FULLY PIPELINED ITERATION UNROLLED DECODERS - THE ROAD TO TB/S TURBO DECODING

Stefan Weithoffer[⋆] *Rami Klaimi*[⋆] *Charbel Abdel Nour*[⋆] *Norbert Wehn*[†] *Catherine Douillard*[⋆]

[⋆] IMT Atlantique, Department of Electronics, Lab-STICC - UMR 6285
[†] Department of Electrical and Computer Engineering, Technical University Kaiserslautern

ABSTRACT

Turbo codes are a well-known code class used for example in the LTE mobile communications standard. They provide built-in rate flexibility and a low-complexity and fast encoding. However, the serial nature of their decoding algorithm makes high-throughput hardware implementations difficult. In this paper, we present recent findings on the implementation of ultra-high throughput Turbo decoders. We illustrate how functional parallelization at the iteration level can achieve a throughput of several hundred Gb/s in 28 nm technology. Our results show that, by spatially parallelizing the half-iteration stages of fully pipelined iteration unrolled decoders into X-windows of size 32, an area reduction of 40% can be achieved. We further evaluate the area savings through further reduction of the X-window size. Lastly, we show how the area complexity and the throughput of the fully pipelined iteration unrolled architecture scale to larger frame sizes. We consider the same target bit error rate performance for all frame sizes and highlight the direct correlation to area consumption.

***Index Terms*—** Beyond 5G, Turbo Decoding, High-Throughput, Hardware Architecture

1. INTRODUCTION

Wireless communication systems are a driving force of connecting our world. Their evolution enables technologies like the *Tactile Internet* [1] and the *Internet of Things* (IoT) [2] but comes with ever increasing demands for higher throughputs, higher spectral efficiencies, lower latencies, a lower power consumption and a larger scalability. First and second generation wireless communication systems required a throughput of less than 1 Mb/s, UMTS already supported a throughput of 2 Mbit/s and LTE-A up to Gb/s [3]. For the 5G standard, data rates greater than 10 Gb/s will be targeted and future use cases, *Beyond 5G* (5G+), are expected to have even larger throughput requirements. The Horizon 2020 project *"Enabling Practical Wireless Tb/s Communications with Next Generation Channel Coding"* (EPIC) aims at throughputs well beyond 100 Gb/s, towards Tb/s for *Forward Error Correction* (FEC) utilizing soft informations [4]. In particular, EPIC considers three widely used code families: *Low-Density Parity-Check* (LDPC) codes, *Polar* codes and *Turbo codes*.

The above-mentioned throughput demands directly translate into constraints at the level of the FEC, a mandatory building block for reliable wireless transmissions. In the past, advancements in silicon technologies and in decoder hardware architectures made it possible these increased requirements. For example, in 2016, state-of-the-art Turbo decoder implementations allowed a throughput of approximately 15 Gb/s at 100 MHz in 65 nm silicon technology [5]. However, when scaling to advanced technology nodes, the frequency for hardware implementations of baseband signal processing is constrained due to power and other design issues and, overall, only a maximum frequency of 1 GHz can be achieved. With limited frequency scaling, the throughput has to be scaled by employing extreme levels of parallelism and using low complexity algorithms. However, an extreme level of parallelism in the decoding of FEC codes is often linked to a decrease in the achieved level of *Bit Error Rate* (BER) performance:

- Decoding for a fixed number of iterations with a fully parallel flooding schedule for *Belief Propagation* (BP) decoding of LDPC codes has a lower BER performance than a partially parallel layered decoding schedule [6],
- Adopting the low complexity *Successive Cancellation* (SC) decoding for Polar codes leads to a significantly reduced BER performance when compared to approaches employing list decoding [7],
- Splitting the trellis of the component codes of Turbo codes into smaller sub-trellises to process them in parallel leads to a BER performance drop that must be mitigated through additional calculations limiting the achievable degree of parallelism [8].

