Advanced Hardware Architectures for Turbo Code Decoding Beyond 100 Gb/s

Stefan Weithoffer, Oliver Griebel, Rami Klaimi, Charbel Abdel Nour, Norbert Wehn

Advanced Hardware Architectures for Turbo Code Decoding Beyond 100 Gb/s

Stefan Weithoffer†, Oliver Griebel*, Rami Klaimi†, Charbel Abdel Nour†, Norbert Wehn*
*Department of Electrical and Computer Engineering, Technische Universität Kaiserslautern
Email: *{griebel, wehn}@eit.uni-kl.de, †{stefan.weithoffer, rami.klaimi, charbel.abdelnour} @imt-atlantique.fr
†IMT Atlantique, Department of Electronics, Lab-STICC - UMR 6285

***Abstract*—In this paper, we present two new hardware architectures for Turbo Code decoding that combine functional, spatial and iteration parallelism. Our first architecture is the first fully pipelined iteration unrolled architecture that supports multiple frame sizes. This frame flexibility is achieved by providing a set of interleavers designed to achieve a hardware implementation with a reduced routing overhead. The second architecture efficiently utilizes the dynamics of the error rate distribution for different decoding iterations and is comprised of two stages. First, a fully pipelined iteration unrolled decoder stage applied for a pre-determined number of iterations and a second stage with an iterative afterburner-decoder activated only for frames not successfully decoded by the first stage. We give post place & route results for implementations of both architectures for a maximum frame size of $K = 128$ and demonstrate a throughput of 102.4 Gb/s in 28 nm FDSOI technology. With an area efficiency of 6.19 and 7.15 Gb/s/mm^2 our implementations clearly outperform state of the art.**

***Keywords*—Forward Error Correction, Turbo decoder, Fully Parallel, High-throughput.**

I. Introduction

The change towards our nowadays "*always-connected*" society has been founded on the evolution of mobile communication standards from 3G to 4G and now to the fifth generation 5G, while research is already looking "Beyond 5G" [1].

For the third and fourth generation mobile communication standards, *Turbo Codes* [2] were used as codes for the downlink channel, and although they will be replaced by *Low Density Parity Check* (LDPC) [3] codes in the 5G standard, they will continue to be part of 5G through the evolution of 4G.

In respect to the requirements of communication systems, LDPC and Turbo Codes have different strengths. While LDPC codes are well suited to achieve ultra high throughputs in the range of hundreds of Gb/s [4], [5], Turbo Codes are superior in terms of frame and rate flexibility in the medium throughput range of several Gb/s. With respect to ultra high throughput decoder architectures, however, flexibility is generally a challenging design aspect. While rate flexibility comes by design for Turbo Codes [6], the most advanced Turbo decoder implementations with respect to information throughput either have very limited or no flexibility at all with respect to frame sizes [7], [8].

In this work, we demonstrate, that with a suitable set of *Almost Regular Permutation* (ARP) interleavers [9], frame flexibility can be achieved for fully pipelined iteration unrolled decoders. Based on a set of interleavers for the frame sizes $K \in \{32, 64, 128\}$ we present a new decoder architecture, which also features a spacial parallelism of 4 and highly localized channel value pipelines.

The second challenge for highly parallel decoder architectures is area consumption. Most advanced implementations occupy area in the order of tens of mm^2 [7], [8]. Therefore, in this work, we combine *spatial*, *functional* and *iteration* parallelism in order to increase the area efficiency. Inspired by the approach from [10] for LDPC codes, we present an afterburner-based decoding variant for fully pipelined Turbo decoders to further increase the area efficiency: By using an iterative decoder stage as an afterburner in our second architecture, we are able to reduce the number of fully pipelined half iteration stages and consequently significantly decrease the area complexity. In unrolled architectures area and power consumption linearly depends on the iteration stages. Thus, the afterburner approach is an efficient way to move from worst case to typical number of iteration assumptions, thus reducing area and power.

The rest of this paper is structured as follows: Section II first gives an overview on the state-of-the-art in parallel architectures for Turbo Decoding. In Section III, we introduce our set of interleavers and present two new decoder architectures in Section IV. Place & route results along with results of power simulations and a comparison with state-of-the-art is given in Section V and Section VI concludes the paper.

