchronization algorithm on the example of IEEE802.11a.

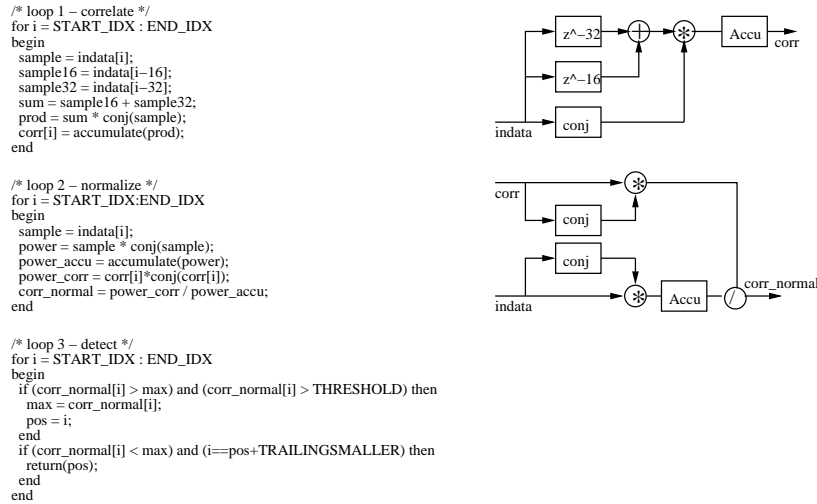


Fig. 1. Pseudo-code and data-flow in 802.11a synchronization

The code mainly consists of three loops. In the first two of them, the correlation in the input signal is explored. Here significant DLP is present that can be efficiently exploited by vector machines. In the third loop, one scans for a peak in the correlation result and compares it to a threshold. This is a more control oriented task. It can also be seen that a number of input samples (correlation window) need to be stored in memory.

The code for IEEE802.16e, shows very similar characteristics. Moreover, many common computational primitives can be identified, which suits the followed ASIP approach. However, compared to the IEEE802.11a synchronization, the algorithms for IEEE802.16e are far more computationally intensive (191 op/sample in average vs. 82 op/sample for IEEE802.11a). In terms of throughput, both applications are very demanding (up to 20Msamples/s). Throughput will be even higher for the multi-antenna operations in IEEE802.11n. Here it is intended to use 1 processor per antenna tile.

Fixed-point refinement shows that all computations for IEEE802.11a and IEEE802.16e can be done within 16bit signed precision. Moreover, all divisions can be removed by algorithmic transformations. The code has been optimized, including merging of the kernels into a single loop to improve data locality and reduce control. Afterwards, the code was vectorized and mapped to a number of pragmatically selected primitives.

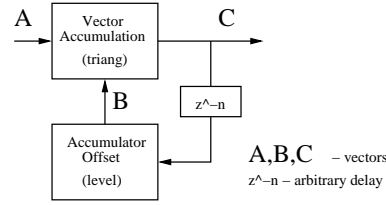
Table 1 shows the derived instruction set and the instruction count breakdown for the computations on a single input vector in our IEEE802.11a and IEEE802.16e detection loop. Vector size is a parameter. The number of vector operations per iteration is independent of the vector size. Bigger vectors will just reduce the number of iterations needed to process the whole input stream.

Table 1. Instruction-set and Statistic Coarse Time Synchronization

vector operation description		11a/g/n 16e	
vmov	fill vector with immediate	0	0
vcml	complex vector multiplication	5	16
vadd,vsub	vector add, sub	2	9
vasr,vlsl	shift vector elements right, left	1	0
vand,vor	and, or vectors	0	0
vtriang	accumulate across vector	2	5
vlevel	fill vector with vector element	2	5
vrotX	rotate vectors X positions	0	2
vcon	conjugate complex vector	2	9
vreal/vimag	real/imag vector components	0	0
generate vector description		11a/g/n 16e	
spread	fill vector with scalar elements	1	4
vload	load vector from address	2	5
pinld	load vector from i/o interface	1	1
evaluate vector description		11a/g/n 16e	
rgrep/igrep	extract real/imag value from vec	1	3
rmax/imax	max in real/imag vec elements	1	1
vstore	store vector to address	1	1
scalar operation description		11a/g/n 16e	
mov	move scalar register	2	5
mul	scalar multiplication	0	0
add, sub	scalar add, sub	4	1
lsl,asr	shift left, right	1	0
and,or,xor	and, or, xor scalar values	6	4
modi	modulo index calculation	0	0
pinst	write value to i/o interface	0	0
branch,jump	cond., uncond. branch	3	3