In addition, these three widely used code families face different challenges for the implementation of high-throughput decoders. Indeed, LDPC codes are classically decoded using the inherently parallel BP algorithm where complexity is dominated by iterative message exchange. This results in a high degree of routing congestion leading to an area overhead for hardware implementation [9]. Polar decoding is performed via exploration of the code tree structure using the SC algorithm. The latter can support multi-bit processing where complexity is balanced between required computations and message exchanges [10, 11]. However in order to achieve competitive BER performance, list decoding needs to be applied, introducing additional memory management and control overhead which significantly increases implementation complexity. Finally, *Maximum a Posteriori* (MAP) decoding used for Turbo codes is inherently serial and the corresponding complexity is dominated by computations and suffers from data dependencies in the state metric recursion, impacting the achievable level of parallelism. In addition, iterative processing required for decoding LDPC and Turbo codes negatively impacts achievable throughput. Bridging the gap between the performance metrics of current state-of-the-art decoders and the requirements identified by the EPIC project can be achieved by fully unrolling the (iterative) decoding onto a single pipeline [9, 11–13].

[⋆][†] This work was partially funded by the EPIC project of the EU's Horizon 2020 research and innovation programme under grant agreement No. 760150

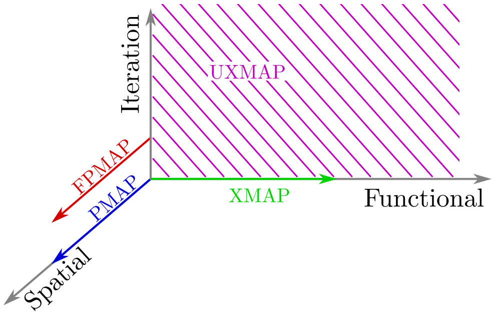

Fig. 1. Dominant types of parallelism for different turbo decoder archetypes.

This paper highlights recent results obtained under the umbrella of the EPIC project with a focus on Turbo decoder implementation. The remainder of this paper is structured as follows: First, Section 2 recapitulates the concept of iteration unrolling in the context of Turbo decoding. Then, Section 3 describes new results on fully pipelined iteration unrolled decoding before Section 4 concludes the paper.

2. THE STEP TO 100 GB/S VIA ITERATION UNROLLING

A Turbo decoder consists of two component decoders connected through an interleaver and a de-interleaver. It applies an iterative loop, exchanging extrinsic information Λ^e between its two components, cooperatively improving the decoding result [14]. State-of-the-art hardware architectures for turbo decoders decoding generally devise one hardware instance alternatingly acting as component decoder 1 and component decoder 2. Moreover, they split the code blocks into smaller *sub-blocks* and employ *spatial* and *functional* parallelization to increase the throughput. Hardware architecture archetypes can be categorized as follows according to the dominant type of parallelization at the decoder level:

- **Parallel MAP (PMAP):** PMAP decoders spatially parallelize the decoding of different parts of the code trellis on multiple sub-decoder cores [15, 16]. However, for smaller sub-blocks and at high code rates, mitigation measures for avoiding BER performance loss are necessary and limit the maximum degree of parallelization [8].
- **Pipelined MAP (XMAP):** The XMAP decoder, named for its X-shaped pipeline structure, uses a functional parallelization approach where the state metric recursions of the MAP algorithm are pipelined [17–20]. It suffers from the same limitations as the PMAP architecture with respect to parallelization. Thus, state-of-the-art implementations of PMAP decoders achieve a throughput of 1-2 Gb/s [21–23], and similarly 1-2 Gb/s have been demonstrated for XMAP decoders [20, 24] in current technologies.
- **Fully Parallel MAP (FPMAP):** This decoder architecture is the extreme case of the PMAP with a sub-block size reduced to 1 trellis stage in combination with a shuffled decoding schedule [6, 25]. It has been shown to achieve a throughput of 15 Gb/s, an order of magnitude more than previously published PMAP implementations [5], but suffers from a reduced BER performance for high code rates [13].

