Communication Centric Architectures for Turbo-Decoding on Embedded Multiprocessors *

Frank Gilbert, Michael J. Thul, Norbert Wehn
Microelectronic System Design Research Group, University of Kaiserslautern
Erwin-Schrödinger-Straße, 67663 Kaiserslautern, Germany
{gilbert, thul, wehn}@eit.uni-kl.de

Abstract

Software implementations of channel decoding algorithms are attractive for communication systems with their large variety of existing and emerging standards due to their flexibility and extensibility. For high throughput, however, a single processor can not provide the necessary compute power. Using several processors in parallel without exploiting the internal parallelism of the algorithm leads to intolerable overhead in area, power consumption, and latency.

We propose a multiprocessor based Turbo-Decoder implementation where inherently parallel decoding tasks are mapped onto individual processing nodes. The implied challenging inter-processor communication is efficiently handled by our framework such that throughput is not degraded. In this paper we present communication centric architectures from buses to heterogenous networks that allow to interconnect numerous processors to perform high throughput Turbo-decoding.

1 Introduction

Programmable platforms are becoming an emerging competitor to dedicated hardware solutions in embedded systems. Design cost reduction, time-to-market, flexibility and extensibility, which allow to adapt to changing or multiple standards, are the driving forces behind this trend. For high throughput applications, communication centric architectures composed of several processors are mandatory. The importance of these platforms in communication applications is continously increasing [1].

The outstanding forward error correction provided by Turbo-Codes, which were introduced in 1993 [2], made them part of today's communication standards, e.g. 3GPP [3]. They consist of concatenated component codes that work on the same block of information bits, separated by interleavers. The component codes are decoded individually. Key to the performance of Turbo-Codes is the iterative exchange of interleaved information between the component decoders. For an introduction to Turbo-Codes see [4].

While the maximum throughput of 3GPP is limited to 2 MBit/s, future communication standards will demand for much higher data rates (>10 MBit/s). Efficient implementations of Turbo-Decoders on programmable architectures are of great importance due to their flexibility to support the various existing or even emerging standards. Throughput on these architectures, however, is limited by the processor's arithmetic and memory architecture.

For higher throughput rates parallelization is required either by increasing the instruction-level parallelism, moving to multiprocessor architectures, or both. We first increase the instruction-level parallelism by an application specific instruction-set extension of a customized RISC core which is tuned for embedded applications [5]. It features a lower power consumption and device cost when compared to high-performance DSPs with VLIW- or SIMD-architectures.

Parallel interleaving in an iterative process obstructs parallelization. Therefore, in standard multiprocessor implementations several blocks are decoded on independent processors, which multiplies the costs (memories, area, power consumption, and latency) along with the throughput.

Exploiting the inherent algorithmic parallelism of Turbo-decoding, however, enables a far more efficient partitioning of the decoding task: as will be shown later, the block to decode can be divided into several sub-blocks. Decoding each sub-block on an individual processor significantly reduces memory overhead and latency, which is a critical parameter in many communication applications.

Due to the iterative exchange of interleaved data each processor has to communicate with each other processor, yielding only limited locality. We extract the communication demands for each processor node. Based on this information we deduce different communication networks dependent on the throughput requirements. We show that the

* This work has been supported by the *Deutsche Forschungsgemeinschaft (DFG)* under Grant We 2442/1-1

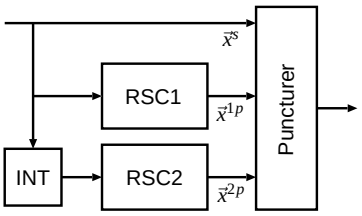

Figure 1. Turbo-Encoder

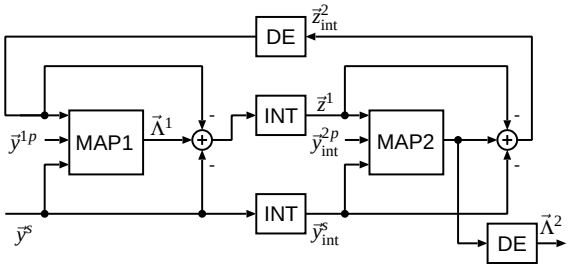

Figure 2. Turbo-Decoder

area overhead is minimal and the interconnection network does not degrade the throughput of the overall system.

