Hardware/Software Trade-offs for Advanced 3G Channel Coding *

Heiko Michel[1], Alexander Worm[1], Michael Münch[2] and Norbert Wehn[1]

[1]University of Kaiserslautern
Institute of Microelectronic System Design
Erwin-Schroedinger-Strasse
67663 Kaiserslautern, Germany
e-mail: {michel, worm, wehn}@eit.uni-kl.de

[2]Alcatel
ASIC Design Labs
Excelsiorlaan 44-46
1930 Zaventem, Belgium
e-mail: michael.muench@alcatel.de

Abstract

Third generation's wireless communications systems comprise advanced signal processing algorithms that increase the computational requirements more than ten-fold over 2G's systems. Numerous existing and emerging standards require flexible implementations ("software radio"). Thus efficient implementations of the performance-critical parts as Turbo decoding on programmable architectures are of great interest. Besides high-performance DSPs, application-customized RISC cores offer the required performance while still maintaining the aspired flexibility. This paper presents for the first time Turbo decoder implementations on customized RISC cores and compares the results with implementations on state-of-the-art VLIW DSPs. The results of our studies show that the Log-MAP performance is about 50 % higher than on an ST120, a current VLIW architecture.

1. Introduction

One of the main drivers for the development of next generation wireless communications systems is the increasing demand for high-rate data services. Today's 2G systems, e.g. GSM, were targeted to voice as primary service. The supported data-services are limited to low rates. Therefore, the computational complexity for the baseband signal processing in the receiver, comprising equalizing, channel decoding and source decoding is relatively low and can be implemented using one state-of-the-art DSP running at 80...150 MHz [10].

The 3G cellular wireless standards comprise advanced equalizing and coding algorithms. Especially Turbo-Codes [3, 9] as channel coding scheme pose great demands on the computing power of communication devices and infrastructure. Along with the higher throughput requirements (384 kbit/s, 2 Mbit/s in UMTS) the resulting computational complexity has raised by at least an order of magnitude compared to 2G. Dedicated hardware solutions for the baseband signal processing fulfill these requirements, but often lack the flexibility to support the various existing or even emerging future communication standards. Therefore efficient implementations on programmable architectures are of great interest ("software radio").

The actual requirements for the DSPs differ for the various fields of application: For hand-held devices, e.g., meeting the signal processing demands of one data channel while minimizing cost and power consumption is critical. In wireless infrastructure, several data channels have to be processed simultaneously, which exceeds the capabilities of a single signal processor even if a high-end device is used. Hence, the baseband receiver can be implemented either by a set of high-performance DSPs or by a combination of DSPs with dedicated IP cores.

It is argued in [4] that "the trends in decoding algorithms are moving from standard Viterbi towards more computationally-expensive algorithms like soft-output Viterbi algorithm (SOVA) and maximum a posteriori (MAP) algorithm. The implementation efficiency of these algorithms will become a differentiating factor for next generation wireless communications – particularly for those employing programmable DSP devices." Thus we focus in this paper on channel decoding, which is besides equalization the computationally most demanding part within the baseband receiver. The computational complexity of these algorithms is very high, e.g. up to about 6000 MOPS for a Log-MAP decoder for 3G, assuming a data-rate of 2 Mbit/s. Therefore it is necessary to identify the primary bottlenecks in pure software implementations.

*This work has been partially supported by the *Deutsche Forschungsgemeinschaft (DFG)* under Grant WE 2442/1-1 within the Schwerpunktprogramm *"Grundlagen und Verfahren verlustarmer Informationsverarbeitung (VIVA)"*.

Depending on the targeted system environment these bottlenecks are solved by using advanced VLIW DSPs, *application-customized RISC cores* or custom IP blocks. Current state-of-the-art DSPs, from low-cost to high-performance, support already the kernel operations of the Viterbi algorithm, used in 2G channel decoding, with dedicated instructions. For the MAP algorithm this support is actually lacking. The new approach of customizing and extending a RISC core blurs the borders between hardware and software. In contrast to a heterogeneous RISC/DSP or RISC/IP-block architecture the application specific hardware is coupled to the RISC core by an extension of its base instruction set. Thus application specific performance bottlenecks can be removed.