The behavior of the derived instructions is widely self-explaining (see description column in Table 1). Because all computations are done on complex samples, we decided to implement complex arithmetic in hardware. This has already been proven efficient for SDR processing [?]. The biggest challenge was the development of a mechanism for vector accumulation. The detection of the synchronization peak must be sample accurate. Hence, all correlation outputs

need to be evaluated. We therefore introduced a scheme that preserves the intermediate results of a vector accumulation (vtriang, vlevel - fig. 2) and instructions to extract maxima from vectors (rmax/imax).



$$C_k = \text{triang}(A_k, B_k) = \begin{pmatrix} a_0(k) + b_0(k) \\ a_0(k) + a_1(k) + b_1(k) \\ a_0(k) + a_1(k) + a_2(k) + b_2(k) \\ a_0(k) + a_1(k) + a_2(k) + a_3(k) + b_3(k) \end{pmatrix}$$

$$B_k = \text{level}(C_k * z^{-n}) = (c_3 * z^{-n} \ c_3 * z^{-n} \ c_3 * z^{-n} \ c_3 * z^{-n})^T$$

Fig. 2. Vector accumulation concept

2.2 Parallel Processing

It has already been demonstrated that in-order VLIW machines with capabilities for vector processing are most energy efficient for SDR [6, 14]. Following this approach, after the instruction set definition, one has to decide about the amount of parallel processing that is needed to guarantee real-time performance at minimum energy cost.

We first derive a target clock. The maximum achievable clock rate is limited to 200MHz by the available low power memories, which we intend to read and write without multi-cycle access or stalling the processor. Next, instruction and data-level parallelism are analyzed. From the application, it is observed that control and data processing can easily be parallelized. This yields separate scalar and vector slots. Since DLP is largely present in the considered algorithms, the amount of vectorization is decided first. Assuming a processor with a single vector slot and a clock rate of 200MHz, we would need a vectorization factor of at least 4.5 to process a zero-slack schedule of our most demanding application real-time (IEEE802.16e at 20MHz input rate). Realistic (close to zero-slack) schedule for a vectorization factor of 4 is made possible by using multiple vector slots with orthogonal instruction set. This also guarantees maximum utilization of the operators [?]. The ratio of vector operations to scalar operations is 46/28 in the IEEE802.16e and 23/16 in the IEEE802.11a kernel (Tab. 1). Accordingly, the target architecture should ideally be able to process 3 vector and 2 scalar operations in parallel. The design is therefore partitioned in three vector and two scalar instruction slots.

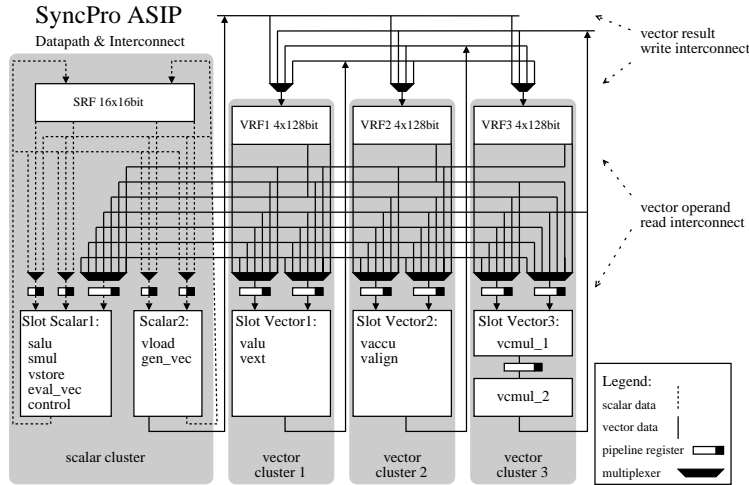


Fig. 3. SyncPro Vector Processor

Figure 3 shows the derived processor micro-architecture and the distribution of the instruction set. The instructions in the scalar slots operate on 16bit signed operands, the instructions in the vector slots on 4 complex samples in parallel (128bit). It is intuitive that further vectorization (256-bit or 512-bit) will lead to larger complexity in the interconnection network and hence is not considered [5]. Register file organization and interconnect will be discussed next.

2.3 Clustered Registerfiles and Interconnect

A shared multiported register files is typically a scalability bottleneck in VLIW structures and also one of the highest power consumers. Therefore, a clustered register file implementation is preferred.

