# Design and Implementation of a Parallel Turbo-Decoder ASIC for 3GPP-LTE

Christoph Studer, *Student Member, IEEE*, Christian Benkeser *Member, IEEE*, Sandro Belfanti, and Qiuting Huang *Fellow, IEEE*

***Abstract*—Turbo-decoding for the 3GPP-LTE (Long Term Evolution) wireless communication standard is among the most challenging tasks in terms of computational complexity and power consumption of corresponding cellular devices. This paper addresses design and implementation aspects of parallel turbo-decoders that reach the 326.4 Mb/s LTE peak data-rate using multiple soft-input soft-output decoders that operate in parallel. To highlight the effectiveness of our design-approach, we realized a 3.57 mm$^2$ radix-4-based 8× parallel turbo-decoder ASIC in 0.13 μm CMOS technology achieving 390 Mb/s. At the more realistic 100 Mb/s LTE milestone targeted by industry today, the turbo-decoder consumes only 69 mW.**

***Index Terms*—3G mobile communication, LTE, parallel turbo-decoder, ASIC implementation, low-power, radix-4**

## I. Introduction

During the last few years, 3G wireless communication standards, such as HSDPA [2], firmly established themselves as an enabling technology for data-centric communication. The advent of smart-phones, netbooks, and other mobile broadband devices finally ushered in an era of throughput-intensive wireless applications. The rapid increase in wireless data traffic now begins to strain the network capacity and operators are looking for novel technologies enabling even higher data-rates than those achieved by HSDPA. Recently, the new air interface standard LTE (Long Term Evolution) [3] has been defined by the standards body 3GPP and aims at improving the data-rates by more than 30× (compared to that of HSDPA) in the next few years. Theoretically, LTE supports up to 326.4 Mb/s [4], whereas the industry plans to realize the first milestone at about 100 Mb/s in 1-or-2 years.

LTE specifies the use of turbo-codes to ensure reliable communication. Parallel turbo-decoding, which deploys multiple soft-input soft-output (SISO) decoders operating concurrently, will be the key for achieving the high data-rates offered by LTE. However, the implementation of such will be among the main challenges in terms of computational intensity and power consumption. The fact that none of the recently reported parallel turbo-decoders [5]–[7] achieves the LTE peak data-rate or provides desirable power consumption for battery-powered devices of less than 100 mW at the 100 Mb/s milestone, indicates that the architecture design for such decoders is a challenging task.

*1) Contributions:* In this work, we discuss concepts and architectures which allow for the power-efficient implementation of high-throughput parallel turbo-decoding for LTE. To this end, we investigate the associated throughput/area tradeoffs for the identification of the key design parameters and optimize the most crucial design blocks. To alleviate the design-inherent interleaver bottleneck, we describe a memory architecture that supports the bandwidth required by LTE and present a general architecture solution—referred to as Master-Slave Batcher network—suitable for maximally-vectorizable contention-free interleavers. We furthermore detail a radix-4-based SISO decoder architecture that enables high-throughput turbo-decoding. As a proof-of-concept, we show an 8× parallel ASIC prototype achieving the LTE peak data-rate and the 100 Mb/s milestone at low power, and finally compare the key characteristics to that of other measured turbo-decoder ASICs [5]–[7].

*2) Outline:* The remainder of the paper is organized as follows. Section II reviews the principles of turbo-decoding and details the algorithm used for SISO decoding. The parallel turbo-decoder architecture is presented in Section III and the corresponding throughput/area tradeoffs are studied. The interleaver architecture is detailed in Section IV and Section V describes the architecture of the SISO decoder. Section VI provides ASIC-implementation results and a comparison with existing turbo-decoders. We conclude in Section VII.

## II. Turbo-Decoding for LTE

Turbo codes [8], capable of achieving close-to-Shannon capacity and amenable to hardware-efficient implementation, have been adopted by many wireless communication standards, including HSDPA [2] and LTE [3]. The turbo encoder specified in the LTE standard is illustrated in the left-hand side (LHS) of Fig. 1 and consists of a feed-through, two 8-state recursive convolutional encoders (CEs), and an interleaver. The feed-through passes one block of $K$ information bits $x_k$, $k = 0, \ldots, K-1$, to the output of the encoder, which are then referred to as systematic bits $x_k^{\mathrm{s}} = x_k$. From the systematic bits, the first CE generates a sequence of parity bits $x_k^{\mathrm{p1}}$. The

This paper was presented in part at the IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, Feb. 2009 [1].

C. Studer is with the Communication Technology Laboratory (CTL), ETH Zurich, 8092 Zurich, Switzerland, (e-mail: studerc@nari.ee.ethz.ch). C. Benkeser and Q. Huang are with the Integrated Systems Laboratory (IIS), ETH Zurich, 8092 Zurich, Switzerland (e-mail: benkeser@iis.ee.ethz.ch; huang@iis.ee.ethz.ch).

The authors would like to thank S. Schläpfer and F. Gürkaynak for their assistance during the ASIC design. Furthermore, the authors gratefully acknowledge the support of H. Bölcskei, A. Burg, N. Felber, W. Fichter, and H. Kaeslin.

Digital Object Identifier XXX-XXX-XXX

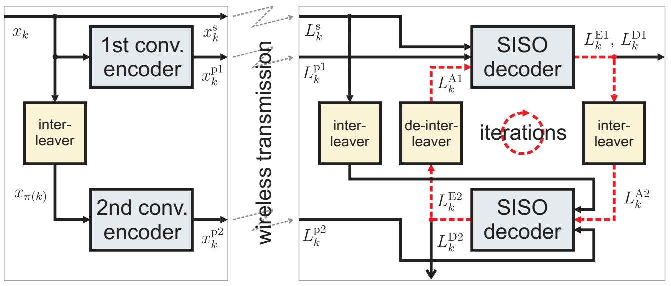


Fig. 1. Left: Parallel-concatenated turbo-encoder. Right: Simplified block-diagram of a turbo-decoder.

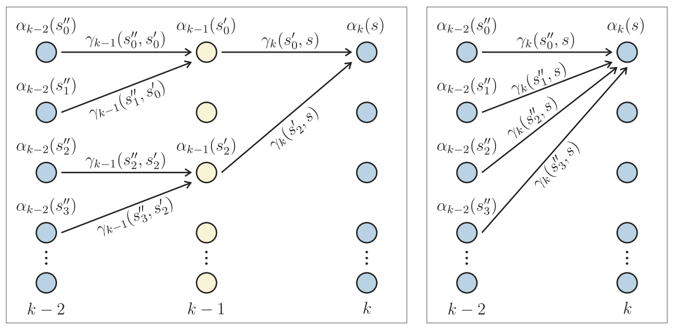


Fig. 2. Left: Radix-2 forward state-metric recursion for two trellis-steps. Right: Radix-4 forward state-metric recursion.

second CE receives an interleaved sequence of the information bits $x_{\pi(k)}$, where $\pi(k)$ stands for the interleaved address associated with address $k$, and generates a second sequence of parity bits $x^{\mathrm{p2}}$. The systematic bits are then transmitted, along with both parity-bit sequences, over the wireless channel. In the receiver, a soft-output detector computes reliability information in the form of log-likelihood ratios (LLRs) for the transmitted bits $x_k^{\mathrm{s}}$, $x_k^{\mathrm{p1}}$ and $x_k^{\mathrm{p2}}$ [8]; the resulting LLRs $L_k^{\mathrm{s}}$, $L_k^{\mathrm{p1}}$ and $L_k^{\mathrm{p2}}$ indicate the probability of the corresponding bits being a binary 1 or 0.

