# Implementation Aspects of Turbo-Decoders for Future Radio Applications

Friedbert Berens

STMicroelectronics
Advanced System Technology
Geneva Applications Laboratory
CH-1215 Geneva 15, Switzerland
*e-mail: friedbert.berens@st.com*

Alexander Worm*, Heiko Michel, Norbert Wehn

University of Kaiserslautern
Department of Electrical Engineering
Institute of Microelectronic Systems
D-67663 Kaiserslautern, Germany
*e-mail: {worm, michel, wehn}@e-technik.uni-kl.de*

**Abstract**

*Turbo-Codes will most likely be employed in future radio systems as a channel coding scheme for high-rate data services. However, Turbo-Decoding is a comparatively complex task. To obtain efficient decoder implementations, the system design space has to be explored on multiple levels. In this paper, we span the system design space for Turbo-Codes and describe a method of exploration, while focusing on the implementation-dependent part. The design decisions taken during exploration are rated regarding complexity, throughput and power consumption.*

*The second part of our paper evaluates sample software and hardware implementations of a* 2 Mbit/s *Turbo-Decoder.*

## 1 Introduction

Turbo-Codes have been a hot topic among coding theorists since their invention in 1993 [1]. They have proven to be the most powerful forward error correction scheme that is known today. Many papers have been published examining the performance of Turbo-Codes, but only a few papers address implementation complexity (e. g. [2]). The anticipated use in future mobile radio systems [3] has recently raised interest among implementation specialists as efficient implementations with emphasis on low cost and low energy consumption are needed.

The term "Turbo-Code" originally describes the parallel concatenation of two convolutional codes (PCCC), whereby one of the two encoders operates on interleaved data blocks. Other possibilities are serial concatenation (SCCC) [4] and using block codes as component codes [5]. This paper concentrates on convolutional component codes in accordance with the standardization discussion [3]. These codes are decoded with the MAP algorithm (maximum a posteriori) [6] or with the SOVA (soft-output Viterbi algorithm) [7]. As optimally decoding Turbo-Codes is almost infeasible due to the large state space, Berrou devised an iterative decoding method [1]: the component codes are decoded separately in an iterative manner while exchanging so-called extrinsic information. Encoding Turbo-Codes is a low-complexity task and is therefore omitted here.

Viterbi himself has pointed out the close relationship of Viterbi-Decoding and Turbo-Decoding with reduced-complexity MAP decoders [8]. Although much research effort has been spent on Viterbi Decoder (VD) implementations [9, 10], efficient Turbo-Decoder (TD) implementation is a widely unexplored area of research, as most of the techniques for implementing VDs are not applicable for TDs.

The paper is structured as follows: section 2 presents the system design space; section 3 subsequently deals with the exploration of the implementation-dependent part of the system design space. Various target architectures (customized hardware, software on DSP, mixed hardware/software solutions) are considered. In section 4, we rate the design decisions with respect to complexity, throughput, and power consumption. Section 5 provides some data about a software and a hardware implementation, while section 6 addresses hardware/software co-design of TDs. Section 7 concludes the paper.

## 2 Turbo-Code System Design Space

Although iterative decoding is significantly less complex than optimal decoding, it remains a computationally complex task due to the iterative use of costly component decoders. Even the use of the sub-optimal Max-Log-MAP algorithm [11] for component code decoding results in considerable high

*Alexander Worm is the recipient of a Motorola Partnerships in Research Grant.

computing performance needs. First order complexity estimations reveal approximately 1500 MOPS for a user data rate of 2 MBit/s, assuming constraint length $K = 3$ codes and five iterations.

In order to achieve complexity reduction, simplifications can be attempted at different abstraction levels (e. g. system, architecture, register-transfer, gate, transistor level). However, the optimization potential is in general closely related to the abstraction level. Application knowledge can be exploited to significantly simplify high-level specifications towards lower implementation complexity, whereas low-level design representations in most cases lack this opportunity. Thus, cost-efficient TD implementations require system design space exploration before mapping the algorithm-level specification onto hardware or DSP.

