A Reconfigurable Outer Modem Platform for Future Communications Systems

Timo Vogt[1], Christian Neeb[1] and Norbert Wehn[1]

University of Kaiserslautern,
Microelectronic Design Research Group,
Erwin-Schroedinger-Strasse, 67663, Kaiserslautern, Germany
{vogt,neeb,wehn}@eit.uni-kl.de

Abstract. Current and future communications systems have to provide a large degree of flexibility e.g. to provide multi-service ability, seamless roaming, soft-infrastructure upgrading, user-defined propriety, simultaneous multi-standard operation, and different quality of service.
This paper presents a multi-processor platform for the application domain of channel decoding. Inherently parallel decoding tasks can be mapped onto individual processing nodes. The implied challenging inter-processor communication is efficiently handled by a Network-on-Chip (NoC) such that the throughput of each node is not degraded. Each processing node features Viterbi and Log-MAP decoding for support of convolutional and turbo codes of various currently specified mobile and wireless standards. Furthermore, its flexibility allows for adaptation to future systems.

Keywords. Domain-specific Reconfigurable Platform, Channel Coding, Outer-Modem

1 Introduction

Next generation mobile communication networks, beyond 3G (B3G), feature new services, especially multimedia applications, high data rates, and multi-access interoperability. The International Telecommunication Union expects that new radio access technologies will be integrated with already existing wireless and mobile networks like UMTS, WLAN, and DVB into a heterogeneous network. Seamless services with soft handover must be guarantied. Modem architectures must adapt to these diverse requirements and support different technologies and standards at the same time. *Thus flexibility becomes a dominant aspect for future modems.*

The focus of this paper is put on *channel decoding in mobile and wireless communications systems*. Here convolutional codes (CC) and concatenated convolutional codes, also known as turbo codes (TC), are established techniques. Table 1 shows coding schemes used in various existing standards. Turbo codes have an outstanding forward error correction capability. They consist of concatenated component codes that work on the same block of information bits, separated by an interleaver. The component codes are recursive convolutional codes which are decoded individually. Key to the performance of turbo codes is the iterative exchange of interleaved information

Dagstuhl Seminar Proceedings 06141
Dynamically Reconfigurable Architectures
http://drops.dagstuhl.de/opus/volltexte/2006/730

between the component decoders. For an introduction to turbo codes see [1]. Convolutional codes are decoded by the Viterbi algorithm (VA) [2] or the Maximum A posteriori Probability (MAP) algorithm [3]. The VA generates hard decision output, whereas the MAP is used if soft decision output is required, as in turbo codes.

The flexibility challenge can only be met by programmable or reconfigurable architectures. ASIC implementations are not suitable for adaptation of changes as required in B3G systems. FPGAs feature bit level flexibility, but the programming model is complex. A simple programming model and instruction level flexibility is provided by processors. Hence, efficient implementations of channel decoders on programmable architectures are of great importance to efficiently support the various existing or even emerging standards. Throughput on single processor architectures, however, is limited. Thus for high throughput applications the parallelism has to be increased on various levels, e.g. on instruction and multi-processor level.

The XiRisc [4] provides a RISC architecture enhanced by a reconfigurable, FPGA alike array that is tightly coupled with the RISC pipeline. However, the restrictions to the RISC pipeline structure, the load-store architecture, and the narrow bandwidth between the register file and the reconfigurable array limit the performance.

Efficient utilization of application specific parallelism and flexibility is key to powerful, efficient, and flexible architectures. High performance combined with the advantages of processors, namely instruction level flexibility and simple programming models, can be achieved by application specific instruction set processors (ASIP). In [5] an ASIP based on the Tensilica XTENSA platform targeting the channel coding domain was presented. It is based on a fixed RISC pipeline extended by application specific instructions. This platform is limited to a load-store RISC architecture with four pipeline stages.

Total freedom in pipeline and memory architecture design gives room for further improvement. Moreover, it allows to add application specific *run time reconfigurability* to the ASIP approach: the flexibility requirements of the application domain can be balanced between instruction level flexibility and reconfigurability, as explained in Section 3. An ASIP using this approach was proposed in [6], but it only targets the field of turbo codes with 8 states or less. Convolutional codes with high constraint lengths, which have similar computational complexity as turbo codes, and 16-state turbo codes are not supported.

In multi-processor implementations several independent data blocks can be decoded on different processors independently, which multiplies the costs (memories, area, and power consumption) along with the throughput. Latency can not be improved with this approach. Exploiting the inherent parallelism of the decoding algorithms 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 latency as a critical parameter in many communication applications, and memory overhead.