The paper is structured as follows: In Section 2 we present the concept of Turbo-Codes, the component decoder algorithm, and the special requirements dictated by the interleaving before we compare different target architectures. Our new approach is outlined in Section 3 and the proposed architecture template laid out in detail in Section 4. The results in Section 5 include throughput and area figures for various degrees of parallelization and component decoder implementations using a Tensilica Xtensa Core [6]. Section 6 finally concludes the paper.

2 Turbo-Codes

Forward error correction is enabled by introducing parity bits. In Turbo-Codes, the original information ($\vec{x}^s$), denoted as *systematic information*, is transmitted together with the parity information ($\vec{x}^{1p}, \vec{x}^{2p}$). For the Third Generation Partnership Project (3GPP) [3], the encoder consists of two recursive systematic convolutional (RSC) encoders with constraint length $K = 4$, which can also be interpreted as 8-state finite state machines. One RSC encoder works on the block of information in its original, the other one in an interleaved sequence, see Figure 1. On the receiver side a corresponding component decoder for each of them exists. The maximum a posteriori (MAP) decoder has been recognized as the component decoder of choice as it is superior to the Soft-Output Viterbi Algorithm (SOVA) in terms of communications performance and implementation scalability [7].

The soft-output of each component decoder ($\vec{\Lambda}$) is modified to reflect only its own confidence ($\vec{z}$) in the received information bit of being sent either as "0" or "1". These confidences are exchanged between the decoders to bias their next estimations iteratively (see Figure 2). During this exchange, the produced information is interleaved following the same scheme as in the encoder. The exchange continues until a stop criterion is fulfilled. The last soft-output is not modified and becomes the soft-output of the Turbo-Decoder ($\vec{\Lambda}^2$). Its sign represents the $0/1$ decision and its magnitude the confidence of the Turbo-Decoder in it. Stop criteria range from "fixed number of iterations done" over cyclic redundancy checks (CRC) to statistical analysis.

2.1 The MAP Algorithm

Given the received samples of systematic and parity bits (*channel values*) for the whole block y_0^N, where N is the block length, the MAP algorithm computes the probability for each bit to have been sent as $d_k = 0$ or $d_k = 1$. The logarithmic likelihood ratio (LLR) of these probabilities is the soft-output, denoted as:

$$\Lambda_k = \log \frac{\Pr\{d_k = 1 | y_0^N\}}{\Pr\{d_k = 0 | y_0^N\}}. \tag{1}$$

Equation 1 can be expressed using three probabilities, which refer to the encoder states S_k^m, where $k \in \{0 \ldots N\}$ and $m, m' \in \{1 \ldots 8\}$:

The *branch metrics* $\gamma_{k,k+1}^{m,m'}(d_k)$ is the probability that a transition between S_k^m and $S_{k+1}^{m'}$ has taken place. It is derived from the received signals, the a-priori information given by the previous decoder, the code structure and the assumption of $d_k = 0$ or $d_k = 1$, for details see [8]. From these branch metrics the probability α_k^m that the encoder reached state S_k^m given the initial state and the received sequence y_0^k, is computed through a forward recursion:

$$\alpha_k^{m'} = \sum_m \alpha_{k-1}^m \cdot \gamma_{k-1,k}^{m,m'}.$$

Performing a backward recursion yields the probability $\beta_{k+1}^{m'}$ that the encoder has reached the (known) final state given the state $S_{k+1}^{m'}$ and the remainder of the received sequence y_{k+1}^N:

$$\beta_k^m = \sum_{m'} \beta_{k+1}^{m'} \cdot \gamma_{k,k+1}^{m,m'}.$$

αs and βs are both called *state metrics*. Equation 1 can be rewritten as:

$$\Lambda_k = \log \frac{\sum_m \sum'_m \alpha_k^m \cdot \beta_{k+1}^{m'} \cdot \gamma_{k,k+1}^{m,m'}(d_k = 1)}{\sum_m \sum'_m \alpha_k^m \cdot \beta_{k+1}^{m'} \cdot \gamma_{k,k+1}^{m,m'}(d_k = 0)}. \tag{2}$$