Figure 1 depicts the system design space. It comprises a service-dependent and an implementation-dependent part. The components of the Turbo-Code encoder directly define the service-dependent part of the system design space: component codes, puncturer and interleaver. This is underlined by only the encoder being defined by standardization bodies. Though the required number of iterations is implementation-dependent, this number may also depend on the service to realize different qualities of service. The number of iterations is either static or determined dynamically during decoding after evaluation of some criteria [12].

Convolutional codes are decoded with the MAP algorithm or with the SOVA. When implementing MAP or SOVA, the designer has to choose among several implementation options which reduce computational complexity, increase throughput, or reduce power consumption. Extrinsic information coupling (for the feedback) is performed according to Berrou's original method [1] or rather directly, which has first been proposed by Robertson [12].

## 3 System Design Space Exploration

The forward error correction control (FEC) has to sustain certain bit-error ratios (BER) for given signal-to-noise ratios (SNR): $\text{BER} = f(\text{SNR})$. However, implementation optimizations influence this function. Due to the lack of a comprehensive Turbo-Code theory, the degradation has to be validated by simulation and traded off against implementation complexity.

To explore the implementation-dependent part, we have developed simulation models on various abstraction levels. Models of AWGN, Rayleigh, and mobile channels are considered during validation also. A two-step design methodology is applied:

1. The first step uses a reference model – which is the link to system level – as input and obtains a bit-true model by performing algorithmic optimizations and quantization.

2. This bit-true model is used during hardware design space exploration to obtain a register-transfer structure. For software solutions, it is used to develop and optimize DSP code.

The bit-true models of the decoder may differ for hardware, software, and mixed hardware/software target architectures to reflect specific requirements. For example, while the bit-width for software solutions is imposed by the DSP architecture, it is an optimization criterion for customized hardware.

As mentioned above, this requires a sophisticated system simulation environment. The system simulator is coupled with a VHDL simulator to allow validation of VHDL models via co-simulation. In our simulation environment, simulation performance degrades on a medium-class workstation from over 3000 symbols per second for pure C-level simulation to about 20 symbols per second for co-simulation. However, this is sufficient to merely validate equivalence of bit-true and VHDL-model. Code for fixed-point DSPs is validated against the bit-true model using a cycle-accurate software simulator.

Results of the influence of service-related parameters like component codes, puncturing, and interleaving on the BER are presented in greater detail in [13] and thus omitted here.

## 4 Optimization

The complexity of a TD is a function of the component code decoder (CD) and the number of iterations (IT):

$$O_{\text{TD}} = f(\text{CD}, \text{IT}),$$

where the complexity of the component code decoder depends on operator strength (OS), amount of data reuse (DR), parallelism (P), and quantization (Q):

$$O_{\text{CD}} = f(\text{OS}, \text{DR}, \text{P}, \text{Q}).$$

In this section, we discuss the design trade-offs with respect to these parameters. Also extrinsic information coupling and intricacies of software implementations are shortly addressed.

1. Mainly two alternatives have been proposed for formulating the SOVA: trace-back [7] and register-exchange [5] structure. These induce different implementation architectures. The discussion of those

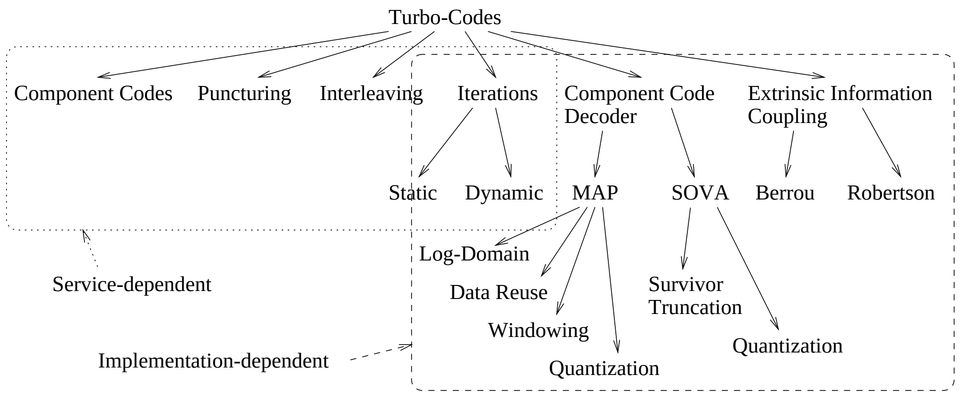

Figure 1: System design space

is omitted here, since the superior performance of Turbo-Decoding with the MAP has been clearly demonstrated [14, 11]. Additionally, implementation complexities of MAP and SOVA do not differ largely.

