Channel Decoder Architecture for 3G Mobile Wireless Terminals

Friedbert Berens
STMicroelectronics N.V.
39, Chemin du Champ-des-Filles
CH-1228 Plan-les-Quates/Geneva
Switzerland
friedbert.berens@st.com

Gerd Kreiselmaier, Norbert Wehn
Microelectronic System Design Research Group
University of Kaiserslautern
Erwin-Schroedinger-Straße
67663 Kaiserslautern, Germany
{kreiselmaier, wehn}@eit.uni-kl.de

Abstract

Channel coding is a key element of any digital wireless communication system since it minimizes the effects of noise and interference on the transmitted signal. In third-generation (3G) wireless systems channel coding techniques must serve both voice and data users whose requirements considerably vary. Thus the Third Generation Partnership Project (3GPP) standard offers two coding techniques, convolutional-coding for voice and Turbo-coding for data services. In this paper we present a combined channel decoding architecture for 3G terminal applications. It outperforms a solution based on two separate decoders due to an efficient reuse of computational hardware and memory resources for both decoders. Moreover it supports blind transport format detection. Special emphasis is put on low energy consumption.

1 Introduction

Today's information society demands access to huge amounts of data anywhere and at any time. In digital communication systems bandwidth and transmission power are critical resources. Thus advanced communications systems have to rely on sophisticated channel coding schemes. Channel coding allows to reduce the transmission power by maintaining the Quality of Service (QoS) or vice versa to improve the QoS for given transmission power. 3G based systems [14] have to carry both voice and data traffic. Thus the standards offers different data rates and coding techniques to satisfy the varying latency, throughput and error performance requirements. E.g. voice traffic requires low latency and can tolerate higher error rates than data services which require lower error rates and higher throughput, but can tolerate a larger latency. Thus the 3GPP standard for UMTS specifies two coding techniques which differ in their complexity and characteristics: convolutional-codes for voice and control, and Turbo-codes for data traffic [15].

The implementation complexity of the encoders is negligible. Both encoders are trellis encoders which map a long input sequence to a coded data stream. They comprise only some shift registers and in the case of Turbo-codes an interleaved address generator. The decoders are based on trellis propagation of the received input sample sequence to calculate a maximum likelihood sequence/bit detection. In contrast to the encoders efficient decoder implementation is a challenging design task. Especially *Turbo-decoding* is very complex and belongs to the most computationally intensive baseband tasks performed by the receiver. Decoding involves information exchange between two soft-in/soft-out (SISO) component decoders in an iterative loop. The maximum a posteriori (MAP) and the soft-out Viterbi algorithm (SOVA) are candidates for these component decoders. In [18, 10] it was shown that the MAP algorithm outperforms the SOVA algorithm from an communications and implementation point of view. *Convolutional-decoding* can be done by the well known Viterbi algorithm which makes only a single decoding pass over the received samples or by the already mentioned more complex MAP algorithm which makes multiple passes.

Decoder implementations for terminal applications are driven by low cost (small silicon area) and low power consumption. Thus the design of an efficient decoder architecture has to be driven by these two cost criteria. In this paper we present a combined channel decoding architecture for 3G terminal applications. It outperforms a solution based on two separate decoders due to an efficient reuse of computational hardware and memory resources for both decoders. Moreover it supports blind transport format detection for transport channels (BTFD)[15].

2 Turbo-Decoder

Forward error correction is enabled by introducing parity bits. For Turbo-codes, the original information, denoted as systematic information ($\vec{x}^s$), is transmitted together with the parity information ($\vec{x}^{1p}, \vec{x}^{2p}_{\text{int}}$). The encoder for 3GPP consists of two recursive systematic convolutional (RSC) encoders with constraint length $K = 4$. One RSC encoder works on the block of information in its original, the other one in an interleaved sequence to break open correlations and yield timing diversity, see Figure 1 a). On the receiver side, a component decoder is provided for each RSC encoder.