The original probability based formulation as presented here involves a lot of multiplications and has thus been ported to the logarithmic domain to become the *Log-MAP Algorithm* [8]: Multiplications turn into additions and additions into maximum selections with additional correction terms[1]. Arithmetic complexity can further be reduced

[1] Also know as the Jacobian logarithm, see [8] for details.

by omitting the correction term (*Max-Log-MAP Algorithm*) which leads to a slight loss in communications performance (about 0.1 dB). Log-MAP and Max-Log-MAP algorithm are common practice in state-of-the-art implementations.

The branch metrics are computed along with the first recursion and the soft-output (LLR) in parallel to the second recursion, therefore, only one set of state metrics, either αs or βs has to be stored.

The data dependency throughout the whole block can be "loosened" by starting the recursions on arbitrary positions in the block with approximated initialization values. For this, a recursion on a certain number of proceeding bits (*acquisition*) must be performed to obtain sufficiently accurate estimates. Windowing [9] exploits this property to divide the data into sub-blocks. Several sub-blocks can thus be decoded sequentially on the same hardware for memory reduction as only the state metrics of one sub-block have to be stored. Moreover, windowing allows to map sub-blocks to individual nodes for parallel processing, allowing to trade off hardware for latency. This is exploited in our approach.

2.2 Interleaving

Interleaving is scrambling the processing order to break up neighborhood-relations. It is essential for the performance of Turbo-Codes. Interleaver and deinterleaver tables contain one-to-one mappings of source addresses to destination addresses. One LLR has to be read for every LLR produced. Interleaving can be performed on the fly through indirect addressing for up to one LLR per clock cycle only. When more LLRs are produced, multiple LLRs have to be read and written concurrently. This is, *several* of them *have to be fetched* from *and stored* to memories *in the same clock cycle*. Without advanced communication schemes the resulting conflicts form the major bottleneck in parallel Turbo-decoding.

In [10] we presented an optimized concurrent interleaving architecture to solve the resulting conflicts. It is a distributed architecture with local entities that resolve local conflicts only, which is called ring-interleaver-bottleneck-breaker (RIBB) architecture. Nodes, each associated with a producer of one LLR/cycle, are connected in a ring structure with buffers between them. These nodes determine locally where the incoming LLRs have to be routed to: store them in the local RAM, or forward them to the next node. See [10] for details.

Note that good interleavers, in terms of communication performance, map neighboring addresses evenly to far spread target addresses. The probability that produced data is assigned to a given node can thus be assumed to have an equal distribution $1/N$, where N is the number of nodes.

2.3 Target Architecture

The computational complexity of the decoding algorithm 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. Recently some very advanced DSPs like the TigerSharc from Analog Devices [11] have implemented a special max^* instruction to support an efficient Log-Map implementation. This is a similar approach as for the Viterbi algorithm which is supported by many DSPs with the special ACS instruction.

In [5] we presented an Turbo-Decoder implementation on a customized RISC core with application specific instruction-set extensions for Turbo-decoding and compared the results to implementations on state-of-the-art VLIW DSPs. This analysis shows competitive throughput results of the customized RISC core but with much lower power consumption and device cost compared to high-performance DSPs.

Dedicated hardware solutions are chosen for applications, where throughput, area, and power requirements are more important than flexibility. In [12] we presented a scalable high-throughput architecture in an ASIC design methodology. A high area/power efficiency is reached by optimized datapaths and a tailored memory- and communication-architecture. Though it is a fully parameterizable design, code rate and generator polynomial must be fixed at design time.