The MAP algorithm is too difficult to implement, basically because of the numerical representation of probabilities, non-linear functions and because of mixed multiplications and additions of these values [11]. Transforming the MAP into the *logarithmic domain* and substituting the remaining logarithms by Jacobian logarithms yields the so-called Log-MAP [11]. The Log-MAP is equivalent in decoding performance, avoids the numerical problems of the MAP and is easier to implement due to operator strength reduction [15]. Thus, from an implementation point of view, the MAP should always be implemented in the logarithmic domain. Further simplification yields the Max-Log-MAP by omitting the correction term of the Jacobian logarithms. However, this can degrade the BER.

As for the Max-Log-MAP, the inner loops of the forward and backward recursions of the Log-MAP comprise an add-compare-select (ACS) operation, but the add operation hereby additionally involves the evaluation of the correction term of the Jacobian logarithm. This correction term is best computed by using a look-up table, but this requires additional memory accesses and an extra addition for each ACS. The memory accesses can be traded for area (assuming a hardware implementation), if the look-up table is implemented as a combinational logic block. Either way, decoding speed decreases and power consumption increases for most target architectures.

Calculations independent of the decoding iteration can be performed only once, and the intermediate results can be *reused* in each MAP iteration: In case of a transition in the trellis, the corresponding transition metric of a (Max-)Log-MAP calculates as a sum of terms depending on the received symbols and the extrinsic information [11]:

$$\begin{aligned}\overline{\gamma}_i[(y_k^s, y_k^p), S_{k-1}, S_k] &= \ln \gamma_i[(y_k^s, y_k^p), S_{k-1}, S_k] \\ &= \frac{2\,y_k^s\,x_k^s(i)}{N_0} + \frac{2\,y_k^p\,x_k^p(i, S_k, S_{k-1})}{N_0} \\ &\quad + \ln \Pr\{S_k | S_{k-1}\}\end{aligned}$$

The extrinsic information $\ln \Pr\{S_k|S_{k-1}\}$ changes after each MAP whereas the terms stemming from the received symbols $y_k^s, y_k^p$ remain constant during the whole decoding process of one data block. Hence, pre-calculation of constant terms and performing e. g. five decoding iterations (with two MAPs each) saves approximately 20% of the computational complexity. The total memory size does not increase, as the pre-calculated intermediate terms replace the received symbols.

The inherent *parallelism* of TDs can be exploited to nearly arbitrarily trade-off area against speed and power consumption. (Regarding energy consumption, the situation is more complex [16]. A detailed discussion must therefore be omitted here.)

On the top-most level, the component decoders can be arbitrarily pipelined (i. e. functionally parallelized). The amount of additional buffer memory hereby depends on the pipeline depth.

On the component decoder level, the functional units of the decoder can be parallelized to some extent. For example, the functional units of a MAP decoder are branch metric calculation, forward recursion, backward recursion, and soft-output calculation. An obvious solution is to parallelize the branch metric calculation with the forward recursion and the backward recursion with the soft-output calculation. In comparison with a serialized solution, this approximately doubles the throughput. Adding a second branch metric calculation unit and a second soft-output calculation unit again roughly doubles the throughput: For the first half of the data block, forward and backward recursion are each parallelized with a branch metric calculation unit. For the second half, each recursion is parallelized with a soft-output calculation unit [17].

Parallelism on an additional level is introduced by observing that the decoding of a data block in a component decoder can be divided into the decoding of a set of overlapping sub-blocks. This is called the "sliding window technique" or "windowing" [17]. Windowing permits to further increase the throughput or to minimize the required memory size. For the window overlap being small compared to the window size, serial window processing reduces the required memory size by the ratio of block size and window size, while retaining throughput[1]. In contrast, parallel window processing increases throughput by the same factor, while memory size remains constant.