In order to enable a throughput beyond 100 Gb/s for Turbo decoder architectures, spatial or functional parallelization at the decoder level alone will not be enough. However, Turbo codes allow the utilization of functional level parallelism at the iteration level. Pipelining the half-iterations and connecting them to a single pipeline leads to a fourth architecture archetype:

- **Fully Pipelined Iteration Unrolled (UXMAP):** In this decoder architecture, complete frames are processed in parallel while traversing through the decoder pipeline [13, 26]. This allows for a very high throughput which is determined by the frame size and the achievable clock frequency, since one complete decoded frame is output per clock cycle, once the pipeline is completely filled.

Figure 1 illustrates the different decoder archetypes with respect to their position in the design space w.r.t. the dominant type of parallelization. The PMAP and XMAP decoder architectures lie on the spatial and functional parallism axes, while the FPMAP architecture lies on a straight line parallel to the spatial parallelism axis, due to the shuffled decoding schedule which can be seen as an iteration parallelism of 2. The UXMAP, as presented in [13], lies in a plane spanned by the functional and iteration parallelism axes. This leads to a large area consumption for hardware implementations. In a previous work [13], the first turbo decoder achieving 100 Gb/s occupied almost 24 mm^2 for a frame size of $K = 128$.

Therefore, in a recent work, we propose to move away from the ”UXMAP-plane” that is shown in Figure 1 and use an approach that combines all three methods of parallelization [27].

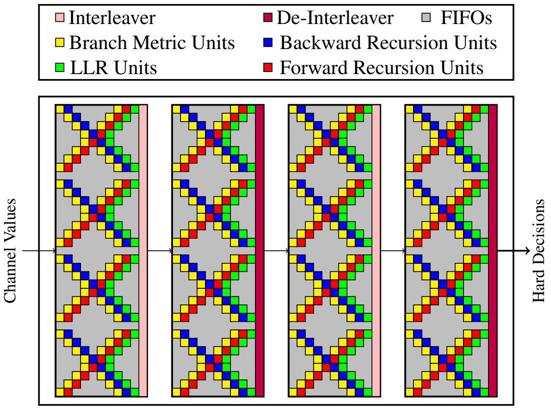

Fig. 2. Architecture schematic of the spatially parallelized UXMAP.

3. ADVANCED ITERATION UNROLLED TURBO DECODING

Combining the UXMAP architecture with spatial parellization by splitting the half-iteration pipelines into smaller *X-Windows* of size 32 leads to a reduction of the area consumption by up to 40% for the same frame size $K = 128$. Moreover, it enables frame size flexibility to support $K = 128, 64, 32$ [27].

Figure 2 illustrates the proposed spatially parallelized UXMAP architecture. It is composed of fully pipelined *Half-Iteration Stages* (HI-Stages) consisting of several X-windows. Note, that in the following, *X_window_size* refers to the size of the sub-trellis which is fed to each X-window. Therefore, the amount of computational units per X-window is divided by $\log_2(4) = 2$ for radix-4. These are then

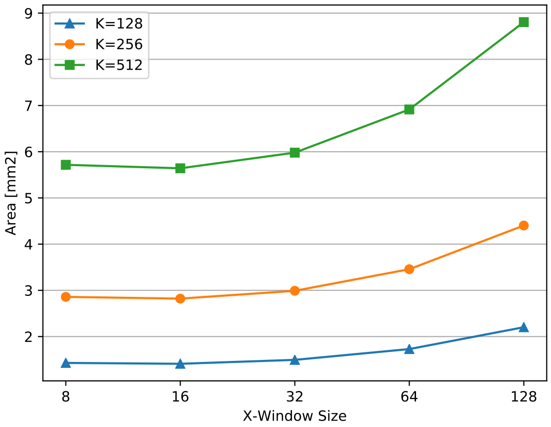

Fig. 3. Area consumption after synthesis depending on the frame size K and the X-window size.

composed of 2·X_window_size/2 radix-4 recursion units (for the forward and backward state metric recursions), 2·X_window_size/2 − 2 radix-4 branch metric units (due to recomputation of the branch metrics) and X_window_size/2 radix-4 LLR-units (for soft output computation). In contrast to the architecture from [13], the FIFO containing the channel values is included directly into the X-windows, allowing for a more localized routing.