Due to the iterative exchange of interleaved data each processor working on the same data block has to communicate with each other, yielding only limited locality. A communication network has to support the communication demands of the different applications without degrading the throughput of the overall system.

Standard	Codes	Rates	States	Blocksize	Throughput
GSM	CC	1/2..1/4	16, 64	33..876	..12kbps
EDGE	CC	1/2..1/3	64	39..870	5..62kbps
UMTS	CC	1/2..1/3	256	1..504	..32kbps
	TC	1/3	8	40..5114	..2Mbps
CDMA2k	CC	1/2..1/6	256	1..744	..38kbps
	TC	1/2..1/5	8	378..20736	..2Mbps
IEEE802.11	CC	3/4..1/2	64, 256	1..4095	6..54Mbps
IEEE802.16	CC	7/8..1/2	64	1..2040	..24Mbps
	TC	3/4..1/2	8	1..648	..24Mbps
Inmarsat	TC	1/2	16	..2608	..64kbps

Table 1. Selection of standards and channel codes

We present a multi-processor platform for channel decoding based on a dynamically reconfigurable application specific instruction set processor (dr-ASIP). The platform is scalable and provides the flexibility to allocate different decoding tasks to different processors. Thus it is possible to decode multiple convolutional and turbo codes in parallel on different hardware resources. The resource allocation can be adapted to application constraints like data rate or latency.

The dr-ASIP features Viterbi and Log-MAP processing for all possible binary convolutional codes with constraint lengths between 3 and 9, and code rates between 1 and 1/4 and supports convolutional and turbo decoding.

Section 2 summarizes the flexibility requirements for the decoder architecture from the application point of view. The ASIP architecture is presented in Section 3, followed by the extension to a multi-processor platform in Section 4. Section 5 concludes the paper.

2 Application Requirements

Thorough investigation of 2G, 3G and upcoming 3.9G/4G mobile and wireless communication scenarios led to following requirements for a convolutional and turbo decoder platform (a selection of standards is summarized in Table 1):

- combined decoder for VA and Log-MAP decoding
- support of convolutional and turbo decoding
- support of constraint lengths $K_c = 3 \ldots 9$
- up to four channel values per information bit
- arbitrary but single feedback polynomial
- arbitrary generator polynomials
- high throughput (up to 100 Mbps for turbo applications)
- fast reconfigurability of channel code structure

The platform must be scalable to different scenarios like terminal or base station implementations.

3 ASIP design

3.1 General Considerations

Before designing the application specific processor, general architectural choices had to be made. The Log-MAP algorithm is computationally more expensive than the VA and will therefore be discussed first.

Various windowing schemes ([7,8]) must be supported. Therefore the recursions are programmable. This gives flexibility for instance to adjust the acquisition length to the code structure and to the communication channel conditions. The different recursions are processed sequentially on the same hardware, and forward or backward recursion can be performed first. The soft output is computed in parallel with the second recursion. One recursion step of turbo code applications is processed in a single cycle for high throughput support. Therefore it must be possible to read channel values, to process branch and state metrics in parallel, and to store multiple state metrics or soft output values, all at the same time. This requires data parallelism within the pipeline, and a customized memory architecture with high memory bandwidth. A maximum of $N = 16$ state metrics are computed in parallel. If N is larger than 16, the state metrics of a single trellis step have to be computed sequentially.

The channel code structure is at least constant for a whole data block. The overhead (area, energy, latency) of specifying it with each instruction is too high. On the other hand it is important to be able to switch within a few clock cycles from one channel code to another, for instance to support soft handover. Therefore the channel code configuration is not specified by the instructions but is kept dynamically reconfigurable within the dr-ASIP.

In case of the VA the path metrics computation and the trace back are performed sequentially. The path metric computation utilizes the same hardware as the forward state metric recursion of the Log-MAP algorithm.

3.2 Architecture Overview

Figure 1 shows the overall architecture of the dr-ASIP. It consists of the dr-ASIP core, memories, and an interleaver and deinterleaver unit (IL/DIL) connected to a network interface. Data is exchanged through the packet based network interface (see Section 4).

Several memories (one channel value (CV) memory and two IO memories) can be accessed by the dr-ASIP core as well as by the network interface. The IO memories are suited for storage of soft or hard output values, local survivors, and extrinsic information. Typically these memories are implemented by synchronous SRAMs. Their size can be tailored to the application and is only limited by the address space assigned to the different memories.