II. Parallel Turbo Decoder Hardware Architectures

In its basic structure, a Turbo decoder consists of two component decoders, an interleaver Π and a de-interleaver Π^{-1} connected in an iterative loop [2] as shown in Figure 1. The two component decoders commonly employ the max-Log-MAP algorithm and exchange extrinsic information Λ^e. Their iterative processing continues until either a pre-determined maximum number of iterations is reached or an early-stopping criterion is fulfilled. One execution of a component decoder is counted as one *Half Iteration* (HI), whereas each completed loop is called a *(full) Iteration* (IT). Hardware architectures for Turbo Decoding are commonly comprised of one (parallel) decoder core, which alternately acts as component decoder 1 and component decoder 2.

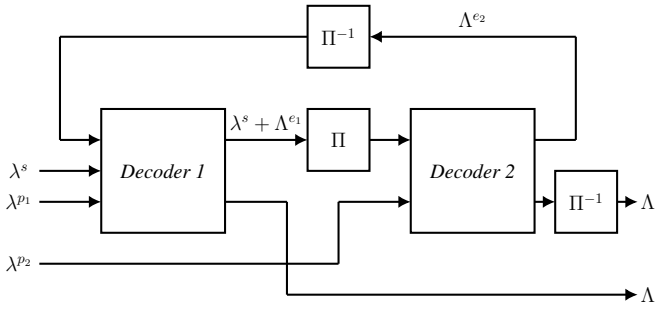


Figure 1: General Turbo Decoder structure.

In order to achieve a high throughput, state-of-the-art Turbo Decoder hardware implementations employ either *spatial* parallelism as in the *parallel MAP* (PMAP, [11]–[14]) architecture, or *functional* parallelism in the *pipelined MAP* (XMAP, [15]–[17]) architecture. The code trellis is in both cases split into smaller sub-trellises, which are then decoded either on parallel sub-decoder cores (PMAP), or several sub-trellises in parallel in a pipelined fashion (XMAP).

Figure 2 illustrates how different methods of parallel processing are used in state-of-the-art Turbo Decoders. Implementations with moderate levels of parallelism employ only one type of parallelisation and use either the PMAP or the XMAP architecture. Moreover, it is well known that for these architectures, the maximum degree of parallelism is limited, since the resulting small sub-blocks lead to an error correction performance loss [14], [17]. As a consequence, the maximum throughput of PMAP/XMAP in today's silicon technologies is limited to less than 10 Gb/s [8].

Therefore, in order to achieve a throughput in the order of $10 - 100$ Gb/s with a single decoder instance, fully parallel architectures which also employ *iteration* parallelism, i.e. the parallel processing of two or more HI, have to be considered. Decoder architectures which use iteration parallelism are the *fully parallel MAP* (FPMAP, [7]) and the *fully pipelined iteration unrolled XMAP* (UXMAP, [8]) architectures.

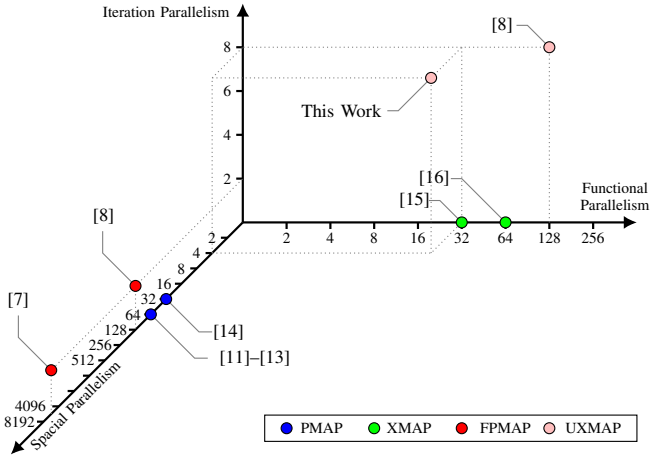


Figure 2: Parallelism in Turbo Decoder hardware architectures.