## *A. Turbo-Decoding Algorithm*

Decoding of turbo-codes is usually performed with the algorithm proposed in [8]. The main idea is depicted on the right-hand side (RHS) of Fig. 1 and amounts to iteratively exchanging *extrinsic* LLRs $L_k^{\mathrm{E1}}$ and $L_k^{\mathrm{E2}}$ between the two SISO decoders (SDs) to improve the error-rate performance successively. The first and second SD perform decoding of the convolutional code generated by the first or the second CE, respectively. One pass by both the first and the second SD is referred to as a full-iteration; the operation performed by a single SD a half-iteration. The total number of full-iterations for each code block is denoted by $I$ (e.g., 11 half-iterations correspond to $I = 5.5$).

Each SD computes intrinsic a-posteriori LLRs $L_k^{\mathrm{D1}}$ and $L_k^{\mathrm{D2}}$, for the transmitted bits, based on the systematic LLRs in natural $L_k^{\mathrm{s}}$ or interleaved order $L_{\pi(k)}^{\mathrm{s}}$, on the parity LLRs $L_k^{\mathrm{p1}}$ or $L_k^{\mathrm{p2}}$, and on the so-called *a-priori* LLRs $L_k^{\mathrm{A1}}$ or $L_k^{\mathrm{A2}}$. For the first half-iteration, the a-priori LLRs are set to zero (i.e., $L_k^{\mathrm{A1}} = 0, \forall k$). In subsequent iterations, each SD $i \in \{1, 2\}$ uses the extrinsic LLRs $L_k^{\mathrm{E}i} = L_k^{\mathrm{D}i} - \left(L_k^{\mathrm{s}} + L_k^{\mathrm{A}i}\right)$ computed by the other SD in the previous half-iteration as a-priori LLRs, i.e., $L_k^{\mathrm{A1}} = L_{\pi^{-1}(k)}^{\mathrm{E2}}$ and $L_k^{\mathrm{A2}} = L_{\pi(k)}^{\mathrm{E1}}$ (see Fig. 1). After a given number of half-iterations, the turbo-decoder generates estimates for the information bits based on the sign of the intrinsic LLRs.

## *B. Radix-4 Max-Log M-BCJR Algorithm*

The maximum a-posteriori (MAP) SISO decoding algorithm developed by Bahl, Cocke, Jelinek, and Raviv (BCJR) [9] forms the basis of the SD used in this work. The BCJR algorithm resembles the Viterbi algorithm [10] and traverses a trellis representing the convolutional code to compute the intrinsic LLRs $L_k^{\mathrm{D1,D2}}$. Fig. 2 shows such a trellis, with nodes corresponding to the states of the CEs and branches indicating admissible state-transitions. Each transition from a state $s'$ (trellis-step $k-1$) to $s$ (trellis-step $k$) is associated with a branch-metric $\gamma_k(s', s)$ (refer to [11], [12] for details). The BCJR algorithm in its original form is impractical due to large memory requirements and the computation of transcendental functions. We adopt an approximate algorithm [11], [13], as briefly described below.

*1) Max-log approximation:* The BCJR algorithm traverses the trellis in both forward and backward directions to compute the state-metrics $\alpha_k(s)$ and $\beta_k(s)$ recursively for all eight states. To avoid transcendental functions, we apply the max-log approximation to the forward state-metric recursions [11]

$$\begin{aligned}\alpha_k(s) = \max\big\{&\alpha_{k-1}(s'_0) + \gamma_k(s'_0, s),\\ &\alpha_{k-1}(s'_2) + \gamma_k(s'_2, s)\big\} \qquad (1)\end{aligned}$$

where $s'_0$ and $s'_2$ correspond to the two possible predecessor states of $s$ (see Fig. 2). The backward state-metrics $\beta_k(s')$ are computed similarly to (1) in the opposite direction. Both recursions can be performed efficiently based on hardware-friendly add-compare-select (ACS) operations.

After all forward and backward state-metrics, the intrinsic LLRs are calculated. To this end, the SD considers the state transitions $(s', s)$ associated with $x_k^{\mathrm{s}} = 0$ and those with $x_k^{\mathrm{s}} = 1$ and computes the intrinsic LLRs according to

$$\begin{aligned}L_k^{\mathrm{D1,D2}} \approx &\max_{(s',s):x_k^{\mathrm{s}}=0} \big\{\alpha_{k-1}(s') + \gamma_k(s', s) + \beta_k(s)\big\}\\ &- \max_{(s',s):x_k^{\mathrm{s}}=1} \big\{\alpha_{k-1}(s') + \gamma_k(s', s) + \beta_k(s)\big\}. \qquad (2)\end{aligned}$$

The max-log approximation entails a mismatch in the output LLRs, which can (at least partially) be compensated by a technique known as *extrinsic scaling* [14], [15] (cf. Section VI-A).

*2) Windowing:* Computation of the intrinsic LLRs (2) requires storage of either all forward or all backward state-metrics. For the maximum code-block length specified in LTE, $K = 6144$, $8 \times 6144$ state-metrics need to be stored. To significantly reduce such large memory requirements, *windowing* is usually employed [13]. In this approach the trellis is processed in small windows of $M$ trellis-steps and the intrinsic LLRs are computed only on the basis of the state-metrics obtained within each window. The corresponding procedure, summarized next, will be referred to as the M-BCJR algorithm.

The forward recursion computes the $\alpha_k(s)$ as in (1) and stores the $M$ forward state-metrics associated with the $m$th window. Since the backward recursion progresses from the end of a window to its beginning, suitable initial values have to be generated. To this end, a *dummy* backward recursion is carried out in the next window $m+1$ to provide initial values for the backward state-metrics $\beta_k(s)$ computed in window $m$. With the aid of the stored forward state-metrics of the $m$th window, the intrinsic LLRs (2) are computed simultaneously with the backward state-metric recursion. As a consequence of windowing, the M-BCJR algorithm can be started at arbitrary steps in the trellis, an ability essential for parallel turbo-decoding (see Section III). To this end, a *dummy forward-recursion* is carried out *once* to compute the state-metrics $\alpha'_k(s)$ for one window, which are then used as initial state-metrics for the remaining forward recursions.

Numerical simulations for the rate-$1/3$ LTE turbo-code show that a window length of $M = 30$ yields close-to-optimal performance (see Fig. 9). Code rates closer to 1 have also been specified in LTE to take advantage of any exceptionally high-SNR/low-EVM scenarios to achieve additional throughput improvements. To support such higher code-rates, our radix-4 M-BCJR architecture can be easily reconfigured to support 2-to-3 times larger window lengths that will be required for optimal performance [16]. This mainly involves increasing the capacity of the M-BCJR memories by a factor of 2-to-3, which will lead to about 30% increase in the overall chip area.