In this paper we show for the first time how the capabilities of application-customized RISC cores can be exploited to obtain high-performance implementations of 3G channel decoding algorithms. We present persuasive performance results that come along with lower power consumption and device costs when compared to state-of-the-art DSPs. Pure hardware implementations are not discussed in broader detail, since many publications already exist, e.g. [6, 12, 11].

The paper is structured as follows: Section 2 gives necessary analysis and transformation steps prior to any efficient implementation of an algorithm. Section 3 identifies the performance critical kernel operation of a Turbo decoder. Section 4 presents Turbo decoder implementations on classical DSPs, and identifies their bottlenecks. Section 5 introduces basic concepts of application-customized RISC cores and presents the results of a Turbo decoder implementation in terms of throughput and silicon area. Finally, Section 6 concludes this paper.

2. Methodology

To obtain efficient implementations of an algorithm, the implementation space has to be explored on multiple abstraction levels. Typically the starting point is a functional model written, e.g., in Matlab or C-language. Based on this specification four major steps have to be performed prior to any implementation:

1. Detailed algorithm analysis with respect to its computational complexity as well as data-transfer and storage demands,
2. extraction and exploration of the inherent algorithmic parallelism,
3. algorithmic transformation, e.g. to decrease the implementation complexity or increase the algorithmic parallelism and
4. quantization.

These steps are strongly interrelated. Some of them are bit-true algorithmic manipulations, others not, i.e. the algorithmic behavior is changed. Thus, in the case of channel decoding, the communication performance can be degraded. A carefull trade-off between implementation complexity and communication performance has to be carried out. The detailed discussion of all this steps is not focus of this paper. The interested reader is referred to, e.g., [7, 13]. Here we put emphasis on the first two steps.

Next to the algorithmic study and transformations, further steps towards an implementation have to be carried out, which strongly depend on the chosen target architecture. In this paper we discuss VLIW architectures and application-customized RISC cores in broader detail.

3. Turbo decoder kernel operation

A Turbo decoder consists of two soft-in/soft-out (SISO) decoders, each corresponding to one of the constituent codes. Decoding is done data-block wise in an iterative process with usually five to ten iterations. During this process the SISO decoders exchange so-called *a priori* information. In the following we assume the SISO decoders being realized by using the MAP algorithm, which outperforms the SOVA. To avoid numerical problems without degrading the decoding performance it is mandatory to implement the MAP algorithm in the Log-domain (Log-MAP), resulting in a key operation of $\ln(e^{\delta_1} + e^{\delta_2})$, which is in the context of Turbo decoders usually written as $\max^*(\delta_1, \delta_2)$. Using the Jacobian logarithm this term is transformed into:

$$\max^*(\delta_1, \delta_2) = \max(\delta_1, \delta_2) + \ln(1 + e^{-|\delta_2 - \delta_1|}), \quad (1)$$

where the approximation $\max^*(\delta_1, \delta_2) \approx \max(\delta_1, \delta_2)$ is assumed for the therefore sub-optimal Max-Log-MAP decoder. The pseudo-code for a standard (Max-)Log-MAP decoder is presented in Algorithm 1. The total number of states M is given by the constraint length K of the code $M = 2^{K-1}$, and the successors/predecessors (i, j) of an encoder state (m) depend on the component code structure. L is the size of the data block, which is defined in 3G standard in the range of $40 \ldots 5114$ bit.

During a forward recursion, the probability that the encoder reached state m at time-step k from the initial state is computed and the related forward path metrics, α, are stored. The probability that the encoder reached the final state from state m at time-step k is related to the backward path metrics β. The latter is produced during a backward recursion and is immediately consumed by the soft-output calculation. No data transfers to memory are necessary. Furthermore, the `softout_calculation` function needs the stored α-metrics to compute the confidence in the bit decision.