In multi-processor turbo decoder applications, the new extrinsic information has to be distributed to different target processors. This task is performed by the IL/DIL and the network interface. The IL/DIL maps the source address of the old extrinsic information to a target address before the data is sent to the network interface. The interface itself performs the message distribution. During decoding one IO-MEM, storing the old

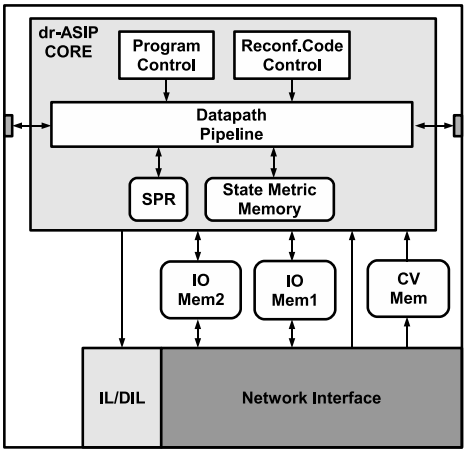

Fig. 1. Overall dr-ASIP architecture

LLRs of the actual iteration, is read by the dr-ASIP core, while the other is filled with the received new extrinsic information for the next iteration.

The control part of the dr-ASIP core consists of two parts: program control and a dynamically reconfigurable channel code control. The program control supports two nested zero overhead loops (ZOL), branches, and limited interrupt services. Pipeline control is also implemented here. The channel code control specifies the channel code structure specific parameters. It is look up table (LUT) based and consists of two sets of LUTs: a working and a shadow configuration. Within a single clock cycle the shadow configuration can be transfered to the working configuration to support run time reconfigurability. The channel code control configures for instance the number of channel values that are read in parallel from the CV memory, and the generator polynomials of the convolutional code.

The data manipulation part comprises a single data path pipeline, special purpose registers (SPR), and a state metric memory (SMM). The SMM is single ported and can store 16 state metric values in parallel. It holds the state metrics generated during the first recursion until they are consumed by the LLR computation during the second recursion. For convolutional codes with constraint lengths $K_c > 5$ the SMM is also used to store intermediate state metrics both during forward and backward recursion. The SPRs implement address generation for the memory read and write accesses of the processor's pipeline, especially the survivor memory read and write pointer generation during VA operation.

The data path pipeline consists in total of 11 stages. Its functionality is controlled by the decoded instruction as well as by the reconfigurable channel code control. A maximum of 16 state or path metrics are computed in parallel. If $K_c < 5$ parts of the pipeline are powered down. If $K_c > 5$ a load store architecture is implemented: intermediate state metrics are loaded from the SMM to a pipeline register, processed, and then stored back to the SMM. A single trellis cycle with $N = 256$ states can thus be computed in 16 consecutive steps, each step takes 4 clock cycles. The soft-output computation of the

Platform	Architecture	Clock freq.	cycles/ (bit*MAP)	Throughput/ 5 iter
Processor STM ST120	GP-DSP VLIW[5]	200 MHz	≈ 100	≈ 0.2Mbps
XTENSA	Conf. RISC[5]	133 MHz	≈ 33	≈ 0.4Mbps
FPGA	VitexII-3000[9]	80 MHz	≈ 1	≈ 8Mbps
Reconf. Proc.	XiRisc[4]	100 MHz	≈ 100	≈ 0.1Mbps
ASIP	[6]	335 MHz	≈ 7.5	≈ 4.4Mbps
ASIP	dr-ASIP	400 MHz	≈ 2	≈ 16Mbps

Table 2. Comparison of different Log-MAP implementations for UMTS turbo code applications

Log-MAP algorithm is pipelined due to critical path reduction. Due to the possibility of dynamic reconfiguration of the processor pipeline, any code structure with $3 \leq K_c \leq 9$ and rates between 1 and 1/4 are supported.

3.3 Implementation Results

The dr-ASIP core was implemented with the LISATek tool set, the generated VHDL model was synthesized with the Synopsys Design Compiler with a high speed 130 nm standard cell library at 400 MHz under worst case conditions (1.05V, 125C). The total gate count of the dr-ASIP core without memories is 64450 gate equivalents (GE). The SMM for a maximum window size of 64 adds another 8967 GE. The data path pipeline alone accounts for 84.5% of the dr-ASIP core area. Compared to the processors of [5] and [6] with 104 kGE and 93 kGE for the core's logic, respectively, the dr-ASIP saves more than 30% of the area.