3 New Approach

In this work we present for the first time a Turbo-Decoder implementation on a multiprocessor platform based on sub-block parallelization. This is, instead of distributing independent blocks, one block is segmented into sub-blocks (or windows) which are decoded on individual processors. Thus latency and memory demand for each processor are decreased significantly, on the other hand, communication between processors becomes mandatory. We show how the communication problem of distributing the interleaved (or de-interleaved) LLRs can be efficiently solved by an optimized message passing communication network. A scalable architecture template and a construction scheme for different throughput requirements based on the decoding performance of a single processor node are proposed. The communication network is derived exploiting the statistical properties of the interleaving scheme that each node is addressed with the same probability.

4 Architecture Template

As motivated above, efficient distribution of the interleaved (or deinterleaved) LLRs is the key to high-throughput Turbo-decoding on any parallel architecture.

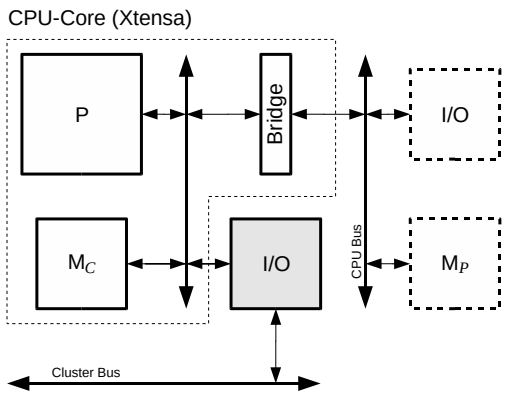

Figure 3. Architecture of Processing Node

For efficient processing all input data of an iteration has to be stored in a fast single cycle access memory (cache or SRAM). Therefore the LLRs must also be delivered in a single cycle access to the processing node. This results in an I/O-device tightly coupled to the pipeline of the embedded processor.

In Figure 3 we show a processing node based on Xtensa processor architecture. We assume that the input data are stored in the fast memory M_C instead of the main memory M_P due to the high memory-bandwidth required by the algorithm. The I/O-device for LLR-distribution is connected to the processors internal address- and databus for fast access. For the Xtensa processor, the I/O-device is connected to the XLMI-interface.

The memory-mapped I/O-device can be accessed by the processor using the example address map of Figure 4 which implements a message-passing model for multiprocessor communication. For LLR-distribution the processor writes a data word (target processor-id, local address in input buffer, and LLR-value) to the LLR Distribution FIFO. From there the LLR is transmitted through the communication network and stored at the appropriate address in the input buffer of the target processor's I/O-device. During decoding one LLR-input buffer is read, while the other is filled with the received LLRs.

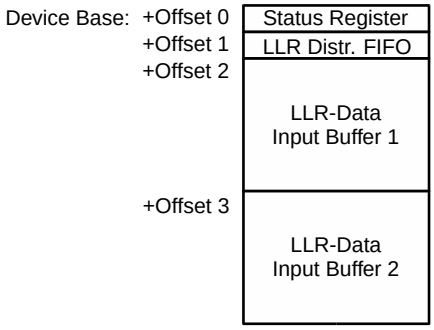

Figure 4. Address-Map of I/O-Device

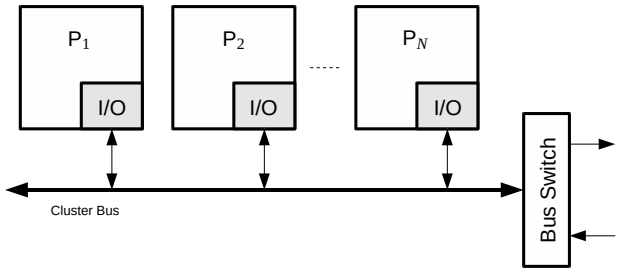

Figure 5. Cluster of Processing Nodes

We characterize a processing node with its average number of clock cycles R for one LLR during soft-output calculation. During decoding of a block with a length of K bits on a multiprocessor platform with N processors the communication network has to be able to process N/R LLRs per clock cycle. For small scale multiprocessor systems with $N \leq R$ a simple bus structure similar to Figure 5 is sufficient.