The MAP algorithm has, besides its high computational complexity, a great demand of data transfers due to the stor-

Algorithm 1 Log-MAP decoder

$\{M$: number of states$\}$
$\{\alpha(m \in \{1 \dots M\}, k \in \{0 \dots L-1\})$: forw. path metrics$\}$
$\{\beta_{\text{pred,curr}}(m \in \{1 \dots M\})$: backw. path metrics$\}$
$\{\Lambda_{\text{out}}(k \in \{1 \dots L\})$: produced likelihood values$\}$
$\{\Lambda_{\text{in}} = (\Lambda_{\text{sys}}, \Lambda_{\text{par}}, \Lambda_{\text{e}})$: consumed likelihood values$\}$

$\alpha(1,0) = 0$, $\beta_{\text{curr}}(1) = 0$
for $m = 2$ to M **do**
  $\alpha(m,0) = -\infty$, $\beta_{\text{curr}}(m) = -\infty$

for $k = 1$ to $L-1$ **do**
  for $m = 1$ to M **do**
    {Let states i and j be the predecessors of state m}
    $\alpha(m,k) = \text{recursion\_step}(\alpha(i,k-1), \alpha(j,k-1), \Lambda_{\text{in}})$
for $k = L$ to 1 **do**
  for $m = 1$ to M **do**
    {Let states i and j be the successors of state m}
    $\beta_{\text{pred}}(m) = \text{recursion\_step}(\beta_{\text{curr}}(i), \beta_{\text{curr}}(j), \Lambda_{\text{in}})$

  $\Lambda_{\text{out}}(k) = \text{softout\_calculation}(\alpha(1,k-1), \dots, \alpha(M,k-1), \beta_{\text{curr}}(1), \dots, \beta_{\text{curr}}(M)), \Lambda_{\text{in}})$
  for $m = 1$ to M **do**
    $\beta_{\text{curr}}(m) = \beta_{\text{pred}}(m)$

age of path metrics and read accesses to the branch metrics. Most of the complexity of this algorithm is in the `recursion_step` operation, which is invoked $2 \cdot L \cdot M$ times. Pairs of subsequent function calls use a common set of parameters and can be grouped together to a so-called *butterfly* arrangement, cf. Figure 1. This is well-known from Viterbi decoding as it cuts the memory accesses by 50 %. A butterfly discriminator n maps the butterfly number to the

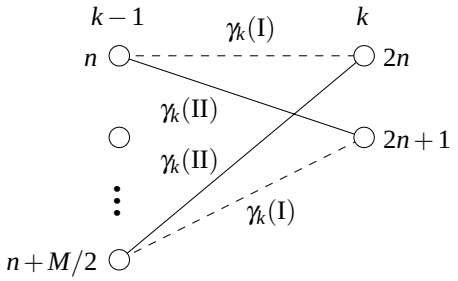

Figure 1. Log-MAP Butterfly

incorporated code states m: $n \in \{0 \dots (\frac{M}{2} - 1)\}$. If the input is binary, as assumed here, there are two possible next states branching out from every code state. Each branch is assigned with a branch metric γ, derived from Λ_{in}. The basic equations for updating the forward path metrics from time-step $k-1$ to k using the Log-MAP algorithm are:

$$\alpha_k(2n) = \max^*(\alpha_{k-1}(n) + \gamma_k(\text{I}), \alpha_{k-1}(n + M/2) + \gamma_k(\text{II})) \quad (2)$$

$$\alpha_k(2n+1) = \max^*(\alpha_{k-1}(n) + \gamma_k(\text{II}), \alpha_{k-1}(n + M/2) + \gamma_k(\text{I})) \quad (3)$$

Two implementation bottlenecks of the butterfly operation can be identified: First, a computational bottleneck based on the $\max^*$ operation (cf. eqn. (1)) which is called twice within every butterfly update and comprises the previously introduced correction term. State-of-the-art DSPs have special hardware support to implement the butterfly-operation of a Viterbi decoder (add-compare-select) in the most efficient way. Using a proper branch metric representation it can also be used for a Max-Log-MAP implementation, but not for the Log-MAP, because the correction term is not supported.