By reducing the X-window size, all pipelines for the channel values, for the forward and backward state metrics as well as for the extrinsic values are shortened. This reduces the pipeline latency but also makes it possible to realize decoders with larger frame sizes.

3.1. Spatially Parallelizing the half-iterations of the UXMAP

Figure 3 shows new synthesis results for one half-iteration stage of UXMAP decoders with different frame and X-window sizes in 28 nm FDSOI technology, targeting a frequency of 800 MHz. The channel value quantization is set to 6 bits. For larger X-window sizes, the pipeline stages contribute the most to the overall area consumption. Indeed when reducing X_window_size from 128 to 64, a sharp drop in area consumption is observed. While a noticeable drop is still observed when moving to an X_window_size of 32, the area savings for sizes 16 and 8 are significantly less pronounced. In fact, similar to PMAP or XMAP decoders, the splitting of the code trellis requires mitigation measures in order to avoid a decrease in BER performance. Besides, to avoid an increase in pipeline latency, *Next Iteration Initialization* (NII) [28] is used instead of performing *Acquisition* (ACQ) computations [20], since these would have to be integrated into the overall pipeline. This however requires additional pipelines for forwarding the NII values for the state metrics to the next iteration for each X-window, which counteracts the area savings through overall shorter pipelines. In addition, reducing X_window_size below 32 comes at a non-negligible penalty in BER performance, especially for higher code rates.

Note that the results from Figure 3 do not include placement & routing overhead and a full UXMAP pipeline must be composed of several half-iteration stages. Still, the area saving of 40 % motivates investigation of larger frame sizes.

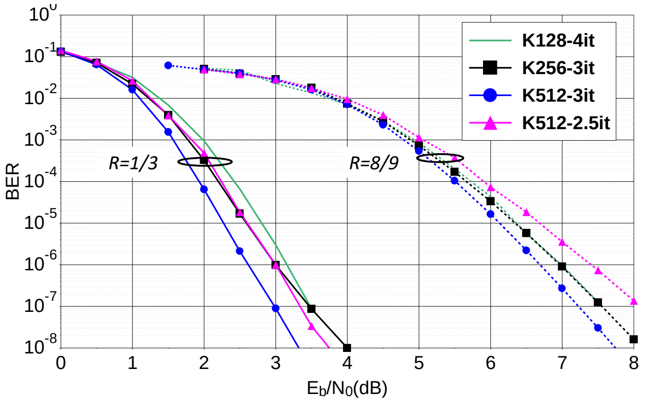

Fig. 4. BER performance for different frame sizes K with an X-window size of 32.

3.2. Increasing the Frame Size

Figure 4 shows BER simulation results for frame sizes of $K = 128$, 256, and 512 bits for a X-window size of 32. The respective *Almost Regular Permutation* (ARP) interleaver parameters were obtained through the methods described in [29, 30] and are listed in Tables 1 and 2. To make the given results comparable to previous work, the decoder with frame size $K = 128$ and 4 full iterations (128,4) applying a X-window size of 32 serves as a reference.

K	P	Q	**S**
128	49	4	[3, 113, 111, 93]
256	79	16	[8, 16, 39, 170, 74, 87, 122, 26, 168, 165, 24, 88, 245, 216, 232, 192]
512	61	16	[8, 50, 107, 192, 258, 289, 454, 360, 376, 7, 316, 494, 173, 434, 292, 398]

Table 1. ARP interleaver parameters.

Punct. Pattern Sys.	[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
Punct. Pattern P_1	[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
Punct. Pattern P_2	[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

Table 2. Puncturing patterns for rate $8/9$.