The throughput for the Log-MAP algorithm at 400 MHz clock frequency varies from 0.9 Mbits/s for $K_c = 9$ to 180 Mbits/s for $K_c \leq 5$, and from 12 Mbits/s to 133 Mbits/s for the VA, respectively. Table 2 summarizes Log-MAP decoder implementations for UMTS turbo code applications on different target platforms. The clock frequencies listed are maximum values. They differ, among other things, because the target technology is not the same. However, the dr-ASIP outperforms the other processor implementations even if they all run with the same clock frequency.

4 Application specific multi-processor platform (AP-MPSoC)

As already mentioned, future communication standards will demand for very high data rates with constrained latency. Hence, only increasing the instruction level parallelism of a single processor core is not sufficient, and we have to move to massively parallel multi-processor architectures, so-called application specific multi-processor systems-on-chip (AP-MPSoC). In [10] a flexible multi-processor architecture for parallel turbo decoding was presented for the first time. Customized RISC cores are augmented with application specific hardware accelerators to increase computational power. However,

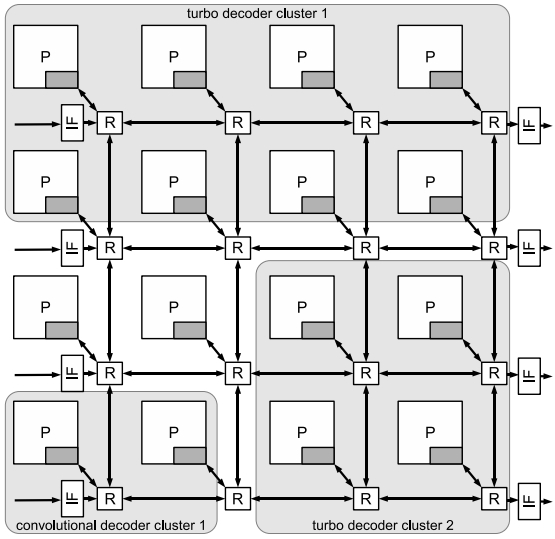

Fig. 2. Example mapping of multiple decoding tasks on application specific multiprocessor platform (AP-MPSoC)

the flexibility to support multiple coding standards is rather limited and only a subset of turbo decoding applications is supported at moderate performance.

A simple solution to increase the degree of parallelism is to decode independent data blocks on different processors independently. However, such a solution does not decrease the decoding latency. Since latency is a very critical issue such a solution is infeasible in many cases. Furthermore, due to the limited size of the processors local IO memory, the support of large block sizes can only be achieved by splitting it into several sub-blocks. Thus, we decompose the algorithm itself into a set of parallel tasks running on a set of communicating dr-ASIP cores forming processing clusters. We exploit windowing to break up a complete block of length L into n smaller sub-blocks of length B, where each sub-block is mapped onto one of the n cores in the platform.

The arising need for data communication in parallel architectures is efficiently realized by a Network-on-Chip [11] [12]. It allows to share physical bandwidth among different classes of data like IO channel data or concurrent interleaving data during decoding. The dimensioning of an application tailored network architecture is a crucial step in the MPSoC design which otherwise might lead to poor performance of the entire platform.

4.1 Architecture Template

Figure 2 shows the arrangement of the dr-ASIP cores integrated by a packet switched Network-on-Chip (NoC). It allows the dynamic mapping of independent decoding tasks onto a single dr-ASIP processor or onto multiple processors grouped in a cluster. For applications demanding for very high throughput with tight latency constraints, all proces-

sors can be assigned to a single decoding task. An example mapping of two independent turbo decoders and one convolutional decoder is depicted in Figure 2. The platform offers a high flexibility to adapt to situations of changing decoding requirements and workloads.

Each of the dr-ASIP cores is directly attached to a network *router* (R) which implements the required communication services. The routers are interconnected by bidirectional point-to-point channels forming a 2D mesh topology. For performance reasons, we provide multiple *IO interfaces* (IF) to the environment which are directly connected to the boundary routers of the network. They allow the adaption of different communication protocols like OCP or Amba AXI to the optimized internal network protocol.