A. Fully Parallel MAP

For the FPMAP architecture, the decoding is reformulated to allow a splitting of the frame into sub-blocks of size 1, which are processed on individual parallel processing elements, thereby fully exhausting spacial parallelism. The state metric information for each step in the code trellis is directly exchanged between neighbouring elements. In addition, the FPMAP uses two groups of processing elements to calculate the decoding result for a complete iteration, one group for each component decoder. This shuffled decoding scheme [18] is used to exchange extrinsic information between the two concurrently running groups representing the component decoders leading to an iteration parallelism of 2 in Figure 2. Due to the extreme spacial parallelism, state metric information exchange between trellis sections that are far apart needs several iteration steps to propagate to the respective processing elements leading to an increase in the number of required iterations for a target error correcting performance level, especially for high code rates [8]. Moreover, the fully parallel exchange of extrinsic information poses a challenge for the implementation of flexible decoders, since the support of different interleaver patterns is often not possible [7]. To mitigate this issue, it was proposed to use unequal window lengths in [19]. However, no implementation results are given in [19].

B. Iteration Unrolled XMAP

Combining iteration parallelism with fully pipelined component decoders leads to the fully pipelined *Iteration Unrolled XMAP* (UXMAP) decoder architecture. In this hardware architecture, the iterative loop is fully unrolled onto a decoder pipeline in which complete frames are processed in parallel as they are progressing through the pipeline. The individual component decoder instances are called *iteration stages* and implement a pipelined serial max-Log-MAP processing of complete code blocks [8].

Since complete decoded code blocks are output in each clock cycle, the UXMAP architecture guarantees a very high throughput albeit at a considerable area cost: The area consumption for pipelining the serial max-Log-MAP processing of complete code blocks grows linearly with the frame size and the number of iteration stages. Similar to the FPMAP, flexible interleaving is challenging for UXMAP decoders, since it has to be realized within the pipeline. Therefore, the only reported UXMAP implementation only supported the single frame size of $K = 128$ [8].

However with careful selection of a set of interleavers, a frame flexible iteration unrolled decoder can be realized with a tolerable amount of multiplexing between the different iteration stages.

III. Interleaving

To achieve frame flexibility, we consider sets of *Almost Regular Permutation* (ARP) interleavers [9]. It was shown in [20] that this family of interleavers can provide the same interleaving properties, guaranteeing minimum Hamming distance values at least as high as most known algebraic interleaver

families. An ARP interleaver is defined by a permutation period P, a shift vector $\mathbf{S}$ and a disorder degree Q. The interleaving function, defining connections between the bits of the frames at the input of the first and second decoders, is then given by (1), where K denotes the frame size:

$$\Pi_{\text{ARP}}(i) = \left(P \cdot i + \mathbf{S}_{(i \bmod Q)}\right) \bmod K \tag{1}$$

Since in the UXMAP decoder architecture complete frames are processed in parallel, also the interleaving of complete frames has to be done in parallel. To support different frame sizes $K \in \{K_0, K_1, \ldots, K_{n-1}\}$, the goal therefore is to find sets of n interleavers with the maximum number of overlapping connections, i.e.

$$\Pi_{K_0}(i) = \Pi_{K_1}(i) = \ldots = \Pi_{K_{n-1}}(i). \tag{2}$$

For the addresses that fulfill (2), the hardware realization can be reduced to a wired connection.

For the remaining addresses, satisfying additional constraints can also simplify the hardware implementation by limiting the number of required multiplexers. Without loss of generality, we limit our analysis to power-of-two frame sizes, namely with $K \in \{32, 64, 128\}$ bits. For supporting two frame sizes K_m and K_n where $K_m = 2K_n$, limiting the number of required multiplexers could simply be achieved by applying twice the interleaver of the smaller K_n size. Let us denote by $\Pi_{K\,2n}$ the resulting interleaver. However, applying $\Pi_{K_{2n}}$ comes at the price of a high communications performance penalty since addresses lower (resp. larger) than K_n are interleaved to addresses lower (resp. larger) than K_n. Consequently, the error correction capability is not expected to improve when increasing the frame size from K_n to K_m bits.

A compromise between hardware efficiency and communications performance can be achieved by directly designing Π_{K_m} with the least possible number of different connections with respect to $\Pi_{K_{2n}}$ and the lowest impact on performance. We propose to do that by having the maximum number of continuous addresses in $\Pi_{K_{2n}}$ that can be modified to apply their corresponding connections in Π_{K_m} by the simple introduction of the same address shift value s_Π. In other words, maximize l, the number of continuous addresses $i, i+1, i+2, \ldots, i+l$ while designing the interleaving functions of K_m and K_n such that:

$$\Pi_{K_n}(i + s_\Pi) \bmod K_m = \Pi_{K_m}(i). \tag{3}$$

For the considered cases, we have $s_\Pi = K_n$. Satisfying (3) for the largest possible number (l) of addresses reduces the complexity for switching between the parallel processing of multiple frames of size K_n and the processing of frames of size K_m. Figure 3 illustrates this on the example of the set of ARP interleavers $\{\Pi_{32}, \Pi_{64}, \Pi_{128}\}$ with $P = 9$, $\mathbf{S} = \{3, 13, 27, 5\}$ and $Q = 4$ which were also chosen with the methods described in [6].