*3) Radix-4 recursion:* The throughput of LTE turbo-decoders can be enhanced by radix-4 state-metric recursions [17]. This is illustrated in the RHS of Fig. 2 for the forward recursion[1] where two trellis-steps are processed at a time, skipping odd-numbered steps. Specifically, the forward state-metrics $\alpha_k(s)$ are computed on the basis of its *four* admissible predecessor states $s''_0$, $s''_1$, $s''_2$, and $s''_3$ (at step $k-2$) according to

$$\begin{aligned}\alpha_k(s) = \max\big\{&\alpha_{k-2}(s'_0) + \gamma_k(s''_0, s), \alpha_{k-2}(s''_1) + \gamma_k(s''_1, s),\\ &\alpha_{k-2}(s''_2) + \gamma_k(s''_2, s), \alpha_{k-2}(s''_3) + \gamma_k(s''_3, s)\big\}. \quad (3)\end{aligned}$$

The radix-4 branch-metrics required in (3) are computed according to

$$\gamma_k(s''_i, s) = \gamma_{k-1}(s''_i, s'_j) + \gamma_k(s'_j, s) \quad (4)$$

using the six branch-metrics associated with the trellis-steps $k$ and $k-1$ required in the radix-2 recursion (see Fig. 2).

## C. LTE Interleaver

Interleavers scramble data in a pseudo-random order to minimize the correlation of neighboring bits at the input of the convolutional encoders (see Fig. 1). Some (e.g., the one specified in HSDPA [2]) can present a challenge for on-the-fly address-computation and lead to rather complex circuits [12]. LTE, on the other hand, specifies the use of a quadratic polynomial permutation (QPP) interleaver [18], [19] that allows efficient computation of interleaved addresses in hardware. Specifically, address-computation for QPP interleavers is carried out according to

$$\pi(k) = \big(f_1 k + f_2 k^2\big) \mod K \quad (5)$$

where $f_1$ and $f_2$ are suitably chosen interleaver parameters that depend on the code-block length $K$. For $k \in \{0, 1, \dots, K-1\}$, interleaved addresses can be generated by means of the following recursion [20]

$$\begin{aligned}\pi(k+1) &= \big(\pi(k) + \delta(k)\big) \mod K\\ \delta(k+1) &= \big(\delta(k) + b\big) \mod K \quad (6)\end{aligned}$$

where $\pi(0) = 0$, $\delta(0) = f_1 + f_2$, and $b = 2f_2$. The recursion can be implemented efficiently in hardware because only additions and modulo operations are involved. Indeed, as QPP interleavers map even addresses to even addresses and odd to odd [19], they facilitate efficient address-generation for radix-4 recursions; in addition, (6) can also be formulated for other address-increments (e.g., for even-numbered addresses).

## III. Parallel Turbo-Decoder Architecture

The critical path of non-parallel turbo-decoders is usually in the state-metric recursion units, since the associated recursive computations exacerbate pipelining. Corresponding speed-optimized implementations achieve not more than tens of Mb/s, e.g., [12], [21], and hence, to meet the 326.4 Mb/s LTE peak data-rate, the decoding time per half-iteration must therefore be reduced. A promising solution is to instantiate $N$ M-BCJR units and to perform $N$-fold parallel decoding of the trellis [22]. To this end, the trellis is divided into $N$ *trellis segments* of equal[2] length $S$ and parallel SISO decoding is carried out in the assigned trellis-segment in parallel fashion. This approach roughly increases the turbo-decoding throughput by a factor of $N$ compared to (non-parallel) turbo-decoders.[3]

### A. High-Level Architecture

The proposed architecture shown in Fig. 3 is based on the (non-parallel) HSDPA turbo-decoder in [12]. SISO decoding is performed by alternating between non-interleaved and interleaved phases which decode the first and second convolutional code, respectively.

*1) Overview:* The architecture contains $N$ max-log M-BCJR instances, input memories for the storage of the systematic and parity LLRs, and one intermediate memory for the storage of the extrinsic LLRs. The LTE interleaver consists of an address-generator unit for the computation of *all* interleaved and non-interleaved addresses, and of dedicated permutation networks located at the input and intermediate memories. Each M-BCJR instance processes one trellis-step per clock cycle for radix-2 and two steps when using radix-4 recursions. We note that the use of radix-4 recursions entails $2\times$ increased memory-bandwidth, since the LLRs associated with even- and odd-numbered trellis-steps are required per clock cycle.

[1]The radix-4 backward and dummy backward state-metric recursions are carried out in a similar fashion.

[2]$NS = K$ is guaranteed for all code-block lengths specified in LTE with $N \in \{1, 2, 4, 8\}$.

[3]The overhead caused by the dummy forward-recursions and by latencies present in the M-BCJR decoders prevents linear scaling of the throughput in the number of parallel SD instances $N$.

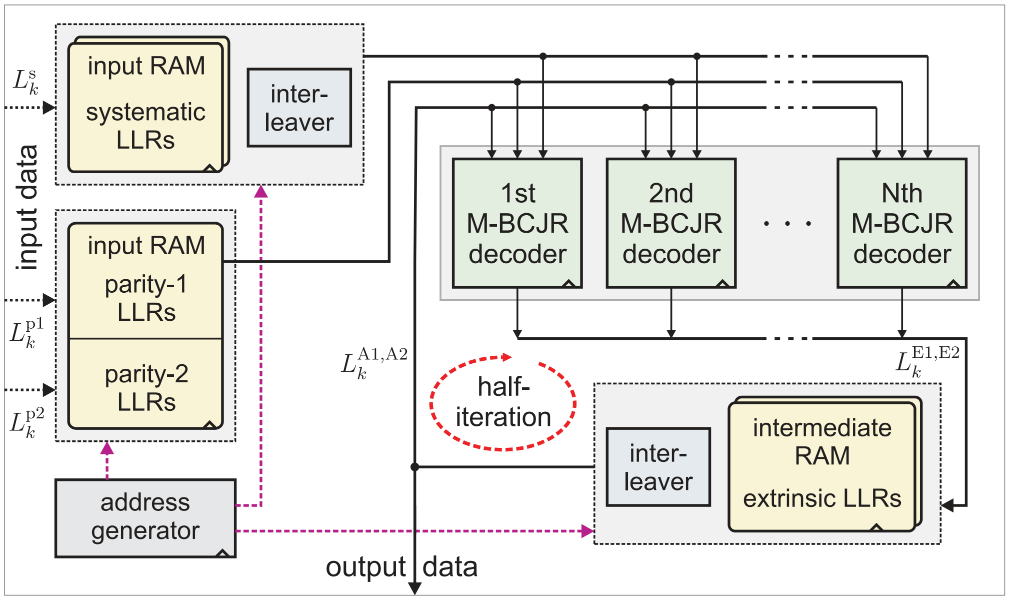


Fig. 3. High-level architecture of the proposed parallel turbo-decoder for LTE.

To cope with this (potentially) high memory-bandwidth, the following memory architecture is used.