Each transmitted data block is iteratively decoded, see Figure 1 b). The systematic information $\vec{\Lambda}^s$ and the parity information $\vec{\Lambda}^{1p}$ serve as inputs of the first component decoder (MAP1). The soft-output of MAP1 ($\vec{\Lambda}^1$) reflects its confidence on the received bits of being sent either as "0" or "1". These soft-outputs are modified ($\vec{\Lambda}^{1e}$) and then interleaved in the same manner as in the encoder and passed to the second component decoder (MAP2) as a-priori information ($\vec{\Lambda}^{2a}_{\text{int}}$). The second component decoder uses this *extrinsic* information to bias its estimation comprising the interleaved systematic information $\vec{\Lambda}^s_{\text{int}}$ and the parity information $\vec{\Lambda}^{2p}_{\text{int}}$ of the second encoder. The soft-outputs are again passed to MAP1, and so on. This process runs between 5 to 10 iterations.

Given the received samples of systematic and parity bits (*channel values*) for the whole block (y^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:

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

Equation 1 can be expressed using three probabilities, which refer to the encoder states S^m_k, 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^m_k and $S^{m'}_{k+1}$ 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 [10].

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^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^N_{k+1}:

$$\beta^k_m = \sum_{m'} \beta^{k+1}_{m'} \cdot \gamma^{k,k+1}_{m,m'}$$

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 implies many multiplications and has thus been ported to the logarithmic domain resulting in the *Log-MAP algorithm* [10]. Multiplications turn into additions and additions into maximum selections with additional correction terms. The resulting max^* operation is defined as:

$$max^*(\delta_1, \delta_2) = max(\delta_1, \delta_2) + ln(1 + e^{-|\delta_2 - \delta_1|}). \tag{3}$$

This transformation does not decrease the communications performance. Arithmetic complexity can further be reduced by omitting the correction term (*Max-Log-MAP algorithm*) which leads to a slight loss in communications performance (about 0.1–0.2dB). Log-MAP and Max-Log-MAP algorithm are common practice in decoder implementations.

State-of-the-art implementations of the MAP algorithm for 3G based 8-state Turbo-codes usually calculate all 8 state metrics of a time step in parallel, thus needing 8 so called ACS* units which evaluate the max^* operation. α state metrics are first computed, afterwards the β state metrics. The soft-output calculation is done in parallel to the β calculation. Thus the α values have to be stored since they are required for the soft-output calculation. The size of this memory corresponds to a complete block. Windowing [4, 5] allows to break open the data dependencies in the state metrics. This allows to subdivide the data block into sub-blocks - called windows - which can be processed independently. If the different windows are processed sequentially on the same MAP unit the memory for the α values can be reduced from a full block to a window size. The size of a window is significantly smaller than the block size, it ranges typically between 32 and 128. Windowing implicates additional computations during the so called acquisition, however the reduction in the α-memory prevails these extra computations.

The size of the input RAMs for the systematic and parity information and the output RAM is determined by the block sizes which are defined in the 3GPP standard. The output RAM serves also as storage for the a-priori values. One soft-output memory is sufficient because the calculated soft-outputs are always written to the previous read position of the a-priori information. Thus no RAM is needed for the

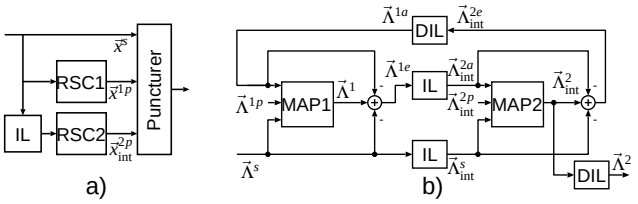

Figure 1. Turbo-Encoder and Turbo-Decoder

interleaved soft-outputs. Moreover, only an interleaver table is needed but no deinterleaver. It becomes obvious, that for large block sizes memory dominates a Turbo-decoder architecture. Thus memory size and access have to be minimized to improve implementation efficiency.

3 Convolutional-Decoder

In 3GPP two different non-systematic convolutional (NSC) encoders with a constraint length $K = 9$ are specified. Thus the number of states ($2^{K-1} = 256$) is significantly larger than the number of states in the Turbo-encoder ($= 8$).

Convolutional-decoding, in contrast to Turbo-decoding, is non-iterative. It can be done by the well known Viterbi algorithm (VA) which detects the maximum likelihood sequence or the MAP algorithm which detects the maximum likelihood bit. Thus the MAP is more accurate. Moreover it provides soft information and thus confidence values in the decoding decisions. Although there is no interleaver and the block size is much smaller than for Turbo-codes, the architecture of a 3GPP convolutional-decoder is also dominated by memory. The I/O memories are, compared to the Turbo-decoder, rather small. The α-memory, however, exceeds that of the Turbo-decoder by a factor of 32 assuming the same window size. This is constituted in the large number of states of the convolutional-decoder.