Interleaver connections are highlighted for all three interleavers in reference to the next smaller frame size. Connections only defined by $\Pi_{128}(i)$ are drawn in grey color, while connections shared by Π_{64} and Π_{128} are drawn in red color. Lastly, connections shared by Π_{32} and Π_{64} are highlighted in blue color. Figure 3 shows a significant overlap between the different interleavers contained in the selected set. As we will show in Section IV-B, the proposed set of ARP interleavers clearly outperforms the set of comparable interleavers from the LTE standard in terms of implementation complexity.

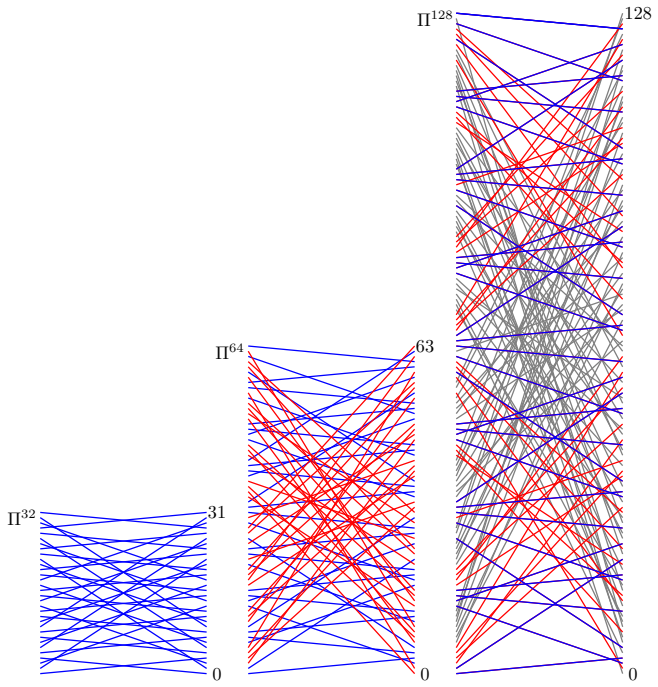
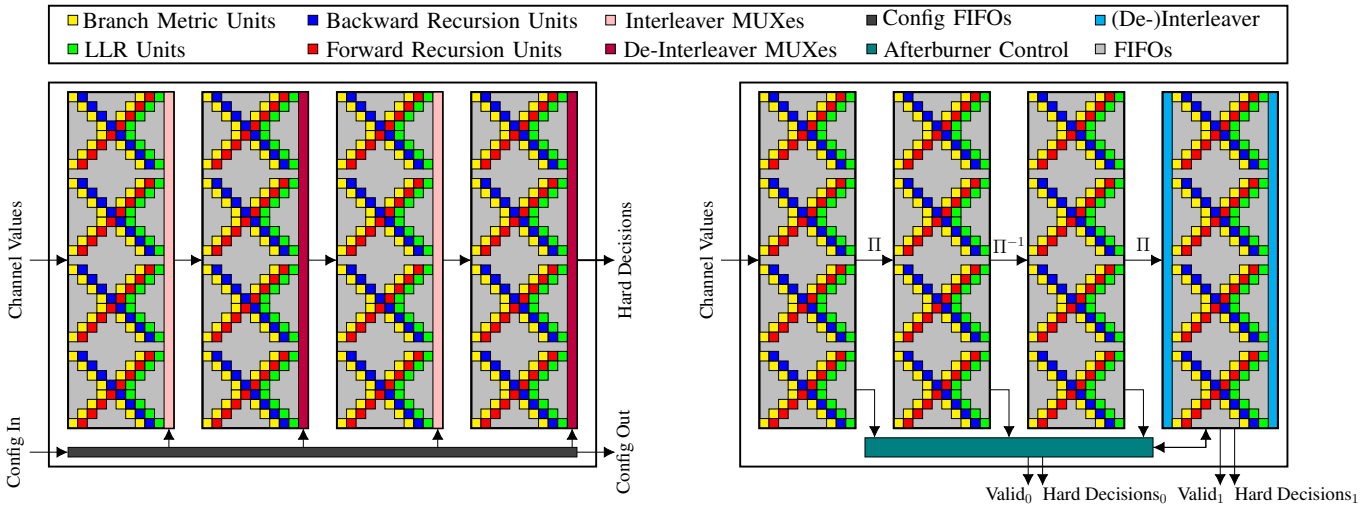