For $N = R$ the maximum capacity of one LLR per cycle on the bus is reached, adding further processing nodes can not increase decoding throughput. Therefore a new communication scheme is necessary. We combine maximal loaded busses with the RIBB-architecture to form a hierarchical communication network. The modified RIBB-cell with a bus switch to connect the cluster bus is shown in Figure 6.

For large number of processing nodes ($N > R$) we propose the architecture of Figure 7. A number N_C of processing nodes is connected to a bus. The number N_C is chosen to fully exploit the bus' capacity. Each cluster is connected over the bus switch to a ring-interleaving network. The total number of clusters is denoted as C in the following. Note that $N_C \cdot C = N$.

Given an equal distribution of the traffic (each node produces an equal number of LLRs for each other node including itself), which is a fair assumption for good interleavers, we can statically analyze the network. The traffic on each cluster's bus can be divided as follows:

- $\frac{N_C}{C} \cdot \frac{K}{N} = \frac{1}{C^2} \cdot K$ LLRs internal cluster traffic, which are the LLRs produced and consumed in the cluster.
- $N_C \cdot \frac{(C-1)}{C} \cdot \frac{K}{N} = \frac{C-1}{C^2} \cdot K$ LLRs leaving the cluster, which are the LLRs produced within the current cluster but consumed by nodes of different clusters.
- $N_C \cdot \frac{(C-1)}{C} \cdot \frac{K}{N} = \frac{C-1}{C^2} \cdot K$ LLRs entering the cluster, which are the LLRs consumed by the nodes of the current cluster but produced by nodes of different clusters.

For a communication network which does not degrade the performance all of the clusters' bus cycles ($\frac{1}{C^2} \cdot K + 2 \cdot \frac{C-1}{C^2} \cdot K$) must be completed during the processors soft-output calculation period of $R \cdot \frac{K}{N}$ cycles.

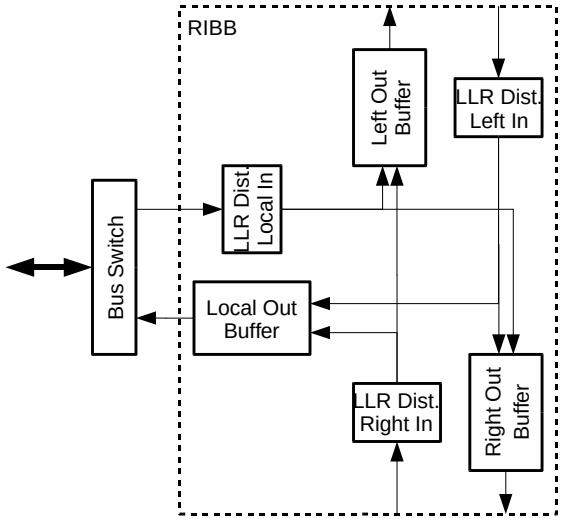

Figure 6. RIBB Cell with Bus Switch

Therefore, for a given processor-node with LLR-throughput R and a targeted parallelism of N nodes the number of clusters C and the number of nodes per cluster N_C must be carefully chosen to satisfy the relation

$$\frac{2C-1}{C} \cdot N_C \leq R, \text{ with } N_C \cdot C = N. \quad (3)$$

The bus arbitration scheme must reflect the ratio of the number of LLRs produced by the current cluster to the total number of LLRs transferred over the bus. In our architecture the arbitration scheme should on average grant the bus in $\frac{C}{2C-1}$ cycles to one of the local nodes. For larger C a round-robin-arbitration is sufficient which grants the bus to one of the local nodes and the bus switch alternately.