Publications on implementation issues of convolutional-codes incorporate, to the best of our knowledge, only the classical Viterbi algorithm. They either address implementations with hard-decision-output [3], or with soft-output (SOVA) but only a small number of states [8].

4 Blind Transport Format Detection

In 3GPP data transport services on the physical layer are realized through the usage of *Transport Channels*. Each Transport Channel can have several different Transport Formats, each Transport Format specifying the respective Transport Block Size, see [15]. At the receiver, the Transport Format of each Transport channel has to be detected. This can be done by using a dedicated Transport Format Combination Indicator (TFCI) field. As soon as the TFCI is detected, the Transport Format combination, and hence the Transport Formats of the individual Transport Channels are known. Beside the TFCI based Transport Format detection the 3GPP standard defines the requirement to blindly detect the Transport Format in use. That is, using *only* the received block of data which is convolutional-encoded to estimate the correct size of that Transport Block. This option is used in order to reduce the TFCI symbol overhead in the transmission frame and thus to increase the effective payload. Furthermore one explicitly blindly detected Transport Channel can be used as a *guiding* Transport Channel. The detection of a specific Transport Format used by this channel defines the Transport Formats used by *all other guided channels*. Thus an efficient Blind Transport Format Detection (BTFD) is crucial. BTFD is performed by decoding of convolutional-encoded blocks. Different coded block sizes (up to 16) are possible and the correct one has to be detected by using the CRC of these blocks.

Only few publications on BTFD implementation exist [1]. Proposals for BTFD processing can be found in the annex of the 3GPP standard [15]. State-of-the-art techniques are based on a Viterbi decoding of this block. All 16 possible block sizes are elaborated and the corresponding CRCs checked. However, especially for small CRC length, this detection process typically yields ambiguity for the block length. Thus an additional evaluation metric becomes necessary. In [9] we presented for the first time a BTFD method using LLRs to overcome the mentioned ambiguity and to improve the detection performance. We use the MAP algorithm and exploit the reliability information. An optimized windowing scheme ensures BTFD decoding of several block sizes in parallel, thus significantly reducing computation time. Simulations showed the superior performance of LLR-assisted BTFD, outperforming BTFD methods based on Viterbi decoding.

5 3GPP Parameters

The 3GPP standard specifies the encoder structure and parameters like block size and throughput requirements [14]. The maximum block size of the Turbo-codes including the data, frame quality indicator (CRC) and two reserved bits is set to 5114. During encoding, an encoder output tail sequence is added which appends another 3 bits, the tail bits, for both systematic and parity information of each encoder. The systematic information of the second encoder, working on the interleaved data set, is not transmitted. This results in a coding rate of $1/3$. The transfer function of one RSC encoder is:

$$G(D) = [1, \frac{g_1(D)}{g_0(D)}]$$

The generator function g_0 equals 13_8, and g_1 equals 15_8 in octal representation. The throughput requirements needed for terminal applications are 384kbps for the Turbo-decoder.

The convolutional-encoders are non-systematic convolutional (NSC) encoders. Two encoders are specified in the standard: one generates two parity informations, the other three. This results in coding rates of $1/2$ and $1/3$. The generator functions for the rate $1/2$ code are G_0 equals 753_8 and G_1 equals 561_8, and for the rate $1/3$ code they are G_0 equals 557_8, G_1 equals 663_8, and G_2 equals 711_8. The maximum block length is 504, the tail sequence, which forces the encoder back into the all-zero state after all data bits are encoded, adds another 8 bits to the block. The maximum throughput of the convolutional-decoder is specified to 12.2kbps.

BTFD is performed on a set of up to 16 different coded block sizes. The data block is convolutional-encoded, possible CRC sizes are 24, 16, 12 or 8 bits.