The second issue is the data-transfer bottleneck. While processing a butterfly, branch metrics are consumed, and updated path metrics are generated, which have to be stored into memory for later use. The required set of branch metrics has to be loaded once per time-step k and is shared by all butterflies processed in this step. Convolutional codes used for a Viterbi decoder have a greater constraint length, e.g. $K = 9$, compared to the codes used in Turbo coding ($K = 3$), leading to a much increased number of states M and therefore a greater number of butterflies processed in each time-step. This lowers the ratio of butterflies per branch metric load in the case of a Turbo decoder and tightens its data-transfer bottleneck.

4. Classical DSPs

In this section we present performance results of our Turbo decoder implementations on a 16-bit and a VLIW DSP. Due to the poor performance of DSP compilers, we implemented all our code in hand-optimized assembly language, i.e. scheduling, functional unit and register assignment were done manually. Especially the register assignment and data storage scheme are highly dependent on the core architecture, and resolving the data-transfer bottleneck is a challenging task.

4.1. 16-bit fixed-point DSPs

The core architecture of a classical 16-bit fixed point DSP usually comprises one ALU/MAC unit, a program memory and two data memories, each with separate address and data busses. ALU unit and address generation unit (AGU) operate independently from each other, thus allowing parallel execution of instructions to a limited extent, e.g. ALU operations and memory-register transfers.

A member of this architectural class of processors is, e.g., **Motorola's DSP 56603**. The DSP56603 is a low-cost low-power DSP, which is specially optimized for mobile wireless applications and provides a processing performance of 80 MIPS. Table 1 presents the performance of our Max-Log-MAP implementation in terms of clock cycles per data bit and MAP (cyc/(bit · MAP)), and its throughput assuming an 8-state Turbo-Code and five iterations.

From this performance number of a Max-Log-MAP decoder, it is obvious that the processing power provided by this class of DSPs is far below the needs of even one data channel in 3G wireless communications. The Log-MAP algorithm is even much more demanding due to the max* operation.

4.2. Modern VLIW DSPs

Modern DSP architectures attempt to increase the signal processing performance by exploiting the inherent parallelism of many signal processing algorithms. This class of DSP architectures provides several independent ALU units along with wide and fast busses to the internal memories. To allow this increased degree of instruction level parallelism, instructions to be executed in parallel are grouped together to so-called very large instruction words (VLIW). Further, the processing units usually support the single-instruction/multiple-data approach (SIMD). An example is sub-word parallelism, where several sub-words of a data word are processed with the same operation. A (Max-)Log-MAP implementation for decoding an 8-state Turbo-Code on this class of DSPs should exploit the benefits of the sub-word parallelism by using 16-bit packed data types.

A state-of-the-art architecture is the **ST120** from ST-Microelectronics. It features two ALU units and supports three different instruction sets: A 16-bit instruction set for compact microcontroller code, a 32-bit instruction set for higher performance and more complex instructions, and a third one for an increased level of instruction parallelism. In this 4x32-bit Score-boarded Long Instruction Word mode the processor is able to execute four 32-bit instructions in one clock cycle. Following the SIMD approach, the processor supports 2x16-bit data packed into one 32-bit data word.

Our hand-coded assembly language implementation of an 8-state Turbo decoder requires 37 cycles per bit and MAP, using packed data types and a Max-Log-MAP algorithm. Assuming five iterations the Turbo decoder achieves a throughput of 540 kbit/s at 200 MHz. Figure 2 shows the optimized scheduled data-flow graph and register assignment for combined backward recursion and soft-output calculation. It exploits best the architectural parallelism of the ST120. Special emphasis has been put on register and sub-word allocation, because unnecessary data shuffling degrades the performance. This lead to an assignment where