*2) Memory architecture:* The instantiated input and intermediate memories store up to 6144 LLRs in support of the maximum LTE code-block length. Each memory contains $N$ LLR-values per address (see Section IV for details). When performing radix-2 computations, the systematic, parity 1-and-2, and extrinsic LLRs are stored in separate RAMs. Since access to either the parity-1 or parity-2 LLRs is required (in the non-interleaved or interleaved phase), one single-port (SP) RAM storing *both* sets of parity LLRs is used to minimize silicon area. The systematic RAM only requires SP-capability, whereas the intermediate memory requires two-port (TP) capability to provide the required memory bandwidth. The $2\times$ higher memory-bandwidth required by radix-4 recursions can be provided by instantiating two RAMs for the systematic LLRs and two for the extrinsic LLRs, all providing half the amount of storage. One RAM instance is then used for the LLRs associated to the even-numbered and the other for the odd-numbered trellis-steps. This partitioning enables the $2\times N$ LLRs to be read per clock cycle. Note that splitting of the parity RAM can be avoided by storing the two sets of LLRs for $k$ and $k-1$ per address.

## *B. Implementation Tradeoffs*

The key design parameters of the parallel turbo-decoder architecture in Fig. 3 are the number of parallel M-BCJR instances $N$ and the use of either radix-2 or radix-4 recursions. In order to determine the most efficient configuration that meets the throughput requirements of LTE, we study the associated throughput/area tradeoffs in Fig. 4 for 0.13µm CMOS technology.[4]

[4]The throughput is given for 5.5 full-iterations and a block-length of $K=3200$. The area and throughput correspond to synthesis results that have been scaled to match with the measured area and throughput of the $8\times$ parallel radix-4 turbo-decoder ASIC presented in Section VI.

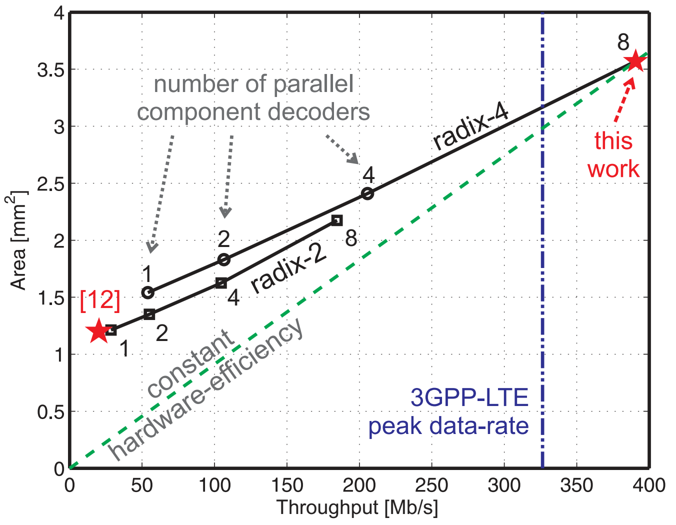


Fig. 4. Throughput/area tradeoffs for parallel radix-2 and radix-4 turbo-decoders for maximum clock frequency in 0.13µm CMOS technology.

Fig. 4 shows that the architectures employing radix-2 computations achieve a throughput ranging from 28 Mb/s ($N=1$) to 180 Mb/s ($N=8$). Note that the HSDPA implementation [12] exhibits similar throughput and area as the $N=1$ radix-2 LTE design. Since $N=8$ is the maximum parallelism supported for decoding of *all* code-block lengths in LTE, we switch from radix-2 to radix-4 to reach higher throughputs. The 326.4 Mb/s LTE peak data-rate (indicated by the horizontal line in Fig. 4) can only be achieved for $N=8$ in combination with radix-4 computations and therefore, we consider the use of this configuration in the remainder of the paper. We emphasize that these parameters additionally lead to best hardware-efficiency (in terms of throughput per area), which is due to the fact that the parallelization improves the throughput almost linearly in $N$ while only increasing the

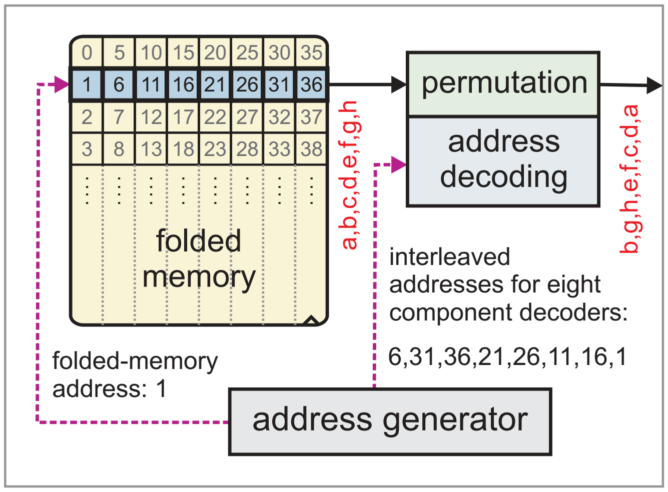


Fig. 5. Architectural principle of the contention-free interleaver for $N = 8$ and $S = 5$. Address-decoding generates the sorting-order that is required to assign the LLRs from the folded memory to the corresponding SISO decoders.

area associated with the M-BCJR instances (and interleaver-related circuitry) but not the area of the rather large (input and intermediate) memories. Hence, increasing the parallelism $N$ reduces the (detrimental) influence of the memories to the overall hardware-efficiency.

We finally note that the general throughput/area tradeoffs remain valid for other technology nodes. However, if a more advanced technology node is used, less parallelism would be necessary or radix-2 recursions might be sufficient to meet the LTE peak data-rate.

## IV. Alleviating the Interleaver Bottleneck

Providing interleaved memory access at the high bandwidth required by eight parallel radix-4 M-BCJR units is a challenging task. For most interleavers, e.g., the one specified in HSDPA [2], parallel and interleaved memory access leads to an *interleaver bottleneck* which is caused by access-contentions, eventually resulting in rather inefficient implementations (see [23] for details). In the ensuing discussion, we propose an architecture solution for LTE that alleviates the interleaver bottleneck arising from parallel turbo-decoding.

### A. Contention-Free Interleaving for LTE

The LTE interleaver exhibits two properties that allow for bandwidth-efficient access to the memories in interleaved and natural order. First, the interleaver is contention-free [24], which ensures that—if $N$ divides all code-block lengths $K$ without remainder, i.e., for $N \in \{1, 2, 4, 8\}$—the LLR-values required by the $N$ SDs can *always* (in interleaved as well as non-interleaved decoding phases) be read out of $N$ different memories. Second, the interleaver is maximally-vectorizable [19], which implies that the address-distance between each of the $N$ interleaved addresses is always an integer multiple of the trellis-segment length $S$. We next describe a general architecture solution that exploits both properties to avoid the interleaver bottleneck.

*1) Memory folding:* As is illustrated in Fig. 5, the sequence of $K$ LLRs associated with one code-block is stored in a *folded-memory*, which consists of $S$ addresses, contains $N$ LLR-values per address, and provides storage for $K = NS$ LLRs.[5] The LLRs are written column-wise, such that the $n$th trellis-segment corresponds to the $n$th column of the memory. In the non-interleaved phase, $N$-fold parallel access to the folded memory is straightforward. Starting from the folded-memory address 0 in incrementing fashion, one reads out all $N$ LLR values residing at the current address and assigns the LLR-value located at the $n$th column to the $n$th M-BCJR instance for $n = 1, \ldots, N$. This access scheme ensures that each M-BCJR instance obtains the LLRs corresponding to its assigned trellis-segment in the right order.