Typically, IO data form long data streams because entire blocks must be completely loaded and stored by a processor before decoding. This is in contrast to the exchange of interleaving data where very small chunks of data must be transferred at high data rates. We pay attention to this by the use of two types of network packets.The first one serves for interleaving purpose and consists of a single flit only containing all header information and a small data payload. A flit (flow control digit) represents the atomic data unit of transfer in our architecture and so determines the width of a physical channel. For the remaining data, a variable size packet format is used which consists of at least three flits. The first two are header flits specifying, in addition to the target processor and a local address information, the packet *length* in terms of flits and a packet classifier for processor internal purpose. The payload can include up to four input channel or soft output values per flit, program instructions, configuration data, state metrics, or interleaver information.

We use the concept of *virtual networks* [13] to control the allocation of network bandwidth separately for the two traffic classes. This is reflected in the data path of the routers where packets located in two separate data queues form independent virtual channels. These virtual channels are then multiplexed over a common physical channel by means of a crossbar switch.

4.2 Network Traffic Estimation

To quantify the resulting network traffic, we analyzed the different processing and communication phases for the parallel turbo and convolutional decoding. The amount of data for program code and configuration of the code parameters is assumed to be small compared to the IO and interleaving data if reconfiguration occurs rarely and was therefore neglected. Furthermore, it can be shown that interleaving generates a multiple of data compared to the IO data rates.

To ensure that a 2D mesh network offers sufficient bandwidth for all configurations, we model it as a directed graph $I(R,C)$, where a vertex $r_i \in R$ represents a router and a directed weighted edge $c_{i,j}$ a (physical) communication channel between its incident routers r_i and r_j. An associated edge weight $\tau(c_{i,j}) = \tau_{i,j}$ represents the average channel traffic measured in flits per clock cycle $0 \leq \tau_{i,j} \leq 1$. Figure 3 illustrates the derivation of the network traffic. Every communication between any two nodes r_i, r_j is mapped to at least one routing path according to the employed routing algorithm. This increases the average traffic $\tau_{i,j}$ on all channels belonging to the routing path. Consequently, network traffic depends on the choice of the topology, the routing algorithm and the

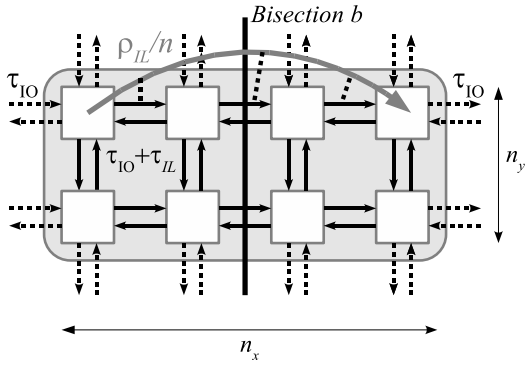

Fig. 3. Derivation of network traffic for interleaving inside a turbo decoder cluster comprising $n_x \cdot n_y$ dr-ASIP cores

communication pattern of the processors. In the case of interleaving, a distributed block permutation is performed where all nodes must communicate with all other nodes with equal probability. This reflects the behavior of good interleavers where data is scrambled randomly inside the data block. From this point of view, a random uniform model adequately models the communication pattern of the decoder cores for interleaving. Hence, all packets have equal probability $1/n$ to be sent to a specific target processor.

To estimate network traffic caused by interleaving, we refer to the network's *bisection bandwidth*. It defines the aggregate bandwidth of a minimum cut which divides the network into two equal node sets and, accordingly, the *bisection width b* as the number of channels that have to be cut. Due to the regular construction and symmetry of the 2D mesh, the bisection width b can easily be derived as $b = 2 \cdot min\{n_x, n_y\}$, with $n = n_x n_y$, and n_x and n_y number of nodes in x- and y-dimension, respectively. It is always orthogonal to the dimension containing most of the nodes. Due to the uniform communication pattern, $n\rho_{IL}/2$ interleaver packets have to cross this bisection during each cycle in average. The bisection channels carry the maximum of the interleaver traffic:

$$\tau_{IL,max}^{turbo} = \frac{n\rho_{IL}^{turbo}}{2 \cdot 2min\{n_x, n_y\}} = 0.11\, max\{n_x, n_y\} \tag{1}$$

$$\tau = \tau_{IL,max}^{turbo} + \tau_{IN} + \tau_{OUT} \leq 1 \tag{2}$$

with $\rho_{IL}^{turbo} \geq 0.45$ for dr-ASIP turbo decoder implementations.

The above equations restrict the possible shapes of the turbo processing clusters: less than nine cores may be grouped along any dimension if no IO traffic arises during interleaving crossing the cluster. This is always true if all cores are configured to process only one turbo code in parallel where the IO phase and interleaving never occur simultaneously. For a quadratic arrangement of the $N = 16$ cores in a 2D mesh no configuration exists that violates the above requirement.