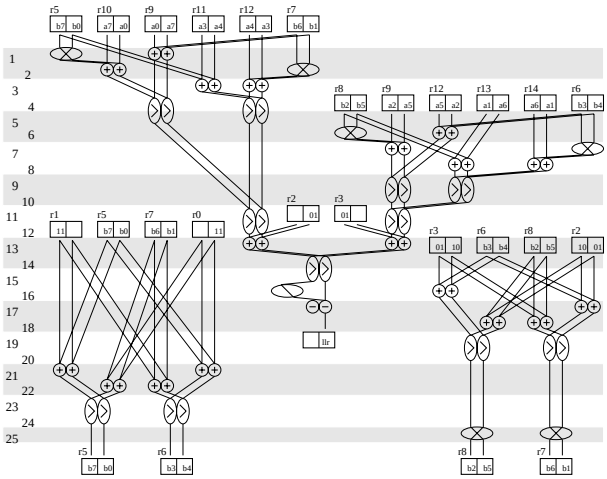

Figure 2. Combined backward recursion and soft-output computation on ST120

all registers and sub-words thereof are updated in-place and can thus directly be used for the next loop iteration. One drawback of the processor's instruction set leads to a severe degradation of the decoding performance: The maximum selection for packed data types needs two clock cycles to complete.

In a Log-MAP implementation, the maximum selection is replaced by the max*-operator (cf. eqn. (1)). The max*-operation comprises these steps: Difference of parameters, absolute value, access to lookup-table (LUT), and adding the LUT result to maximum of parameters. The sequence of these steps increases the cycle count by a factor of ten compared to the plain maximum operation. Thus the overall cycle count per bit is tripled compared to Max-Log-MAP.

Other implementations on VLIW DSPs, e.g. [5, 8], suffer also from the fact that using the Log-MAP algorithm degrades the throughput by up to 70 %. Although the Log-MAP algorithm can provide an improved bit-error performance of up to about 0.5 dB over the Max-Log-MAP, it is apparent that the Max-Log-MAP is the best compromise, as long as a dedicated instruction for the max* operation is lacking. User-configurable architectures are one means to close this gap.

5. Application-customized RISC cores

Significant attention has recently been drawn towards configurable processor cores such as those offered by ARC [1] and Tensilica [2]. These are based on classical RISC architectures that can be configured in two dimensions: First with respect to the architectural features of the core, e.g. inclusion of fast MAC units, cache sizes and policies, mem-

ory size and bus-width. Secondly the instruction set can be extended by user-defined instructions. This approach offers the benefit of removing application specific performance bottlenecks while still maintaining the flexibility of a software implementation, thus blurring the borders between hardware and software. For instance, performance-critical code portions that require multiple instructions on a generic RISC architecture can be compressed into a single, user-defined instruction to obtain a significant speed-up. More importantly, this can on system-level eliminate the need for a heterogeneous RISC/DSP or RISC/IP-block architecture, therefore simplifying the architecture, reducing the cost of the system and simplifying the validation of the total system.

Key to efficiently using a configurable processor core is the methodology behind defining and implementing the custom instructions.

ARC's approach uses two independent descriptions that have to be provided to the tools: A C-model that describes the behavior of the new instruction to the instruction set simulator. And second, a synthesizable VHDL model that extends the VHDL description of the processor core. However, it is difficult to validate the consistency of the two models, therefore creating a potential validation hole.

The approach followed by Tensilica tries to tackle this validation problem by a single source model that is used for both extending the software tool chain with the custom instructions as well as for generating the synthesizable HDL of the extended processor. The implementation of the new instructions is done using a special extension language, called *TIE*, which is essentially a hybrid of Verilog and C.

The performance critical parts of a Turbo decoder are the path metric update, performed by the `recursion_step` function, and `softout_calculation`, cf. Algorithm 1. As discussed before, a tailored add-compare-select (ACS) instruction, calculating one or even more butterflies in one clock cycle, supporting the max*-operation and addressing the according data-transfer issues could improve the performance significantly. Therefore we introduce a new operation, called "LM_ACS", highlight in the following its arithmetic and data-transfer issues and present for the first time throughput and silicon performance results.

5.1. Arithmetic issues

The arithmetic functionality of the LM_ACS-operation is based on equations (2) and (3). The use of the Lookup-table for the correction term within the max*-operations is seamlessly integrated within this operation, thus realizing a full Log-MAP implementation at little additional hardware cost. The LM_ACS operation requires two path metrics (PM1, PM2) and two branch metrics (BM1, BM2) to process a butterfly. As result, two updated path metrics (RM1, RM2) are returned:

$$(RM1, RM2) = LM\_ACS(PM1, PM2, BM1, BM2) \quad (4)$$

Each complete path metric update for either forward or backward recursion at time step k can be mapped to a multiple execution of the above LM_ACS operation. The parameters $\gamma_k(I)$ and $\gamma_k(II)$ of Figure 1 correspond to BM1 and BM2 in the functional interface.