6 Architecture

As mentioned in the previous sections the terminal architecture has to support the two channel coding schemes and BTFD. To the best of our knowledge only Bickerstaff *et al.* presented in [2] a unified Turbo/Viterbi channel decoder architecture, reusing the state metric unit and LLR-RAM for both codes. However, this decoder targets base station applications and does not support BTFD. We have shown that the MAP algorithm is an excellent building block to carry out terminal functionalities and can be efficiently reused. Although a Viterbi decoding unit is smaller than a MAP decoder unit, the reuse of the MAP for *both* decoding schemes results in a more efficient architecture. Thus the decoder architecture is based on a *combined Turbo and Convolutional decoder architecture*, utilizing a MAP component decoder. Both Log-MAP and Max-Log-MAP are implemented and can be selected via software. Since the Log-MAP algorithm is very sensitive to SNR mismatch, the decoder can be switched between the optimal Log-MAP and suboptimal but SNR-insensitive Max-Log-MAP algorithm.

An efficient merge of both decoders requires a maximum *reuse of computational hardware and memory* is an attractive alternative which yields much less area than two separate hardware units. Important to note is that the whole architecture is dominated by memory. Therefore an efficient memory partitioning is crucial for the architectural efficiency. Both decoders differ substantially in the memory requirements. The Turbo-decoder (TC) is dominated by the I/O memories due to the large block sizes. The α-RAM is negligible small. In case of the convolutional-decoder (CC) it is vice versa: the α-RAM is significantly larger than the I/O memories (see Table 1). Therefore it is not possible to use the same I/O RAMs of both TC and CC. The same counts for the α-RAMs. As the TC I/O-RAMs and the CC α-RAM store about the same amount of data, these two memories should apparently be merged instead. The TC I/O memories can be partitioned in such a way that it is possible to use them as CC α-RAM. To built this memory, each of the TC I/O RAMs is split into three separate RAMs resulting in total 12 RAMs each of size 1728x6. These RAMs are then concatenated together with an additional 1728x16 RAM, forming the required bit-width for the storage of 8 state metrics in parallel. This memory sharing enables a window size of 54 for CC decoding.

	I/O-RAMs	α-RAM	Others	Total
Turbo Dec.	122.880	5.632	384	128.896
Convol. Dec.	12.288	152.064	5.632	169.984
Comb. Arch.	169.984		6.016	176.000

Table 1. Memory Utilization of Decoders

Table 1 lists the necessary memory (in bits) for a single Turbo-decoder (window size = 64), a single convolutional-decoder (window size = 54), and for the combined decoder. I/O data for both decoder are quantized to 6 bit. The table shows that the overhead of the memory for the combined architecture is about 4% compared to the convolutional decoder and saves about 60% compared to two separate decoders.

In addition many of the computational units can be shared. Forward and backward recursions for Turbo-decoding can be calculated serially on the same hardware unit (SMU) due to the moderate throughput requirements (384kbps). All 8 states are processed in parallel within an SMU. The intermediate metric values are stored in registers when moving from the actual trellis step to the next one (SMU update). A branch metric unit (BMU) calculates the necessary branch metrics. The LLR values are calculated in a dedicated unit (LLRU) which is composed of two pipelined trees which perform additions, comparisons and subtractions. As already mentioned the LLR calculation is done in parallel to the backward recursion. Thus only the α values have to be stored in a memory (TC-alpha RAM), the β-values are directly consumed after calculation. A window size of 64 is a good trade-off between computational overhead and memory size.

The convolutional-decoder is based on a single MAP with a structure which is similar to that of the Turbo-decoder. This decoder requires three input memories due to the code rate 1/3. In contrast to the Turbo-codes, the block-size of the convolutional-codes is only 512, thus the corresponding input RAM sizes are much smaller. Since the constraint length of the convolutional-code is 9, 256 states have to be decoded in each trellis step which is 32 times more than in the Turbo-decoder. But due to the moderate throughput requirements (12.2kbps) a partially sequential processing is possible, only 8 states have to be processed in parallel in the SMU unit. Two sets of state metrics have to be stored because the state metrics are calculated recur-

sively. The code structure demands that the previous metrics have to be available until all the actual ones are computed. Thus two times 256 state metrics have to be stored temporarily within the SMU. This leads to four temporary RAMs for the state metrics within the SMU update in addition to the α-RAM.

The SMU of the TC can be reused for both decoders with only small modifications. Due to an efficient data ordering and optimized memory organization only a set of 2:1 multiplexers has to be added to the SMU which slightly increases the critical path. Eight state metrics are produced in each cycle. Subcomponents of the pipeline tree of the TC LLRU can be reused for the CC as well, only small modifications are necessary. During convolutional-decoding backward recursion and LLR calculation are computed simultaneously. Due to the non-recursive code structure of the convolutional-code, no branch metrics are necessary to calculate an LLR which is different to the Turbo-decoder. The LLR unit calculates the minimum of the sums of four even and four odd states (representing the last bit being transmitted as "0" or "1", respectively) in a pipeline similar to the Turbo-decoder LLRU. The soft-output is the difference of all even and all odd state metric sums of one time step. Thus two additional feedback loops are inserted. Figure 2 shows the architecture of the combined architecture.