Figure 3: Overlap in the interleaver set $\{\Pi_{32}, \Pi_{64}, \Pi_{128}\}$ with $P = 9$, $\mathbf{S} = \{3, 13, 27, 5\}$ and $Q = 4$.

Moreover, it has been shown in [6], that with a careful selection of puncturing patterns, LTE interleavers are outperformed by this type of ARP interleavers in terms communications performance, especially for high coding rates.

IV. Proposed Architectures

In this section, we present two new hardware architectures for fully pipelined iteration unrolled Turbo Decoding based on the interleaver set discussed in Section III. Note, that both architectures use all three types of parallelism: Spatial, functional and iteration parallelism (see also Figure 2). As depicted in Figure 4, both architectures are composed of fully pipelined *Half-Iteration Stages* (HI-Stages), which consist of a number of parallel *X-Elements*, each with its own channel value FIFO. This localized channel value storage allows a more localized routing and a significantly lowered area footprint compared to the architecture from [8], where each HI-Stage consists of a monolithic X-Element and the channel value pipeline is separate from the HI-Stages. Moreover, through the reduced size of the individual X-Elements, the length of the FIFOs and therefore overall area consumption is lowered.



(a) FF-UXMAP (b) AB-UXMAP

Figure 4: Architecture schematics for the new iteration unrolled architectures.

A. Frame Flexible UXMAP

Figure 4 (a) shows the *Frame Flexible UXMAP* (FF-UXMAP) architecture. Based on the interleaver set presented in Section III, it supports flexibility w.r.t. frame sizes $K \in 32, 64, 128$. In addition, different combinations of smaller frame sizes can be decoded in parallel allowing for a constant throughput of 102.4 Gb/s for each combination, assuming a completely filled pipeline. The supported configuration modes are listed in Table I along with a complexity comparison with LTE. Since LTE does not support a frame size of 32, the comparison is done for two cases:

- 2 configurations: 128 and 64/64
- 4 configurations: 128, 64/64, 32/32/32/32 and 64/32/32

Thanks to the shared connections between the different interleavers, only half of the $2:1$ MUXes are needed compared to the LTE case. The rest of the connections are identical between the configurations 128 and 64/64 and can be realized by direct wire-connections, which significantly lowers the routing overhead compared to the LTE case where there is an overlap

Supported Configurations	2 configurations		4 configurations	
	LTE	This Work	LTE	This Work
Wires	3328	113152	**n/a**	59904
2:1 MUXes	209664	99840	**n/a**	99840
3:1 MUXes	/	/	**n/a**	53248

Table I: Complexity comparison of different interleaver sets.

for only two connections. Moreover, with additional $3:1$ MUXes, two more configuration modes (32/32/32/32 and 64/32/32) can be supported while keeping the total amount of multiplexers 25% lower than the LTE case. Note, that the numbers are representing the amount of MUXes and the amount of direct wire-connections for each (de-)interleaving instance assuming a quantization of 6 bits for the channel values and 7 bits for each extrinsic information to be (de-)interleaved. A $2-$bit FIFO buffer, shown in dark grey in Figure 4, is used to control the (de-)interleaver MUXes (pink and purple respectively) at the output of each HI-Stage.

B. UXMAP with Iterative Afterburner

The drawback of unrolled architectures is the static number of iterations, i.e. pipeline stages, that have to be fixed at design time. A worst case assumption on the iterations has to be made to enable good communication performance also for lower SNR regions. Serial architectures leverage stopping criterions to exploit the dynamics in the channel and stop iterations as early as possible.

In [10] it was proposed to utilize a hybrid afterburner approach to combine conventional min-sum decoding and the more complex *saturated min-sum* (SMS) decoding for LDPC codes. There, an advanced stopping criterion was used to detect erroneous decoding by the min-sum decoder and only activate the saturated min-sum decoder for the wrongly decoded frames. This sequential application of two variants min-sum decoding allowed for a SNR gain of up to 1.6 dB.