We decided to implement 4 general purpose register files (Fig. 3). The scalar register file (SRF) contains 16 registers of 16 bit and has 4 read and 2 write ports. Because of its small word-width, the costs of sharing it amongst the FUs in the two scalar slots is rather low [?]. The vector side of the processor is fully clustered. Each of the three vector register files (VRF), holds 4 registers of 128 bit and has 3 read and 1 write port. Two of the read ports are dedicated to the FUs in a particular vector slot. The third one, is used for operand broadcasting (intercluster read) and can be accessed from all the other FUs in non-local issue slots and slot scalar1 (vector evaluation, vector store). Because each VRF has only one broadcast port, only one intercluster read per VRF can be carried out per cycle. Routing the vector operands is done via a *vector operand read interconnect*. Respectively, the *vector result write interconnect* is used to route computation results to the write ports of the VRFs. Each VRF write port can be written from all vector slots and from FUs in slot scalar2 (generate vector, vector load). The programmer is responsible to avoid access conflicts. The selected

interconnect provides almost as much flexibility as a central register file, but at a lower energy cost [12].

2.4 Memory and I/O

For all targeted modes, at least the correlation distance of the input signal needs to be buffered in memory. We therefore implemented a data scratchpad with a capacity of 256 vectors (4 kByte). In order to share interconnect, vector load and vector store are implemented in different units. The load FU is connected to the first scalar slot, which is capable of writing vectors. The store FU is assigned to the second scalar slot, from which vector operands can be read (Fig. 3). The L1 program memory has a capacity of 256 words of 96bit (3kByte). Both, L1 program memory and vector scratchpad are implemented as single port SRAM macros. To ease platform integration, the processor provides a number of I/O ports, specifically a blocking interface for reading vectors from an input stream.

2.5 Pipeline Model

Given the described architecture and the target technology in mind, it is now required to decide on the amount of pipelining that is needed to reach the targeted clock rate of 200MHz and seamlessly interface the instruction and data memory. We derived a pipeline model with two instruction fetch (FE1, FE2) and one instruction decode (DE) stage. Additionally, the units in the scalar slots and in the first and second vector slot have one execution stage (EX). The complex vector multiplier FU, in the third vector slot, has two execution stages (EX, EX2). The FE1 stage implements the addressing phase of the program memory. The instruction word is read in FE2. In stage DE, the instruction is decoded and the data memory is addressed. The decoder decides, which register file ports need to be accessed. Routing, forwarding and chaining of source operands are fully software controlled. Source operands are saved in pipeline registers at the end of DE and consumed by the activated FUs in the following cycle. Register files are written at the end of EX (or EX2).

3 Implementation

The proposed processor architecture has been implemented in a 90nm CMOS technology. Therefore, two major steps have been carried out. First, the processor was modelled in LISA (Language for Instruction-Set Architecture), capitalizing on the Processor DesignerTM toolsuite from Coware [8]. Then, RTL code was generated and used as a starting point for logic synthesis.

3.1 Instruction-Set Architecture Modelling

Coware Processor DesignerTM is a tool-suite for automated embedded processor design [1]. Our motivation for using it is that it enables the generation of software

development tools, such as assembler, linker and instruction-set simulator very early in the design process. So that, the processor micro-architecture can be co-optimized with the kernel software. Moreover, the tools offer strong support for platform integration (by generating wrapper for SystemC-based virtual platform modeling) and good-quality automated RTL code generation. Figure 4 illustrates the mentioned co-optimization strategy.

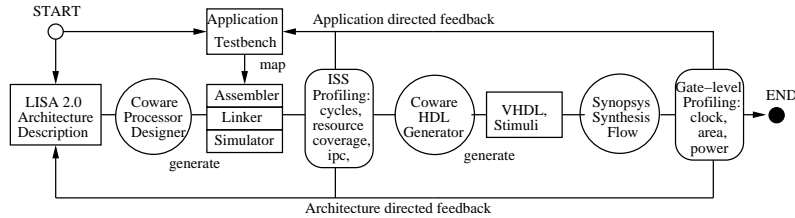


Fig. 4. Optimization methodology

We start with describing the instruction-set architecture in LISA. The resulting model is then iteratively refined into a pipelined micro-architecture representation, from which RTL code can be generated. Software and hardware are developed in parallel. The tools offer profiling functions, enabling fast application and architecture directed feedback, based on information about cycle count, IPC, code coverage and resource utilization. As soon the targets on those high level figures are met, the exploration is extended to architecture implementation level.

3.2 Logic Synthesis and Layout

Implementation cost assessment requires the knowledge of silicon area, achievable clock rate and power consumption. Therefore, a VHDL description is generated from Processor Designer (HDL Generator - fig. 4) and synthesized using a logic synthesis tool-chain (fig. 5). Compilation and optimization of the design are done with the Physical CompilerTM environment from Synopsys. For place & route and clock tree insertion, we utilize Cadence SOCEncounterTM.