Low power is extremely important in terminal applications. Thus low power techniques have to be applied on all abstraction levels in the design phase. Obviously the largest reduction can be achieved on system level. We apply iteration control, optimized quantization/ renormalization and interleaver table management on system level to minimize power. On RT- and gate level we extensively exploit the well known techniques like power-down mode, clock gating and low frequency. We focus in the following on the system level.

The number of necessary iterations for Turbo-decoding can differ from block to block e.g. due to a fading channel. Sometimes more iterations are necessary, sometimes less iterations. Occasionally decoding is not possible at all, even with an infinite number of iterations. Thus the iteration process can be stopped immediately. Unnecessary iterations waste energy. An efficient *iteration control* helps to reduce energy. In [17] we have shown that iteration control is the most efficient technique to reduce energy in Turbo-decoding. Many papers are published on iteration control e.g. [12, 11] which mainly focus on efficient stop criteria for decodable blocks. In [7, 6] we have presented a new stop criterion which is based on the monitoring of the soft-output values of the two MAP-Decoders combined with CRC checks. This criterion has a very low implementation complexity and allows to stop iterations for decodable *and* undecodable blocks. It saves up to 70% energy compared to a solution which is based on a fixed number of iterations. This sophisticated iteration control is implemented in our architecture. The iteration control unit is also reused for BTFD (see Figure 2). Interleaving can be done by usage of a dedicated hardware interleaver address generator (IAG) or by usage of interleaver tables stored in RAM. The tables are programmed by an external IAG. For terminal applications the block size and thus the interleaver sequence can change, but is constant within one time transmission interval. Therefore, from an energy efficiency point of view, a table based interleaver is more efficient, because the address sequences have to be computed only once for a dedicated block size, instead of recomputed for every block. The interleaver RAM is an external unit, which is updated on demand. Appropriate table buffering ensures seamless decoding of varying block sizes. Quantization and an efficient renormalization helps to reduce the area and energy. In [19] we presented an efficient quantization and renormalization scheme for Turbo-decoders which saves up to 22% energy compared to state-of-the-art quantization and renormalization.

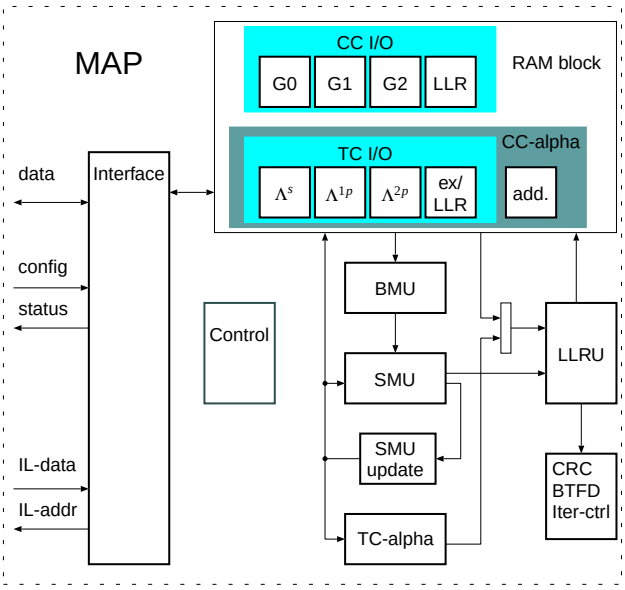

Figure 2. Combined 3GPP Terminal Decoder Architecture

7 Synthesis Results

On system level the decoder was developed and validated with CoCentric System Studio (CCSS) [13] in a complete UMTS-downlink chain according to 3GPP reference channels (static, multi-path fading, *etc.*) [16]. On RT-level the architecture was implemented as a fully synthesizable VHDL model. Synthesis was carried out with the Synopsys

Area Logic	655.088 μm^2
Area RAMs	2.370.592 μm^2
Total Area	3.025.680 μm^2

Table 2. Area of Combined 3GPP Channel Decoder

Design Compiler for a state-of-the-art 0.18μm technology under worst case conditions (1.2V, 125C). To reduce energy the frequency (70MHz) is moderate and was not set to the technology limit. The final area is listed in Table 2. The throughput fulfills the 3GPP terminal requirements up to 2Mbit/s. The netlist was validated on a ST emulation platform. Chip prototyping is in progress.