On a lower level, the trellis butterflies can obviously be processed in parallel during forward and backward recursion. It is quite evident, that throughput, area, and power consumption increase according to the degree of parallelism.

When exploiting the parallelism of Turbo-Decoding, many trade-offs are possible to obtain an optimal architecture. However, the impact of the parallelism on different levels on memory size and structure is complex and should be carefully watched. A detailed discussion will follow in a future paper.

*Quantization* and range limiting are ways to decrease the bit-width of a fixed-point formulation of a TD. Each saved bit has a significant impact on area and power of the implementation. Memory can be saved as the bit-width of the values is reduced, too. In our current research we systematically examine quantization effects and their impact on the decoding performance. First results confirm the potential of this approach.

2. Depending on the quality of the channel and the demanded quality of service, the number of decoding iterations can be varied dynamically in order to save power. Nevertheless, the decoder has to be designed to handle the worst case. So no complexity and hence area can be saved with this optimization. However, dynamic voltage scaling techniques [18] may become applicable.

3. Regarding extrinsic information coupling, we recommend Robertson's method, because it is computationally less complex, performs as well, and does not require knowledge of the extrinsic information distribution parameters. For example, this saved us up to 30% logic area for our FPGA implementation (section 5).

4. For a software implementation of a TD it is essential to find a formulation of the algorithm that fits best to the given core and memory architecture, which are highly dependent on the target device. This can be achieved by application of common data and control flow transformations. As Turbo-Decoding is a data-flow dominated application, memory mapping and register assignment are important issues for the resulting performance. In case of multiple memory banks or multiport memory, parallel transfers highly increase the bandwidth. Skillful arrangement of the data within the memory simplifies the memory access order and allows easier addressing modes, leading to incremental addressing for example.

## 5 Implementation

In this section, we present some results of two sample implementations. We have selected as a test vehicle a 2 Mbit/s TD for future radio applications with constraint length $K = 3$ component codes and five iterations. To implement this decoder, the required system[2] throughput is 33 k data blocks of 600 bits length per second.

The Max-Log-MAP algorithm has been implemented on a state-of-the-art Motorola DSP56603 to explore the performance of software Turbo-Decoding. The DSP56603 is a 16-bit fixed-point DSP specially optimized for mobile wireless applications and provides a processing performance of 80 MIPS. Its architecture allows parallel execution of certain pairs of instructions, e. g. ALU operations and memory-register transfers. Due to limits of compiler technology, an optimal exploitation of the processor's instruction-level parallelism requires hand-written DSP assembly code. However, even for hand-optimized code the resulting performance of 48.6 kbit/s on a 80 MHz DSP56603 is clearly below the needs of high-rate data services.

A detailed code analysis of the assembly code reveals that the total number of instructions consists of 36% ALU operations, 25% register $\leftrightarrow$ register transfers, 35% on-chip-memory $\leftrightarrow$ register transfers, and

[1]Practical overlap sizes are within the same order of magnitude as window sizes. However, to decode a block of 600 symbols with 60 symbols window size and 15 symbols window overlap, the number of operations is increased by only 6%, according to first order complexity estimations.

[2]"System" refers to one MAP decoder.

4% control-flow operations. The data storage scheme is highly dependent on the core architecture and is best exploited by the hand-optimized code. However, the results show, that not the computational complexity but data transfers are the bottleneck of the implementation.

The target real-time performance of 2 Mbit/s requires approximately 40 parallel DSPs (cores), which is clearly infeasible for low-cost implementations today.

Another possible target architecture and a step towards an application-specific implementation is reconfigurable hardware. We implemented the same algorithm on an Altera Flex 10K100 FPGA. In this implementation a device running at a clock frequency of 20 MHz yields a throughput of 200 kbit/s. A throughput of 2 Mbits/s can only be provided by an architecture consisting of ten parallel, pipelined FPGAs. According to our investigations, it is feasible to increase the clock frequency to 200 MHz on a state-of-the-art CMOS standard-cell library. This results in an architecture capable of performing the five TD iterations serially, i. e. without a pipeline.