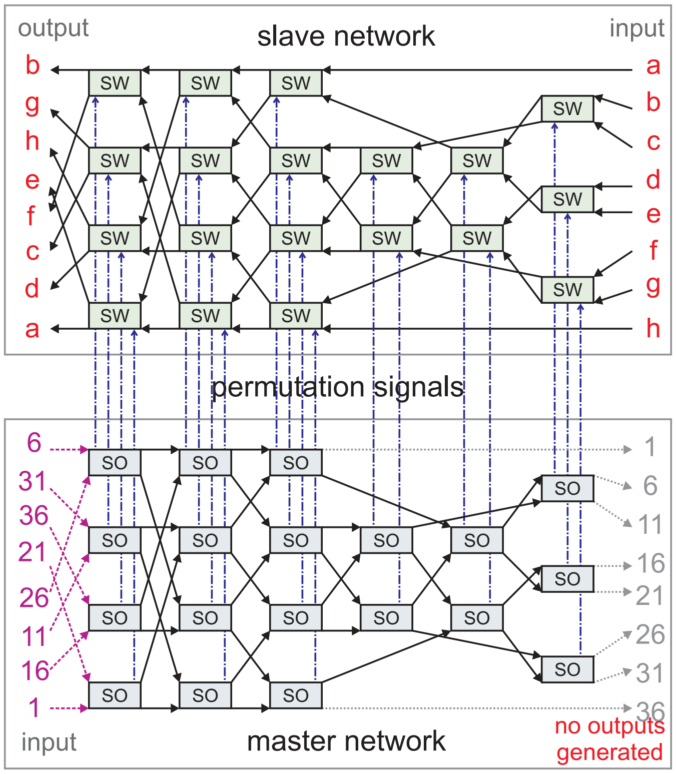
*2) Interleaving:* In the interleaved phase, each M-BCJR instance is associated with an address-generator computing the corresponding interleaved address. Since the LTE interleaver is maximally-vectorizable, the $N$ interleaved addresses *always* point at the same row in the folded memory. The address for the folded memory is immediately given by the smallest address among the $N$ interleaved addresses. The assignment of the $N$ LLR-values to the M-BCJR instances is more involved and is performed in two stages. First, *address-decoding* extracts the sorting-order that is required to assign the $N$ LLRs of the folded memory to the $N$ M-BCJR instances in the right order. Second, a *permutation* according to the extracted sorting-order is applied to the $N$ LLR values, which are then passed to the corresponding M-BCJR instances. This concept enables $N\times$ parallel and interleaved access to the folded memory while avoiding access-contentions.

### B. Master-Slave Batcher Network

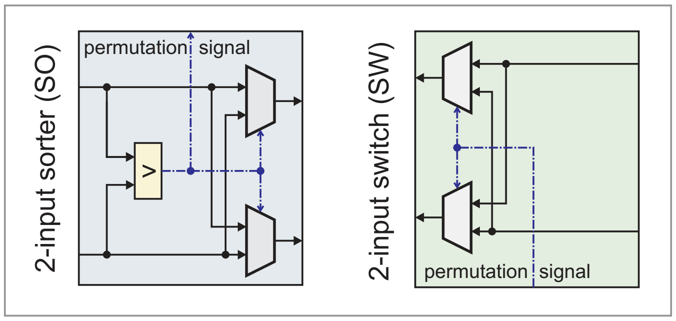
On-the-fly address-generation for the LTE interleaver is—thanks to the recursion (6)—not hardware-critical. However, interleaved memory-access additionally requires i) the permutation signals that enable the network to reverse the interleaved order of the $N$ addresses to its original order to be worked out and ii) the LLRs to be assigned to the corresponding M-BCJR unit. Both tasks are critical from a hardware-perspective, since complex address-decoding logic and signal routing for the distribution of the LLRs to the M-BCJR units are involved. Multiplexers (MUXs), for example, offer an obvious solution for the permutation in hardware, but require rather complex address-decoding circuits and cause large fan-out and wiring overhead for $N > 4$ with $N$ MUXs each having $N$ inputs. In addition, achieving high throughput with pipelining is rather inefficient, because many flip-flops are required to cover all paths. We next propose an efficient solution that can be used not only for the LTE interleaver, but also for *arbitrary* contention-free interleavers that are maximally-vectorizable.

*1) General idea:* The proposed architecture implements the two-step approach outlined in Section IV-A2. The first step performs address-decoding, which amounts to sorting of the $N$ interleaved addresses in ascending order and extracting the

[5]In our radix-4 implementation, even- and odd-numbered systematic and extrinsic LLRs are stored in separate RAMs with $S/2$ addresses (cf. Section III).



(a) MS Batcher network for $N = 8$.



(b) 2-input sorter (SO) and 2-input switch (SW) units.

Fig. 6. Address-decoding and permutation for maximally-vectorizable contention-free interleavers based on the proposed master-slave (MS) Batcher network.

corresponding *sorting-order*. The second step realizes the permutation, which is obtained through sorting of the $N$ LLRs using the *inverse* sorting-order. The inverse sorting-order provides the permutation that generates the sequence of interleaved addresses from the sorted address-sequence. Finally, the $n$th output of the permutation stage is assigned to the $n$th M-BCJR unit for $n = 1, \ldots, N$.

*2) Architecture:* Our architecture, which is based on two Batcher sorting networks [25], is shown in Fig. 6. Address-decoding is carried out in the *master* network consisting of a small number of 2-input sorter (SO) units to perform sorting of the interleaved addresses in ascending order. As depicted in Fig. 6(b), each 2-input sorter generates a permutation signal indicating whether the inputs have been swapped or not. The *slave* network performs the permutation by applying the inverse sorting-order to the $N$ LLRs (see Fig. 6(a)). To this end, we use a second network having the *same* interconnections as the master network and consisting of 2-input switches (SW) instead of sorters. The permutation signals generated in the master network control the switches in the slave network. In order for the slave network to implement the *inverse sorting-order*, the LLRs are fed into the slave network from the opposite direction.[6]

*3) Properties:* The proposed master-slave (MS) Batcher network offers an area-efficient way to perform address-decoding *and* permutation in hardware and accommodates pipelining with little hardware resources, which is essential to achieving high-throughput interleaving. In addition, the architectural concept enables the use of asymptotically optimal sorting networks for an arbitrary number of inputs $N$.[7] Consequently, our solution allows for the economic implementation of interleavers for highly-parallel turbo-decoders. Note that another efficient permutation network, applicable to QPP interleavers only, has been proposed in [7].

## V. Radix-4 Max-Log M-BCJR Architecture

Pipelining applied to the MS Batcher network removes the critical path from the interleaver. Hence, the maximum clock frequency of the turbo-decoder will be determined by the critical path of the M-BCJR architecture. In addition, the eight M-BCJR instances will carry out almost all computations and hence, dominate the circuit area and power consumption of the whole turbo-decoder. Therefore, to meet the LTE peak data-rate while remaining area- and power-efficient, optimization of the M-BCJR units is of paramount importance.