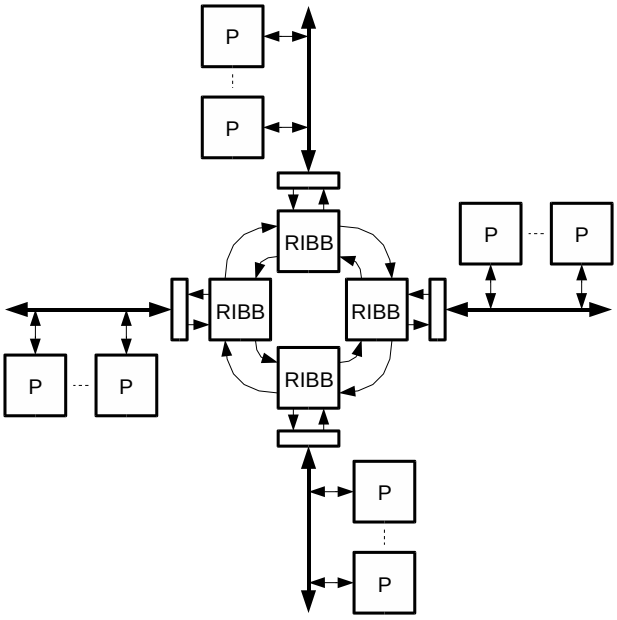

Figure 7. Ring-connected Clusters

Simulation and static analysis of our ring-interleaving network (which is beyond the scope of this paper) show that it has a limited scalability. The total number of nodes should not exceed a certain maximum ($N \leq 8 \cdot R$). If the number of nodes is further increased, the latency induced by this communication architecture becomes dominant, limiting the throughput increase. Communication architectures for higher degrees of parallelism and their static analysis are ongoing work.

Although our proposed architecture is optimized for the distribution of LLRs, it can also be used for the state-metric exchange after acquisition and first recursion. Furthermore it can be easily extended to the transmission of the input and decoded data.

5 Results

In Section 4 we have shown that the multiprocessor architecture strongly depends on the implementation performance on a single processor R. Therefore we compare the throughput and area of different multiprocessor configurations based on two different decoder implementations on the Xtensa architecture. All area and throughput results are based on synthesis with Synopsys Design Compiler on a 0.18μm ASIC-technology under worst-case conditions.

Configuration A is described in detail in [5]. It implements a Turbo-Decoder on a Xtensa processor with customized instruction-set extensions running at 100MHz (worst-case conditions). It takes about 33 cycles per LLR in each MAP decoder leading to a total throughput of 0.303MBit/s with 5 Turbo-iterations. During soft-output calculation one LLR is calculated every 15 cycles ($R = 15$). The CPU-core with the application specific extensions requires 1.28mm^2.

Configuration B is an optimized version of configuration A (see [13]). This configuration runs at 133MHz (worst-case conditions), it takes about 9 cycles per LLR in each MAP decoder leading to a total throughput of 1.48MBit/s with 5 Turbo-iterations. During soft-output calculation one LLR is calculated every 5 cycles ($R = 5$). The CPU-core with the application specific extensions is expected to require 1.54mm^2.

The results of the throughput calculation include the decoding time for all nodes plus the multiprocessor overhead (i.e time for calculation of the acquisition, exchange of state-metric values after acquisition and first recursion, and distribution of the LLRs). A blocklength of 5114 Bit is assumed, which is the maximum blocklength of the 3GPP standard. The throughput does not include the time for transmission of input or decoded data.

The total area includes: the area of all processor cores, their associated data-memories for input- and intermediate

Total Nodes (N)	# of Clusters (C)	Cluster Nodes (N_C)	Approx. Throughp. [MBit/s]	Area Comm. [mm^2]	Area Total [mm^2]
1	1	1	0.303		6.16
15	1	15	4.44	0.64	29.10
20	2	10	5.90	1.26	37.41
32	4	8	9.33	2.27	56.99
64	8	8	17.99	4.91	109.14
112	16	7	29.99	10.67	189.57
119	17	7	31.63	11.53	201.36

Table 1. Results for Different Degrees of Parallelization, Configuration A

Total Nodes (N)	# of Clusters (C)	Cluster Nodes (N_C)	Approx. Throughp. [MBit/s]	Area Comm. [mm^2]	Area Total [mm^2]
1	1	1	1.48		6.42
5	1	5	7.28	0.21	14.45
6	2	3	8.72	0.66	16.73
9	3	3	12.96	1.03	22.52
8	4	2	11.58	1.25	20.91
16	8	2	22.64	2.88	36.98
32	16	2	43.25	7.29	70.26
40	20	2	52.83	10.05	87.47

Table 2. Results for Different Degrees of Parallelization, Configuration B

values, the I/O-device logic, the memories for the LLR-Distribution-FIFO, the LLR input-buffers, and the area for all RIBB-cells. The instruction memories of the processor cores for Turbo-Decoder application, interrupt service routines, or boot-code etc. are not included.