For rate $1/3$, the BER performance of the reference decoder is met by the decoder with $K = 256$ bits at 3 decoding iterations, while for $K = 512$ bits, 2.5 iterations give the same performance. Decoding with the $(512, 3)$ configuration improves performance by 0.5 dB at a BER of 10^{-6}.

Similarly, for rate $8/9$, decoding $K = 256$ bits with 3 iterations gives identical performance as decoding with $(128, 3)$. For frame size $K = 512$ bits, 3 decoding iterations improve on the reference result by about 0.3 dB at a BER 10^{-6}, while decoding only with 2.5 iterations comes at a penalty of approximately 0.4 dB for the same BER of 10^{-6}.

3.3. Adding a Short Acquisition

As noted above, the splitting of the trellis into X-Windows impacts the BER performance. This effect is more pronounced for larger frame sizes since more positions in the trellis are weakened through

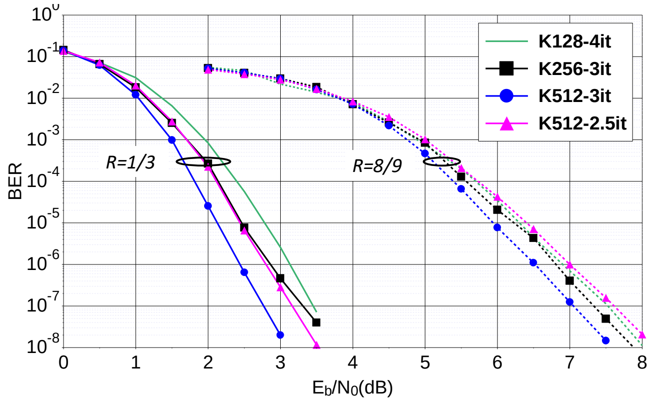

Fig. 5. BER performance for different frame sizes K with an X-window size of 32 with added ACQ before the first iteration.

the estimations of the initialization values at the X-window borders. Decoding with more iterations or adding an ACQ calculation in addition to the NII for each window mitigates the BER performance penalty. However, the former significantly increases the pipeline latency and the latter is especially costly for UXMAP decoders with larger frame sizes.

A compromise is to only add a very short ACQ calculation before the first iteration stage to supply it with initialization values for the state metrics. Figure 5 shows the BER performance for $K = 128$, 256 and 512 bits and an X-window size of 32 with an ACQ of 2 trellis sections.

For rate $1/3$, the coding gain when comparing $(128, 4)$ and $(512, 3)$ increases to more than 0.6 dB at a BER 10^{-6}, confirming the larger impact of the splitting of the trellis on larger frame sizes. The $(256, 3)$ and $(512, 2.5)$ configurations are now about 0.4 dB better than $(128, 4)$. This trend translates to the higher code rate of $8/9$, where the $(512, 3)$ configuration leads to a coding gain of approximately 0.5 dB whereas the performance of $(512, 2.5)$ is now almost matching the performance of $(128, 4)$.

Moreover, an ACQ of only two trellis sections can be implemented at negligible hardware overhead.

3.4. Estimates for Complete Decoders

Correlating the results from Figures 3, 4 and 5, allows to give good qualitative estimates for complete decoders which are listed in Table 3. The reference configuration of $(128, 4)$ which uses 8 half-iteration stages requires 12 mm^2, whereas the $(256, 3)$ configuration is estimated to occupy an area of $\approx$ 18 mm^2. Note that the throughput for this configuration would be doubled resulting in a throughput of over 200 Gb/s at a clock frequency of 800 Mhz. Moreover, moving from the $(128, 4)$ to the $(256, 3)$ or the $(512, 3)$ configuration results in an improved BER performance. The $(512, 2.5)$ configuration allows a throughput of 409 Gb/s at 30 mm^2, making it almost twice as area efficient as the $(128, 4)$ configuration.

Configuration	Area Est. [mm^2]	Throughput @ 800 MHz	Area Eff. [Gb/s/mm^2]
(128, 4)	12	102.4	8.5
(256, 3)	18	204.8	11.37
(512, 3)	36	409.6	11.37
(512, 2.5)	30	409.6	13,65

Table 3. Comparison of different decoder configurations.