### A. VLSI Architecture

The radix-2 M-BCJR unit of [12] forms the basis of the radix-4 architecture depicted in Fig. 7. We implemented radix-4 computations and applied further optimizations to improve throughput and hardware-efficiency. The computation of two trellis-steps is performed per clock cycle using three parallel state-metric recursion units. These units implement the forward, backward, and dummy backward state-metric recursions, and contain one radix-4 ACS circuit for each of the eight trellis-states. This processing scheme allows the radix-4 M-BCJR unit to compute two LLR values (associated with an even- and odd-numbered trellis-step) per clock cycle, so that each $M = 30$ trellis-step window is computed sequentially in 15 clock cycles.

In order to reduce storage requirements, the branch-metric preprocessing unit computes $L_k^{\mathrm{A}} + L_k^{\mathrm{s}}$ (for $k$ and $k-1$) and stores this intermediate result together with the corresponding $L_k^{\mathrm{p1,p2}}$ in one of the three $\gamma$-memories [12]. The $\gamma$-memories are realized by arrays of latches (requiring approximately the same area compared to SRAM macrocells) to simplify placing of the eight M-BCJR units during the back-end design. The radix-4 branch-metric computation units first

[6]In [1], the slave Batcher Network was implemented incorrectly, so that only a subset of code-block lengths specified in LTE can be decoded.

[7]The number of required sorters scales with $O\big(N\log(N)\big)$ and the depth of such networks (i.e., the number of sorters in the critical path) is $O\big(\log(N)\big)$, e.g., [26].

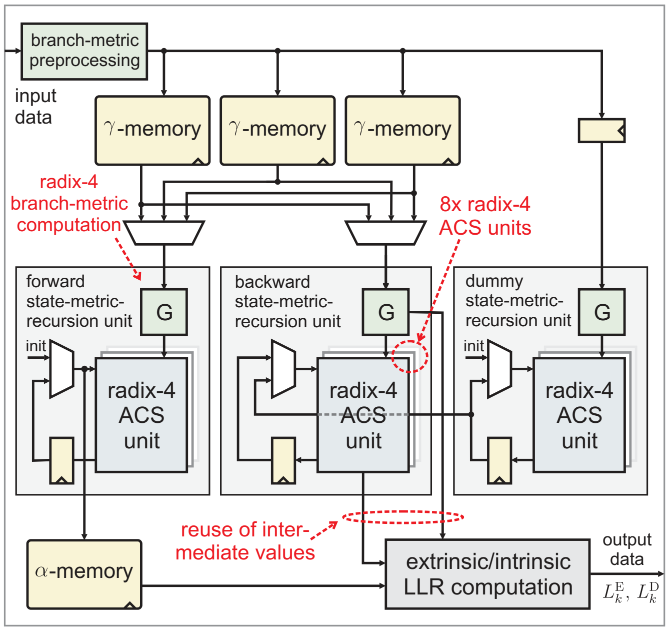

branch-metric preprocessing
input data
γ-memory
γ-memory
γ-memory
radix-4 branch-metric computation
8x radix-4 ACS units
forward state-metric-recursion unit
backward state-metric-recursion unit
dummy state-metric-recursion unit
G
G
G
init
init
radix-4 ACS unit
radix-4 ACS unit
radix-4 ACS unit
reuse of inter-mediate values
α-memory
extrinsic/intrinsic LLR computation
output data
$L_k^E$, $L_k^D$

Fig. 7. Architecture of the implemented radix-4 max-log M-BCJR core.

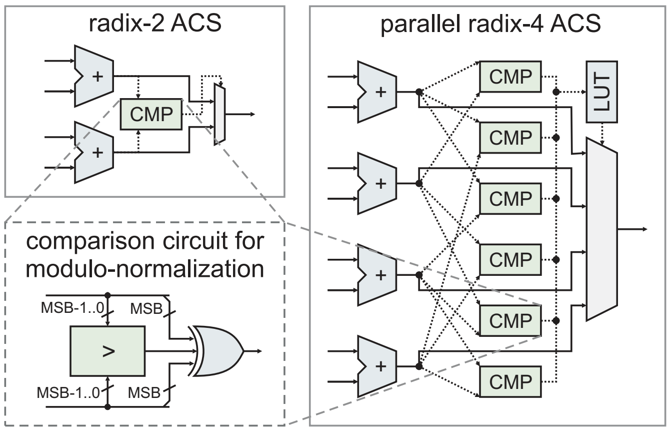


Fig. 8. Radix-2 (left) and radix-4 (right) ACS architectures based on modulo-normalization.

work out the radix-2 branch-metrics and then calculate the radix-4 branch metrics (4). The forward state-metric recursion unit starts with a dummy recursion for one window to obtain initial forward state-metrics and then, computes the $\alpha_k(s)$ for each window, which are cached within the $\alpha$-memory (realized using latches only). The backward state-metric recursion unit computes the $\beta_k(s)$ for each window. The initial backward state-metrics required at the beginning of each window are generated by the dummy state-metric recursion unit. The cached forward state-metrics, the intermediate results obtained in the backward recursion, and the radix-2 branch-metrics are finally used to compute the intrinsic LLRs (2).

### B. Radix-4 ACS Units with Modulo-Normalization

To arrive at a short critical path in the ACS units, (expensive) re-normalization circuitry is avoided by employing *modulo-normalization* [27], [28], which only requires minor modifications of the comparison circuit involved [12]. Fig. 8 shows corresponding radix-2 and radix-4 ACS architectures. The critical path of the presented (parallel) radix-4 ACS unit is optimized for throughput. Here, the selection signal is carried out by six parallel comparators followed by a look-up table (LUT). This architecture shortens the critical path-delay by about 50% compared to a tree-like radix-4 ACS implementation. We finally note that radix-4 recursions require one bit more for the state-metrics (as for radix-2) to ensure proper functioning of modulo-normalization, which is due to the fact that the radix-4 branch-metrics (4) have $2\times$ larger dynamic range.

Our implementation results have shown that radix-4 M-BCJR units achieve approximately 40% higher throughput (at $0.7\times$ lower clock frequency) while being only 15% less hardware-efficient (in terms of Mb/s per area) compared to radix-2-based designs. We therefore conclude the use of radix-4 is essential for turbo-decoders aiming at maximum throughput and considerably relax the corresponding design constraints due to the reduced clock frequency.

### C. LLR Computation Unit

The LLR computation unit shown in Fig. 7 computes the intrinsic and extrinsic LLRs for the trellis-steps $k-1$ and $k$ in each clock cycle. Since radix-4 computations only obtain the state- and branch-metrics associated with even trellis-steps $k=2k'$, computation of the (intermediate) state- and branch-metrics associated with the *odd* trellis-steps $k=2k'+1$ is required. These values can be derived from the state-metrics $\alpha_{k-2}(s)$ stored in the $\alpha$-memory, from the intermediate results of the backward state-metric recursion $\gamma_k(s_i'',s)+\beta_k(s)$, and from the (radix-2) branch metrics $\gamma_k(s_i',s)$ and $\gamma_{k-1}(s_j'',s_i')$. LLR computation first works out the forward $\alpha_{k-1}(s)$ and backward $\beta_{k-1}(s)$ state-metrics associated with the odd trellis-step $k-1$ and then—with the aid of the radix-2 branch-metrics—calculates the intrinsic LLRs (2). Computation of (2) is realized using two maximization trees, which have been pipelined to ensure that the critical path of the M-BCJR block is in one of the three state-metric recursion units.