5 Conclusion

Application specific flexibility is mandatory to meet the flexibility and performance requirements of B3G communications systems. It can be achieved by application specific instruction set processors with specialized pipeline topology and dedicated communication and memory infrastructure. Dynamic reconfigurability is necessary to switch during run time between different coding schemes. In this paper we presented a dynamically reconfigurable ASIP for the application domain of channel decoding (dr-ASIP). It features Viterbi and Log-MAP processing for binary convolutional codes with constraint lengths between 3 and 9, code rates between 1 and 1/4, and arbitrary feedback and generator polynomials. Convolutional and turbo decoding for more than 10 currently specified mobile and wireless standards is supported.

For high-throughput decoding, we proposed a reconfigurable application specific multi-processor platform (AP-MPSoC) as a natural transition to parallel decoding. Here, multiple dr-ASIP cores can be configured to form parallel processing clusters enabling low latency decoding. As the inter-processor communication becomes the bottleneck for high degrees of parallelization, we presented a Network-on-Chip approach to efficiently interconnect the dr-ASIP cores. A traffic analysis showed that the overall processing performance in a 4x4 2D mesh topology is not degraded by the network's limited communication bandwidth.

6 Acknowledgments

This work has been supported by the Deutsche Forschungsgemeinschaft (DFG) under grant WE 2442/4-1 within the Schwerpunktprogramm "Rekonfigurierbare Rechensysteme".

References

1. Barbulescu, S.A., Pietrobon, S.S.: Turbo Codes: A Tutorial on a New Class of Powerful Error Correcting Coding Schemes, Part 1: Code Structures and Interleaver Design. Journal of Electrical and Electronics Engineering, Australia **19** (1999) 129–142
2. Viterbi, A.J.: Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. IEEE Transactions on Information Theory **IT-13** (1967) 260–269
3. Robertson, P., Hoeher, P., Villebrun, E.: Optimal and Sub-Optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding. European Transactions on Telecommunications (ETT) **8** (1997) 119–125
4. LaRosa, A., Passerone, C., Gregoretti, F., Lavagno, L.: Implementation of a UMTS Turbo-Decoder on a dynamically reconfigurable platform. In: Proc. 2004 Design, Automation and Test in Europe (DATE '04), Paris, France (2004)
5. Michel, H., Worm, A., Münch, Wehn, N.: Hardware/Software Trade-offs for Advanced 3G Channel Coding. In: Proc. 2002 Design, Automation and Test in Europe (DATE '02), Paris, France (2002)
6. Muller, O., Baghdadi, A., Jezequel, M.: ASIP-Based Multiprocessor SoC Design for Simple and Double Binary Turbo Decoding. In: Proc. 2006 Design, Automation and Test in Europe (DATE '06), Munich, Germany (2006)

7. Dawid, H., Meyr, H.: Real-Time Algorithms and VLSI Architectures for Soft Output MAP Convolutional Decoding. In: Proc. 1995 International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC '95), Toronto, Canada (1995) 193–197
8. Zhang, Y., Parhi, K.K.: Parallel Turbo Decoding. In: Proc. 2004 International Symposium on Circuits and Systems (ISCAS '04), Vancouver, Canada (2004) II–509–II–512
9. Paulin, P., Balzano, J.M., Silburt, A., Berkel, K.V., Bramley, R., Wehn, N.: Panel: "Chips of the Future: Soft, Crunchy or Hard?". In: Proc. 2004 Design Automation and Test in Europe (DATE '04), Paris, France (2004)
10. Gilbert, F., Thul, M., Wehn, N.: Communication Centric Architectures for Turbo-Decoding on Embedded Multiprocessors. In: 2003 Design, Automation and Test in Europe (DATE '03), Munich, Germany (2003) 356–361
11. Benini, L., Micheli, G.D.: Networks on Chips: A New SoC Paradigm. IEEE Computer **35** (2002) 70–78
12. Thul, M.J., Neeb, C., Wehn, N.: Network-on-Chip-Centric Approach to Interleaving in High Throughput Channel Decoders. In: Proc. 2005 IEEE International Symposium on Circuits and Systems (ISCAS '05), Kobe, Japan (2005)
13. Duato, J., Yalamanchili, S., Ni, L. In: Interconnection Networks: An Engineering Approach. IEEE Computer Society Press, 1997, San Francisco, California, USA (1997)