For example, to update the path metrics for the forward recursion of an 8-state Turbo decoder relevant for UMTS, the path metric update can be split into four butterflies. The used code defines the state numbers of the incorporated states for each processed butterfly. Four different branch metrics exist ($\gamma_k^{0,0}$, $\gamma_k^{1,1}$, $\gamma_k^{0,1}$, $\gamma_k^{1,0}$) that are pairwise mapped onto the parameters BM1 and BM2. With these parameters the butterflies are processed using the LM_ACS instruction:

$$(\alpha_k(0), \alpha_k(1)) = LM\_ACS(\alpha_{k-1}(0), \alpha_{k-1}(4), \gamma_k^{0,0}, \gamma_k^{1,1}) \quad (5)$$

$$(\alpha_k(2), \alpha_k(3)) = LM\_ACS(\alpha_{k-1}(1), \alpha_{k-1}(5), \gamma_k^{0,1}, \gamma_k^{1,0}) \quad (6)$$

$$(\alpha_k(4), \alpha_k(5)) = LM\_ACS(\alpha_{k-1}(2), \alpha_{k-1}(6), \gamma_k^{1,0}, \gamma_k^{0,1}) \quad (7)$$

$$(\alpha_k(6), \alpha_k(7)) = LM\_ACS(\alpha_{k-1}(3), \alpha_{k-1}(7), \gamma_k^{1,1}, \gamma_k^{0,0}) \quad (8)$$

5.2. Data-transfer issues

An appropriate addressing scheme must ensure the correct mapping of the source operands and results. The way these parameters are transferred, highly depends on the actual processor environment, e.g. with regard to bit-widths, addressing schemes, internal memories, size of register file, etc. These parameters can be given either directly or implicitly.

Processing the update of the forward path metrics for a data block, equations (5) to (8) are executed in a loop. During each loop iteration, which is supposed to take only a minimum number of clock cycles, a set of branch metrics has to be read from memory and the updated path metrics $\alpha_k(0\ldots7)$ have to be stored. Usually, memory accesses take more than one clock cycle to complete. Using means like pre-fetching of values and implicit address generation even multi-cycle memory accesses can be parallelized with the data processing. In case of writing, the latency is hidden in the processor pipeline. Considering these techniques we defined and implemented a set of instructions, each of which performs a different memory operation in addition to the butterfly processing. Thus the data transfers could be totally hidden in the forward recursion loop that takes only four clock cycles, assuming zero-overhead looping.

5.3. Throughput and silicon performance

Further dedicated instructions, not explained here, are used to support the soft-output calculation, termed LLR

Processor	clock freq.	cyc/ (bit · MAP)	throughput @ 5 iter.
Max-Log-MAP			
56603	80 MHz	165	48.6 kbit/s
ST120	200 MHz	37	540 kbit/s
Log-MAP			
ST120	200 MHz	≈ 100	≈ 200 kbit/s
cust. RISC	100 MHz	33	303 kbit/s

Table 1. (Max-)Log-MAP performance

(log-likelihood ratio). We performed two independent implementation studies using the previously introduced set of application specific instructions on both ARC's and Tensilica's RISC core. Both cores showed similar silicon and throughput performance. The synthesis results are based on a $0.18\,\mu$m target library under worst-case conditions (1.55 V, 125° C). A configured RISC core without instruction set extensions requires 48 k gates and runs at 180 MHz, whereas this core with the application specific instruction set extensions requires 104 k gates. Since the hardware extensions are on the critical path of the core, the maximum clock frequency decreases to 100 MHz.