Using the described flow, the design was synthesized for a 90nm general purpose process with a standard cell library for nominal V_t . We target a clock rate of 200MHz under worst case operating conditions ($V_{DD} = 0.9V$, $T = 125C$). Post-layout timing is back annotated proving 200MHz operation. The total chip area is $0.8mm^2$, including the 3KB program memory ($0.06mm^2$) and 4KB data memory ($0.07mm^2$) (fig. 6). Power estimation based on gate-level activity has been done with Synopsys PrimePowerTM. This will be detailed in the next section. Post-Layout back-annotation was only performed for the final design, to ensure timing closure. To get faster feedback on hardware cost during micro-architecture and implementation refinements, we use physical pre-layout estimates.

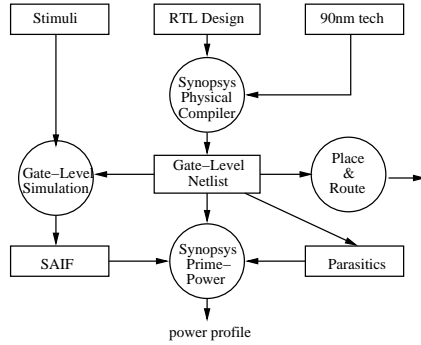


Fig. 5. Power Estimation Flow

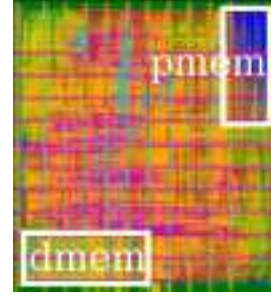


Fig. 6. Chip Layout

4 Power Estimation and Optimization

The designed processor is targeted to power constrained SDR. Hence, minimizing its power consumption is a major concern. So far, the power constraints have been considered during ISA and micro-architecture design. However, the power consumption of the automatically generated implementation (sec. 3.1) can still be optimized.

To obtain feedback on the power consumption, the automatically generated netlist, after preliminary placement and physical synthesis, is simulated with stimuli from the IEEE802.11a synchronization kernel (fig. 5). The left plot in figure 7 shows the resulting power profile ($V_{DD} = 1V$, $T = 25C$) for a single kernel iteration. The instruction- and data-dependent instantaneous power variation is averaged out since non-significant. The example refers to an input sample rate of 20MHz (5M vector/s with 4x vectorization) and a clock rate of 200MHz. The profile is characteristic for the processing of a real-time stream. At $0ns$, the first vector is read from the input interface. Consequently, the processor starts operating and consumes maximum power (P_{kernel}). After all necessary computations are done (t_{kernel}), the processor goes back to halt mode, waiting for new data (t_{iter}) to arrive. In this state, far less power is consumed (P_{wait}).

The optimization target is the reduction of the surface (energy) in the power profile. Therefore, we try to reduce the switching power by operand isolation. Another option is the reduction of the relatively high cell internal power, which can be tackled with clock gating. Theoretically, we could also exploit the duty cycle of the IEEE802.11a kernel, which is low compared to the worst case (IEEE802.16e), to reduce clock rate and V_{DD} . Unfortunately, the margins for voltage scaling in the considered technology are rather small. Moreover, the required voltage regulator would be a significant overhead relative to the small active area of the design.

With our optimizations the average power in the IEEE802.11a testbench could be reduced by 64%. While clock gating dramatically reduces the cell internal power, the effect of operand isolation on the switching power is neglectable

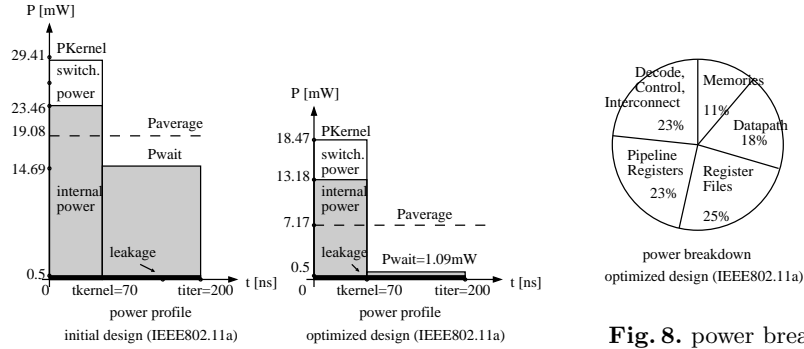


Fig. 7. power profile

small (7). The latter can be explained by the orthogonal instruction-set and the fact that only 18% of the total power are consumed in the datapath (Fig.8). Biggest power consumer are the flip-flops (48%). It is particularly noted that the pipeline registers (23%) consume almost the same share of the total power as the four general purpose register files (25%). Indeed, our design contains a high number of frequently accessed very wide (128bit) pipeline registers, to buffer source operands between the decode (DE) and execute (EX) pipeline stage (Sec. 2.5). Moving the register file read in the execute stage could eliminate those registers. However, it would have dramatic impact on the achievable clock rate.