4. CONCLUSION

In this work, we explore different implementations of fully pipelined iteration unrolled Turbo decoders, targeting ultra-high throughput applications.

Extending our previous work from [27], we show the impact of spatially parallelizing the X-windows of UXMAP decoders on the area of one half-iteration stage. Going from an X-window size of 128 down to 32 is associated with an area reduction of 40%. Further reduction through limiting the X-window size is minimal and comes at a cost in the BER performance. Motivated by the area saving through the reduction of the X-window size, we demonstrate the feasibility of reaching a throughput of up to 409 Gb/s for a frame size $K = 512$ bits, by correlating area complexity and BER performance of UXMAP decoders.

Moreover, we show that the required number of decoding iterations, i.e. hardware instances of half-iteration stages, can be further reduced by adding a very short ACQ ahead of the first iteration.

Our promising results show that UXMAP decoder hardware implementations are the prime candidates for ultra-high throughput, constituting a major milestone on the road towards Tb/s Turbo decoding.

5. REFERENCES

[1] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis, “5G-enabled tactile internet,” *IEEE Journal on Selected Areas in Commun.*, vol. 34, no. 3, pp. 460–473, March 2016.

[2] G. A. Akpakwu, B. J. Silva, G. P. Hancke, and A. M. Abu-Mahfouz, “A survey on 5g networks for the internet of things: Communication technologies and challenges,” *IEEE Access*, vol. 6, pp. 3619–3647, 2018.

[3] Third Generation Partnership Project, *LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (3GPP TS 36.213 version 13.1.0 Release 13)* , Apr. 2016.

[4] EPIC Project, “Enabling practical wireless Tb/s communications with next generation channel coding (EPIC),” 2019, https://epic-h2020.eu/, Last accessed on 2019-10-01.

[5] A. Li, L. Xiang, T. Chen, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, “VLSI implementation of fully parallel lte turbo decoders,” *IEEE Access*, vol. 4, pp. 323–346, 2016.

[6] Juntan Zhang and M. P. C. Fossorier, “Shuffled iterative decoding,” *IEEE Transactions on Communications*, vol. 53, no. 2, pp. 209–213, Feb 2005.

[7] I. Tal and A. Vardy, “List decoding of polar codes,” *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015.

[8] S. Weithoffer, K. Kraft, and N. Wehn, “Bit-level pipelining for highly parallel turbo-code decoders: A critical assessment,” in *2017 IEEE AFRICON*, Sep. 2017, pp. 121–126.

[9] A. Balatsoukas-Stimming, M. Meidlinger, R. Ghanaatian, G. Matz, and A. Burg, “A fully-unrolled ’LDPC decoder based on quantized message passing,” in *IEEE Inter. Worksh. on Sig. Proc. (SiPS)*, Oct 2015, pp. 1–6.

[10] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Communications Letters*, vol. 15, no. 12, pp. 1378–1380, December 2011.

[11] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "237 gbit/s unrolled hardware polar decoder," *Electronics Letters*, vol. 51, no. 10, pp. 762–763, 2015.

[12] P. Schläfer, N. Wehn, M. Alles, and T. Lehnigk-Emden, "A new dimension of parallelism in ultra high throughput LDPC decoding," in *IEEE Inter. Worksh. on Sig. Proc. (SiPS)*, Oct 2013, pp. 153–158.

[13] S. Weithoffer, C. A. Nour, N. Wehn, C. Douillard, and C. Berrou, "25 years of turbo codes: From Mb/s to beyond 100 Gb/s," in *Int. Symp. on Turbo Codes and Iter. Inf. Proc. (ISTC)*, Dec 2018, pp. 1–6.

[14] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes," in *IEEE Int. Conf. on Commun. (ICC)*, May 1993, vol. 2, pp. 1064–1070 vol.2.