## VI. Implementation Results and Comparison

In this section, we summarize the key characteristics of the $8\times$ parallel LTE turbo-decoder designed in this paper and compare our ASIC implementation results to that of other turbo-decoder implementations.

### A. Error-Rate Performance

To achieve near-optimal error-rate performance, the input LLRs are quantized to 5 bit, the extrinsic LLRs to 6 bit, all state-metrics in the radix-4 ACS units require 10 bit, and extrinsic scaling with a hardware-friendly constant of 0.6875 was used. Fig. 9 shows the resulting bit error-rate (BER) of the hardware implementation and provides a comparison to the ideal turbo-decoding algorithm (i.e., employing floating-point arithmetics and using the BCJR algorithm) at 5.5 full-iterations. One can observe that the use of extrinsic scaling leads to a 0.2 dB improvement compared to the standard max-log M-BCJR algorithm and the final implementation loss is

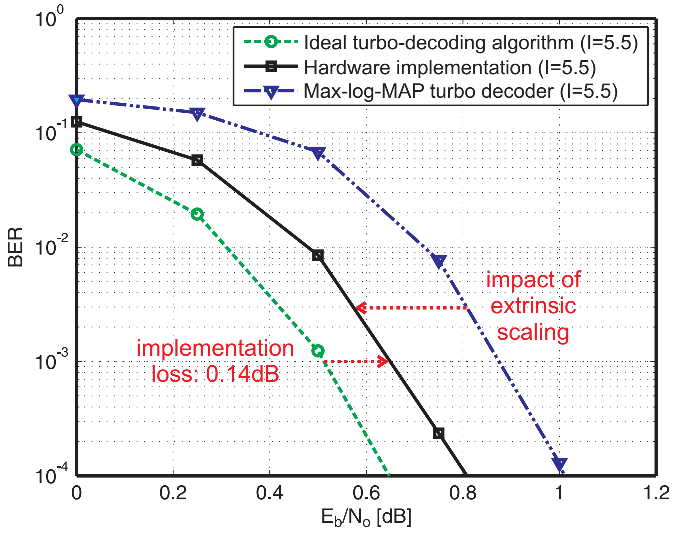


Fig. 9. BER performance in AWGN channel for code-block length 3200.

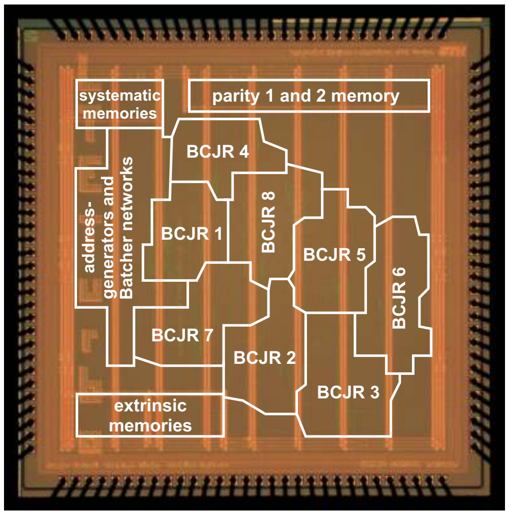


Fig. 10. Turbo-decoder ASIC micrograph with highlighted units.

smaller than 0.14 dB (in terms of $E_b/N_0$) compared to that achieved by the ideal turbo-decoding algorithm.

## B. Key Characteristics and Comparison

Table I summarizes the key characteristics of the implemented turbo-decoder prototype [1] in 0.13 µm (1P/8M) CMOS technology. The corresponding ASIC micrograph is shown in Fig. 10. About 2/3 of the area is occupied by the eight radix-4 max-log M-BCJR instances, which confirms that optimization of the M-BCJR unit is of paramount importance during the design of a parallel turbo-decoder.

Normalized hardware efficiency (in Mb/s/mm$^2$) of our ASIC prototype is best in class. The active area of the chip is 3.57 mm$^2$, comprises 553 kGE (excluding the memories), and uses 129 kb of RAM. The maximum measured clock frequency is 302 MHz, at which a throughput of 390.6 Mb/s at 5.5 iterations has been measured. Our ASIC prototype is therefore the first reported in the open literature that achieves the theoretical 326.4 Mb/s LTE peak data-rate (with a 20% safety-margin). The power consumption (measured at maximum throughput) is 788.9 mW, leading to an energy-efficiency of 0.37 nJ/bit/iteration.[8] Note that only [5] reports a slightly better energy-efficiency at less than half the throughput and about 5× larger silicon area.

TABLE II
POWER CONSUMPTION WITH SUPPLY-VOLTAGE SCALING

| Throughput [Mb/s] | Supply voltage [$V_{dd}$] | Leakage current | Power consumption | Energy eff. [nJ/bit/iter.] |
|---|---|---|---|---|
| 326.4 | 1.06 V | 0.98 mA | 503 mW | 0.28 |
| 100.0 | 0.71 V | 0.56 mA | 68.6 mW | 0.13 |

Table II shows the power consumption required for the 326.4 Mb/s LTE peak data-rate and the more realistic 100 Mb/s throughput targeted by industry today. The supply voltage was scaled to meet the specified throughputs, leading to 503 mW for the LTE peak data-rate and only 68.6 mW for the 100 Mb/s milestone. A comparison of the 0.13 nJ/bit/iteration energy-efficiency with other turbo-decoders in Table I highlights the effectiveness of our implementation concept.

## VII. Conclusions

In this paper, we detailed the design of a parallel turbo-decoder for the 3GPP-LTE standard. The analysis of the throughput/area tradeoffs associated with parallel turbo-decoders have shown that radix-4 in combination with eight M-BCJR instances are necessary for the LTE peak data-rate in 0.13 µm CMOS technology. Parallel and interleaved access to the memories at high throughput was achieved through the development of a master-slave Batcher network. Optimizations in the radix-4 M-BCJR unit finally led to a high-performance and low-area turbo-decoder architecture. In addition to setting a record in turbo-decoding throughput, both ultra low-power and cost-effectiveness have been demonstrated by the presented implementation concept.

## References

[1] C. Studer, C. Benkeser, S. Belfanti, and Q. Huang, "A 390Mb/s 3.57mm$^2$ 3GPP-LTE turbo decoder ASIC in 0.13$\mu$m CMOS," in *IEEE ISSCC dig. tech. papers*, vol. 1, San Francisco, CA, USA, Feb. 2010, pp. 274–275.
[2] *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Channel Coding and Multiplexing Examples*, 3GPP Organizational Partners TS 25.944, Rev. 4.1.0, Jun. 2001.
[3] *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (Release 9)*, 3GPP Organizational Partners TS 36.212, Rev. 8.3.0, May 2008.
[4] 3GPP TR25.912 V8.0.0, "Feasibility study for evolved universal terrestrial radio access (UTRA) and universal terrestrial radio access network (UTRAN)," Feb. 2009.
[5] C.-C. Wong, M.-W. Lai, C.-C. Lin, H.-C. Chang, and C.-Y. Lee, "Turbo decoder using contention-free interleaver and parallel architecture," *IEEE JSSC*, vol. 45, no. 2, pp. 422–432, Feb. 2010.
[6] J.-H. Kim and I.-C. Park, "A unified parallel radix-4 turbo decoder for mobile WiMAX and 3GPP-LTE," in *Proc. CICC*, San Jose, CA, USA, Sept. 2009, pp. 487–490.