The power profile for the optimized design, is depicted in the right part of figure 7. The energy spend for the processing of one input vector of four complex samples could be reduced from 3.97 to 1.43nJ. First power simulations for the IEEE802.16e synchronization show very similar results for P_{kernel} and P_{wait} . However, for the most demanding mode with 20MHz input rate, we estimate a duty cycle of 85%. Under this assumption, the energy for the computations on one input vector is 3.17 nJ.

5 Conclusion

Heterogeneous MPSOC platforms are considered to enable SDR implementation competitive in energy efficiency with dedicated hardware solutions. In such platforms, specific engines are dedicated to classes of functions that relate in their computation characteristics and duty cycle. In burst-based communication, signal detection functions have high duty cycle and hence need ultra low power implementation. Besides, programmability must be preserved to support multiple modes implementation. A ultra low-power pre-synchronization ASIP is designed targeting IEEE802.11a/g/n and IEEE802.16e. The processor delivers a theoretical maximum performance of 5 GOPS (32bit equivalent) at a peak power of 25 mW. Energy efficiency is hence 200 MOPS/mW (fully loaded). This is 2-4 times higher than other SDR solutions in comparable technologies.

IEEE802.11a synchronization (20 MHz) requires only 630 MOPS. The processor consumes 7.17 mW when executing this kernel (79.5 MOPS/mW). The more demanding IEEE802.16e synchronization (20MHz) requires 1838 MOPS. The estimated average power is then 15.86 mW, corresponding to 115.89 MOPS/mW. The processor can therefore be used to implement ultra low power packet detection, enabling energy aware multi-processor SDR platforms.

References

1. <http://www.coware.com/>.
2. P. Biswas, V. Choudhary, K. Atasu, L. Pozzi, P. Ienne, and N. Dutt. Introduction of local memory elements in instruction set extensions. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 729–734, 2004.
3. B. Bougard, L. Hollevoet, F. Naessens, A. Ng, T. Schuster, and L. V. der Perre. A low power signal detection and pre-synchronization engine for energy-aware software defined radio. In *SDRForum*, November 2006.
4. D. N. Bruna, B. Bougard, P. Raghavan, T. Schuster, K. Hong-Soek, H. Yang, and L. V. der Perre. Energy-performance exploration of a cga-based sdr processor. In *SDRForum*, November 2006.
5. H. DeMan. Ambient intelligence: Giga-scale dreams and nano-scale realities. In *Proc of ISSCC, Keynote Speech*, February 2005.
6. J. Glossner, M. Moudgill, and D. Iancu. The sandbridge sdr communication platform. In *SympoTIC*, October 2004.
7. M. Gries, K. Ketuzer, H. Meyr, and G. Martin. *Building ASIPS: The Mescal Methodology*. Springer, 2005.
8. A. Hoffmann, H. Meyr, and R. Leupers. *Architecture exploration for embedded processors with LISA*. Kluwer Academic Publishers, 2002.
9. P. Ienne and R. Leupers. *Customizable Embedded Processors: Design Technologies and Applications*. Morgan Kaufman, 2006.
10. Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner. SODA: A low-power architecture for software radio. In *Proc of ISCA*, 2006.
11. S. Mamidi, E. R. Blem, M. J. Schulte, J. Glossner, D. Iancu, A. Iancu, M. Moudgill, and S. Jinturkar. Instruction set extensions for software defined radio on a multithreaded processor. In *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, pages 266–273, 2005.
12. S. Rixner, W. J. Dally, B. Khailany, P. R. Mattson, U. J. Kapasi, and J. D. Owens. Register organization for media processing. In *HPCA*, pages 375–386, January 2000.
13. T. Schuster, B. Bougard, D. N. Bruna, V. Derudder, A. Hoffmann, and L. V. der Perre. Subword-parallel vliw architecture exploration for multimode software defined radio. In *SIPS*, October 2006.
14. K. van Berkel, F. Heinle, P. Meuwissen, K. Moermann, and M. Weiss. Vector processing as an enabler for software-defined radio in handsets from 3g+ wlan onwards. In *SDR Technical Conference*, 2004.
15. P. Yu and T. Mitra. Characterizing embedded applications for instruction-set extensible processors. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 723–728, 2004.