Along with the branch metric calculation and some code overhead the total performance of the implementations is 33 cycles per bit and MAP, cf. Table 1. There is still optimization potential of up to 25 %, because the calculation of the branch metrics can also be efficiently supported by dedicated instructions. We expect that these instructions will not further extend the critical path of the customized core and thus not decrease the maximum clock frequency.

6. Conclusions

Turbo-Codes as channel coding scheme are part of the 3G cellular wireless standards. The complexity of the decoding algorithm and the throughput requirements pose great demands on the computational power of the signal processing devices. Current DSPs support the kernel operations of the Viterbi algorithm, however for the MAP algorithm, used for decoding Turbo-Codes, this support is lacking. To obtain efficient implementations of an algorithm we have identified four major methodical steps which are strongly interrelated. In this paper, we have explained those steps in broader detail that are dependent on the target architecture.

The signal processing power provided by 16-bit fixed point DSPs is far below the needs of 3G wireless communications. Using VLIW DSPs one 3G data channel can be processed, however, as long as dedicated instruction support for Log-MAP algorithm is missing, only the sub-optimal Max-Log-MAP algorithm can be efficiently implemented.

Application-customized RISC cores are a new promising alternative which melts the borders between hardware and software. We have introduced tailored instructions for supporting the Log-MAP algorithm on those RISC cores, addressing arithmetic and data-transfer issues. The results of these implementations show that the Log-MAP performance is about 50 % higher than on an ST120, a current VLIW architecture. This persuasive result of the customized RISC cores comes along with lower power consumption and device costs compared to high-performance DSPs.

References

[1] ARC International Ltd. `http://www.arccores.com`.

[2] Tensilica Inc. `http://www.tensilica.com`.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes. In *Proc. ICC '93*, pages 1064–1070, Geneva, Switzerland, May 1993.

[4] M. Bickerstaff et al. DSP Systems for Next-Generation Mobile Wireless Infrastructure. In *Proc. ICASSP 2000*, pages 3710–3713, 2000.

[5] A. Chass and A. Gubeskys. On Performance/Complexity Analysis and SW Implementation of Turbo-Decoding. In *Proc. 2nd International Symposium on Turbo-Codes & Related Topics*, pages 531–534, Brest, France, September 4–7, 2000.

[6] S. Halter, M. Öberg, P. M. Chau, and P. H. Siegel. Reconfigurable Signal Processor for Channel Coding & Decoding in Low SNR Wireless Communications. In *Proc. 1998 Workshop on Signal Processing Systems (SiPS '98)*, pages 260–274, Cambridge, Massachusetts, USA, Oct. 1998.

[7] H. Michel and N. Wehn. Turbo-Decoder Quantization for UMTS. *IEEE Communications Letters*, 5(2):55–57, Feb. 2001.

[8] J. Nikolic-Popovic. Implementing a MAP Decoder for cdma2000 Turbo Codes on a TMS320C62x DSP Device. Technical report, Texas Instruments Incorporated, Houston, Texas, USA, May 2000. Application Report SPRA629.

[9] P. Robertson, P. Hoeher, and E. Villebrun. Optimal and Sub-Optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding. *ETT*, 8(2):119–125, Mar.–Apr. 1997.

[10] I. Verbauwhede and C. Nicol. Low-Power DSP's for Wireless Communications. In *Proc. ISLPED 2000*, pages 303–310, Rapallo, Italy, 2000.

[11] J. Vogt, K. Koora, A. Finger, and G. Fettweis. Comparison of Different Turbo Decoder Realizations for IMT-2000. In *Proc. 1999 Global Telecommunications Conference (Globecom '99)*, volume 5, pages 2704–2708, Rio de Janeiro, Brazil, Dec. 1999.

[12] Z. Wang, H. Suzuki, and K. K. Parhi. VLSI Implementation Issues of Turbo Decoder Design For Wireless Applications. In *Proc. 1999 Workshop in Signal Processing Systems (SiPS '99)*, pages 503–512, Taipei, Taiwan ROC, Oct. 1999.

[13] A. Worm et al. Advanced Implementation Issues of Turbo-Decoders. In *Proc. 2nd International Symposium on Turbo Codes & Related Topics*, pages 351–354, Sept. 2000.