[15] C. Roth, S. Belfanti, C. Benkeser, and Q. Huang, "Efficient parallel turbo-decoding for high-throughput wireless systems," *IEEE TCAS I: Regular Papers*, vol. 61, no. 6, pp. 1824–1835, June 2014.

[16] Y. Sun and J.R. Cavallaro, "Efficient hardware implementation of a highly-parallel 3GPP LTE/LTE-advance turbo decoder," *Integration VLSI Journal*, 2010.

[17] A. Worm, H. Lamm, and N. Wehn, "Design of low-power high-speed maximum a priori decoder architectures," in *Design, Autom. and Test in Eur. Conf. (DATE)*, March 2001, pp. 258–265.

[18] M. May, T. Ilnseher, N. Wehn, and W. Raab, "A 150Mbit/s 3GPP LTE turbo code decoder," in *Design, Autom. and Test in Eur. Conf. (DATE)*, March 2010, pp. 1420–1425.

[19] M. May, C. Neeb, and N. Wehn, "Evaluation of high throughput turbo-decoder architectures," in *2007 IEEE International Symposium on Circuits and Systems*, May 2007, pp. 2770–2773.

[20] S. Weithoffer, F. Pohl, and N. Wehn, "On the applicability of trellis compression to turbo-code decoder hardware architectures," in *Int. Symp. on Turbo Codes and Iter. Inf. Proc. (ISTC)*, Sep. 2016, pp. 61–65.

[21] T. Ilnseher, F. Kienle, C. Weis, and N. Wehn, "A 2.15gbit/s turbo code decoder for lte advanced base station applications," in *Int. Symp. on Turbo Codes and Iter. Inf. Proc. (ISTC)*, Aug 2012, pp. 21–25.

[22] R. Shrestha and R. P. Paily, "High-throughput turbo decoder with parallel architecture for lte wireless communication standards," *IEEE TCAS I: Regular Papers*, vol. 61, no. 9, pp. 2699–2710, Sep. 2014.

[23] S. Weithoffer and N. Wehn, "Where to go from here? New cross layer techniques for LTE turbo-code decoding at high code rates," *Advances in Radio Science*, vol. 16, no. C., pp. 77–87, 2018.

[24] G. Wang, H. Shen, Y. Sun, J. R. Cavallaro, A. Vosoughi, and Y. Guo, "Parallel interleaver design for a high throughput HSPA+/LTE multi-standard turbo decoder," *IEEE TCAS I: Regular Papers*, vol. 61, no. 5, pp. 1376–1389, May 2014.

[25] R. G. Maunder, "A fully-parallel turbo decoding algorithm," *IEEE Trans. on Commun.*, vol. 63, no. 8, pp. 2762–2775, Aug 2015.

[26] S. Weithoffer, M. Herrmann, C. Kestel, and N. Wehn, "Advanced wireless digital baseband signal processing beyond 100 Gbit/s," in *2017 IEEE International Workshop on Signal Processing Systems (SiPS)*, Oct 2017, pp. 1–6.

[27] S. Weithoffer, O. Griebel, R. Klaimi, C. Abdel Nour, and N. Wehn, "Advanced Hardware Architectures for Turbo Code Decoding Beyond 100 Gb/s, submitted at WCNC 2020," working paper or preprint, Oct. 2019.

[28] 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 *Proceedings of IEEE ISSCC 2003*, Feb 2003, pp. 150–484 vol.1.

[29] R. Garzón-Bohórquez, C. Abdel Nour, and C. Douillard, "Improving turbo codes for 5G with parity puncture-constrained interleavers," in *Int. Symp. on Turbo Codes and Iter. Inf. Proc. (ISTC)*, Sep. 2016, pp. 151–155.

[30] R. Garzón-Bohórquez, C. Abdel Nour, and C. Douillard, "Protograph-based interleavers for punctured turbo codes," *IEEE Trans. on Commun.*, vol. 66, no. 5, pp. 1833–1844, May 2018.