8 Conclusions

In this paper we presented a fully 3GPP compliant channel decoder for mobile terminals which is based on the MAP algorithm. The decoder uses a combined architecture to decode convolutional- and Turbo-codes, furthermore a sophisticated blind transport format detection is supported. Due to efficient memory partitioning and hardware reuse this architecture outperforms a solution based on two separate decoders.

References

[1] W. K. M. Ahmed. Block-Size Estimation and Application to BTFD for 3GPP UMTS. In *Proc. 2001 Global Telecommunications Conference (Globecom '01)*, pages 3045–3049, 2001.

[2] M. A. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, G. Zhou, L. M. Davis, G. Woodward, C. Nicol, and R. Yan. A Unifi ed Turbo/Viterbi Channel Decoder for 3GPP Mobile Wireless in 0.18-μm CMOS. *IEEE Journal of Solid-State Circuits*, 37(11):1555–1564, Nov. 2002.

[3] Y. Chang, H. Suzuki, and K. K. Parhi. A 2-Mb/s 256-State 10-mW Rate-1/3 Viterbi Decoder. *IEEE Journal of Solid-State Circuits*, 35(6):826–834, June 2000.

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

[5] H. Dawid and H. Meyr. 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)*, pages 193–197, Toronto, Canada, Sept. 1995.

[6] F. Gilbert, F. Kienle, and N. Wehn. Low Complexity Stopping Criteria for UMTS Turbo-Decoders. In *Proc. 2003-Spring Vehicular Technology Confernce (VTC Spring '03)*, Jeju, Korea, Apr. 2003.

[7] F. Gilbert, A. Worm, and N. Wehn. Low Power Implementation of a Turbo-Decoder on Programmable Architectures. In *Proc. 2001 Asia South Pacific Design Automation Conference (ASP-DAC '01)*, pages 400–403, Yokohama, Japan, Jan. 2001.

[8] O. J. Joeressen, M. Vaupel, and H. Meyr. High-Speed VLSI Architectures for Soft-Output Viterbi Decoding. *Journal of VLSI Signal Processing Systems*, 8:169–181, 1994. Kluwer Academic Publishers, Boston.

[9] G. Kreiselmaier and F. Berens. Method of blindly detecting a transport format of an incident convolutional encoded signal, and corresponding convolutional code decoder. Patent Application, Apr. 2003.

[10] 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.

[11] R. Y. Shao, S. Lin, and M. C. P. Fossorier. Two Simple Stopping Criteria for Turbo Decoding. *IEEE Transactions on Communications*, 47(8):1117–1120, Aug. 1999.

[12] A. Shibutani, H. Suda, and F. Adachi. Reducing Average Number of Turbo Decoding Iterations. *Electronic Letters*, 35(9):701–702, Apr. 1999.

[13] Synopsys Inc. `http://www.synopsys.com`.

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

[15] Third Generation Partnership Project. 3rd Generation Partnership Project; Technical Specifi cation Group Radio Access Network; Multiplexing and channel coding(FDD) 3GPP TS 25.212 V5.3.0. `www.3gpp.org`, Dec. 2002.

[16] Third Generation Partnership Project. 3rd Generation Partnership Project; Technical Specifi cation Group Radio Access Network; UE Radio Transmission and Reception (FDD) 3GPP TS 25.101 V5.3.0. `www.3gpp.org`, June 2002.

[17] M. J. Thul, T. Vogt, F. Gilbert, and N. Wehn. Evaluation of Algorithm Optimizations for Low-Power Turbo-Decoder Implementations. In *Proc. 2002 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '02)*, pages 3101–3104, Orlando, Florida, USA, May 2002.

[18] 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.

[19] A. Worm, H. Michel, F. Gilbert, G. Kreiselmaier, M. J. Thul, and N. Wehn. Advanced Implementation Issues of Turbo-Decoders. In *Proc. 2nd International Symposium on Turbo Codes & Related Topics*, pages 351–354, Brest, France, Sept. 2000.