[8]All power measurements are performed at $T = 300$ K for a code-block length $K = 3200$ and 5.5 full-iterations using typical stimuli generated at $E_b/N_0 = 0.6$ dB.

TABLE I
KEY CHARACTERISTICS AND COMPARISON TO OTHER MEASURED TURBO-DECODER ASICS

| Publication | This work | Wong *et al.* [5] | Kim and Park [6] | Wong *et al.* [7] | Benkeser *et al.* [12] | Bickerstaff *et al.* [21] |
|---|---|---|---|---|---|---|
| Standard | LTE | — | LTE / WiMAX | LTE | HSDPA | HSDPA |
| Radix/parallel decoders | 4/8 | 2/16 | 4/8 | 2/8 | 2/1 | 4/1 |
| CMOS technology [nm] | 130 | 130 | 130 | 90 | 130 | 180 |
| Supply voltage [V] | 1.2 | 1.32 | 1.2 | 1.0 | 1.2 | 1.8 |
| Gate count | 553 k | $2.67^a$ M | $800^a$ k | — | 44.1 k | 410 k |
| Memory [kb] | 129 | — | — | — | 122 | 410 |
| Active area [mm$^2$] | 3.57 | 17.81 | 10.7 | 2.1 ($4.3^b$) | 1.2 | 14.5 ($7.3^b$) |
| Maximum throughput at 5.5 iterations [Mb/s] | 390.6 | $233^c$ | $271^c$ | $188^c$ ($133^{bc}$) | 20.3 | $26.2^c$ ($37^{bc}$) |
| Normalized hardware-efficiency$^b$ [Mb/s/mm$^2$] | 109.4 | 13.1 | 25.3 | 30.9 | 16.9 | 5.07 |
| Leakage current [mA] | $1.23^d$ | — | — | — | 1.73 | — |
| Power consumption in [mW] at [Mb/s] | $788.9^d$ at 390.6 | 275 at 160 | — | 219 ($619^b$) at 129 | 57.8 at 10.8 | 956 ($338^b$) at 10.8 |
| Normalized energy efficiency$^{bc}$ [nJ/bit/iter.] | $0.37^d$ | 0.22 | 0.61 | 0.59 | 0.7 | 3.9 |

$^a$Including the gate count of the memories.
$^b$Technology scaling to 0.13 μm CMOS assuming: $A \sim 1/s^2$, $t_{\text{pd}} \sim 1/s$, and $P_{\text{dyn}} \sim 1/s^3$.
$^c$Throughput linearly scaled to 5.5 iterations.
$^d$Measured at $V_{\text{dd}} = 1.2\,\text{V}$ and $T = 300\,\text{K}$.

[7] C.-C. Wong and H.-C. C. Y.-Yu Lee, "A 188-size 2.1mm$^2$ reconfigurable turbo decoder chip with parallel architecture for 3GPP LTE system," in *Symp. VLSI circuits dig. tech. papers*, Kyoto, Japan, June 2009, pp. 288–289.

[8] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Comm.*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.

[9] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Th.*, vol. 20, no. 2, pp. 284–287, Mar. 1974.

[10] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Th.*, vol. 13, no. 2, pp. 260–269, Apr. 1967.

[11] J. P. Woodard and L. Hanzo, "Comparative study of turbo decoding techniques: an overview," *IEEE Trans. Vehicular Tech.*, vol. 49, no. 6, pp. 2208–2233, Nov. 2000.

[12] C. Benkeser, A. Burg, T. Cupaiuolo, and Q. Huang, "Design and optimization of an HSDPA turbo decoder ASIC," *IEEE JSSC*, vol. 44, no. 1, pp. 98–106, Jan. 2008.

[13] V. Franz and J. B. Anderson, "Concatenated decoding with a reduced-search BCJR algorithm," *IEEE J. Sel. Areas in Comm.*, vol. 16, no. 2, pp. 186–195, Feb. 1998.

[14] M. van Dijk, A. J. E. M. Janssen, and A. G. C. Koppelaar, "Correcting systematic mismatches in computed log-likelihood ratios," *Europ. Trans. Telecomm.*, vol. 14, no. 3, pp. 227–244, Jul. 2003.

[15] J. Vogt and A. Finger, "Improving the max-log-MAP turbo decoder," *Elec. Letters*, vol. 36, no. 23, pp. 1937–1939, Nov. 2000.

[16] M. May, T. Ilnseher, N. Wehn, and W. Raab, "A 150MBit/s 3GPP LTE turbo code decoder," in *Proc. DATE*, Dresden, Germany, Mar. 2010.

[17] G. Fettweiss and H. Meyr, "Parallel Viterbi algorithm implementation: breaking the ACS-bottleneck," *IEEE Trans. Comm.*, vol. 37, no. 8, pp. 785–790, Aug. 1989.

[18] J. Sun and O. Y. Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," *IEEE Trans. Inf. Th.*, vol. 51, no. 1, pp. 101–119, Jan. 2005.

[19] A. Nimbalker, Y. Blankenship, B. Classon, and T. K. Blankenship, "ARP and QPP interleavers for LTE turbo coding," in *Proc. IEEE WCNC*, Las Vegas, NV, USA, Mar. 2008, pp. 1032–1037.

[20] B. Moision and J. Hamkins, "Coded modulation for the deep-space optical channel: Serially concatenated pulse-position modulation," *IPN Progess Report 41-161*, pp. 1–25, May 2005.

[21] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, "A 24Mb/s radix-4 logMAP turbo decoder for 3GPP-HSDPA mobile wireless," in *IEEE ISSCC dig. tech. papers*, vol. 1, San Francisco, CA, USA, Feb. 2003, pp. 150–484.

[22] R. Dobkin, M. Peleg, and R. Ginosar, "Parallel VLSI architecture for MAP turbo decoder," in *Proc. IEEE Int. Symp. PIMRC*, vol. 1, Sept. 2002, pp. 384–388.

[23] C. Benkeser, "Power efficiency and the mapping of communication algorithms into VLSI," Ph.D. dissertation, ETH Zürich, Switzerland, Series in Microelectronics, vol. 209, Hartung-Gorre Verlag Konstanz, 2010.

[24] O. Y. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," *IEEE Trans. Inf. Th.*, vol. 52, no. 3, pp. 1249–1253, Mar. 2006.

[25] K. E. Batcher, "Sorting networks and their applications," in *Proc. of 32th AFIPS Spring Joint Computer Conf.*, Atlantic City, NJ, USA, 1968, pp. 307–314.

[26] M. Ajtai, J. Komlós, and E. Szemerédi, "An $O(n \log n)$ sorting network," *Ann. ACS Symp. Theory of Comp.*, pp. 1–9, 1983.

[27] A. P. Hekstra, "An alternative to metric rescaling in Viterbi decoders," *IEEE Trans. Comm.*, vol. 37, no. 11, pp. 1220–1222, Nov. 1989.

[28] C. B. Shung, P. H. Siegel, G. Ungerboeck, and H. K. Thapar, "VLSI architectures for metric normalization in the Viterbi algorithm," in *Proc. IEEE ICC*, vol. 4, Atlanta, GA, USA, Apr. 1990, pp. 1723–1728.