## 6 Hardware/Software Co-Design

As shown above, the performance of a pure software implementation is clearly below the requirements of high-rate data services. Hardware implementations outperform software solutions, but they lack flexibility. Therefore, application of Turbo-Coding in practice enforces mixed hardware/software decoder implementations. The major advantage of a mixed hardware/software implementation is the combination of both the flexibility of a software solution, e. g. easy adaptation to different services and channels, and the superior performance of a dedicated hardware implementation. While deciding which part of the TD is to be implemented in software and hardware, special emphasis can be put either on performance or flexibility.

In a performance-driven partitioning, calculations that have to be done in each iteration are implemented in hardware, whereas parts that have to be calculated once for each data block are done in software. In a flexibility-driven approach, parts that need most flexibility are done in software, and fixed parts are realized in hardware.

## 7 Conclusions

We have rated design decisions on different levels regarding complexity, throughput, and power consumption. The large design space of TDs allows manifold optimizations, but the impacts have to be carefully watched.

Pure software solutions for high-rate mobile data services with Turbo-Codes are not applicable today. For future mobile radio applications, the speed of hardware solutions will be combined with software flexibility. We will therefore explore the huge design space of mixed hardware/software architectures for TDs in future research.

## References

[1] 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.

[2] H. Koorapaty, Y.-P. Wang, and K. Balachandran. Performance of Turbo-Codes with Short Frame Sizes. In *Proc. VTC '97*, pages 329–333, Phoenix, 1997.

[3] Third Generation Partnership Project. `http://www.3gpp.org`.

[4] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara. Serial Concatenation of Interleaved Codes: Performance Analysis, Design, and Iterative Decoding. *IEEE Transaction on Information Theory*, 44(3):909–926, May 1998.

[5] J. Hagenauer, E. Offer, and L. Papke. Iterative Decoding of Binary Block and Convolutional Codes. *IEEE Transactions on Information Theory*, 42(2):429–445, March 1996.

[6] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate. *IEEE Transactions on Information Theory*, IT-20:284–287, March 1974.

[7] J. Hagenauer and P. Hoeher. A Viterbi Algorithm with Soft-Decision Outputs and its Applications. In *Proc. Globecom '89*, pages 1680–1686, November 1989.

[8] A. Viterbi. An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes. *IEEE Journal on Selected Areas in Communications*, 16(2):260–264, February 1998.

[9] G. Fettweis and H. Meyr. High-Speed Parallel Viterbi Decoding: Algorithm and VLSI-Architecture. *IEEE Communications Magazine*, 29:46–55, May 1991.

[10] I. Kang and A. N. Willson Jr. Low-Power Viterbi Decoder for CDMA Mobile Terminals. *IEEE Journal of Solid-State Circuits*, 33(3):473–482, March 1998.

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

[12] P. Robertson. Illuminating the Structure of Code and Decoder of Parallel Concatenated Recursive Systematic (Turbo) Codes. In *Proc. Globecom '94*, pages 1298–1303, December 1994.

[13] F. Berens and T. Bing. Performance of low complexity Turbo-Codes in the UTRA-TDD-Mode. In *Proc. VTC Fall '99*, Amsterdam, The Netherlands, September 1999.

[14] P. Jung. Comparison of Turbo-Code Decoders Applied to Short Frame Transmission Systems. *IEEE Journal on Selected Areas in Communications*, 14(3):530–537, April 1996.

[15] J. Sacha and M. Irwin. The Logarithmic Number System for Strength Reduction in Adaptive Filtering. In *Proc. ISLPED '98*, pages 256–261, Monterey, CA, August 1998.

[16] C. Schurgers, M. Engels, and F. Catthoor. Energy Efficient Data Transfer and Storage Organization for a MAP Turbo Decoder Module. In *Proc. ISLPED '99*, San Diego, CA, August 1999.

[17] H. Dawid. *Algorithmen und Schaltungsarchitekturen zur Maximum a Posteriori Faltungsdecodierung*. PhD thesis, RWTH Aachen, Shaker Verlag, Aachen, Germany, 1996.

[18] T. Pering, T. Burd, and R. Brodersen. The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms. In *Proc. ISLPED '98*, pages 76–81, Monterey, CA, August 1998.