For comparison, parallelization on *block level* with 16 *independent* processing nodes (Configuration B) would lead to a throughput of 23.68 MBit/s with an area of 102.7 mm^2. Our approach (see Table 2), in contrast, can reach up to 95 % of that throughput with only 36 % of the area. This shows the superior efficiency of sub-block parallelization.

6 Conclusion

In this paper we present, to the best of our knowledge for the first time, an efficient mapping of a Turbo-decoding algorithm onto a multiprocessor platform by optimally exploiting its inherent parallelism:

The instruction-level parallelism is increased by choosing the Tensilica Xtensa core with an appropriate application specific instruction set extension. The sub-block parallelism is exploited by distributing the task of decoding one block on multiple communicating processors. The major bottleneck in parallel Turbo-decoding, the exchange of interleaved data, is solved through an advanced heterogenous communication network, thus minimizing the parallelization overhead.

The architecture is, however, not limited to Turbo-decoding, it solves the communication bottleneck for any multiprocessor platform where each node communicates with each other node with equal probability.

Future work will focus on integration aspects for large numbers of processors when communication architectures for even higher throughputs are incorporated.

References

[1] J. Rabaey, R. Camposano, D. Samani, L. Lerner, and R. Hetherington. Panel: "What's the Next EDA Driver?". In *Proc. 2002 Design Automation Conference (DAC '02)*, page 652, New Orleans, Louisiana, USA, June 2002.

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

[3] Third Generation Partnership Project. 3GPP home page. `www.3gpp.org`.

[4] S. A. Barbulescu and S. S. Pietrobon. Turbo Codes: A Tutorial on a New Class of Powerful Error Correcting Coding Schemes. *Journal of Electrical and Electronics Engineering, Australia*, 19(3):129–152, Sept. 1999.

[5] H. Michel, A. Worm, M. Muench, and N. Wehn. Hardware/Software Trade-offs for Advanced 3G Channel Coding. In *Proc. 2002 Design, Automation and Test in Europe (DATE 2002)*, Paris, France, Mar. 2002.

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

[7] 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.

[8] P. Robertson, P. Hoeher, and E. Villebrun. Optimal and Sub-Optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding. *European Transactions on Telecommunications (ETT)*, 8(2):119–125, March–April 1997.

[9] H. Dawid, G. Gehnen, and H. Meyr. MAP Channel Decoding: Algorithm and VLSI Architecture. In *VLSI Signal Processing VI*, pages 141–149. IEEE, 1993.

[10] M. J. Thul, F. Gilbert, and N. Wehn. Optimized Concurrent Interleaving for High-Speed Turbo-Decoding. In *Proc. 9th IEEE International Conference on Electronics, Circuits and Systems - ICECS 2002*, Dubrovnik, Croatia, Sept. 2002.

[11] Analog Devices, Inc. `http://www.analog.com`.

[12] M. J. Thul, F. Gilbert, T. Vogt, G. Kreiselmaier, and N. Wehn. A Scalable System Architecture for High-Throughput Turbo-Decoders. In *Proc. 2002 Workshop on Signal Processing Systems (SiPS 2002)*, San Diego, California, USA, Oct. 2002.

[13] H. Michel. *Implementation of Turbo-Decoders on Programmable Architectures*. PhD thesis, Institute of Microelectronic Systems, Department of Electrical Engineering and Information Technology, University of Kaiserslautern, 2002. ISBN 3-925178-87-2.