Instead of using multiple iterative decoder instances as in [10], in our architecture multiple frames are decoded in parallel in a pipelined afterburner stage where they are processed iteratively for a pre-determined number of HI after emerging from the UXMAP pipeline. The resulting *UXMAP with iterative afterburner* (AB-UXMAP) architecture is illustrated in Figure 4 (b), where, for more clarity, frame flexibility is not considered. In contrast to [10], the iterative afterburner stage is used here to minimize the number of needed HI-Stages for a target error correcting performance. Since the power consumption can be estimated to increase linearly with the number of HI-stages, this afterburner approach not only reduces significantly the area but also the power consumption.

To avoid a loss in error correcting performance, a reliable, low complexity stopping criterion is employed in the *Afterburner Control* unit in Figure 4 (b) to decide, which frames are sent to the afterburner stage for iterative decoding. A number

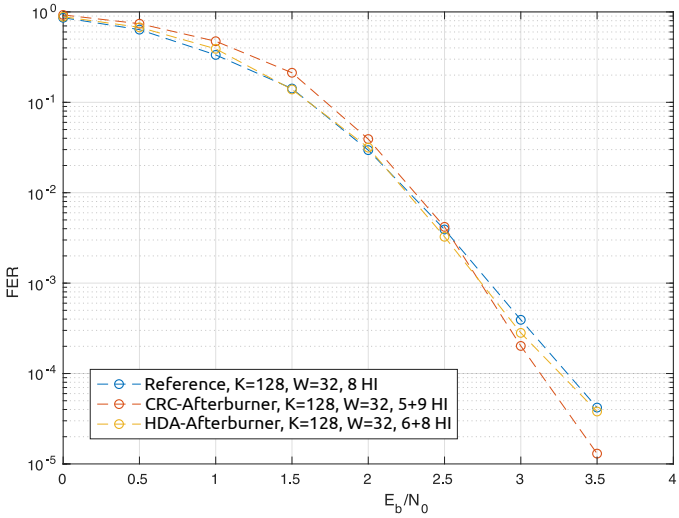


Figure 5: Comparison of the FER performance of different AB-UXMAP variants.

of stopping criteria are known from literature [21]. Since the hard decisions on the decoded frames are available in the pipeline with no overhead, the *Hard Decision Aided* (HDA) stopping criterion [21] and a *Cyclic Redundancy Check* (CRC) based criterion are low complexity choices.

Figure 5 shows *Frame Error Rate* (FER) simulation results of afterburner decoding with both criteria along with the reference consisting of a constant number of 8 HIs for a frame size of $K = 128$. For the afterburner decoding, the total number of HIs (HI-stages + iterations in the afterburner) for frames which go into the afterburner is set to 14. The amount of frames being iteratively decoded in the afterburner at any given time is limited to 32 which dictates the capacity of the afterburner pipeline.

The HDA-based afterburner configuration matches the FER performance of the reference for 6 pipelined HI and 8 "afterburner-HI" while the CRC based afterburner configuration only needs 5 pipelined HI and in the high SNR region even outperforms the reference by approximately 0.25 dB.

In order to avoid non-negligible rate loss due to the CRC at frame sizes < 200 bits, our implementation discussed in Section III uses the HDA criterion. Note, that for larger frame sizes, CRC becomes viable again, since the additional area savings due to lower number of initial pipelined HIs justify the rate loss due to the CRC.

V. Place & Route Results

The hardware architectures presented in Section IV were implemented in VHDL and placed & routed with *Synopsis IC Compiler* for a 28 nm FD-SOI process under worst case PVT *(Process/Voltage/Temperature)* constraints. For a fair comparison with previous works, the same frame size and number of iterations was used as in [8]. Table II lists the algorithmic parameters of the case study implementations. The max-Log-MAP algorithm with an *Extrinsic Scaling Factor* (ESF) of 0.75 and a radix-4 processing of the trellis [15] are used and the inputs are quantized with 6 bits. Due to the splitting of the decoding into sub-blocks, an initialization of the state metrics at the sub-block borders becomes necessary, which here is done via *Next Iteration Initialization* (NII, [22]).

	FF-UXMAP	AB-UXMAP
Frame Size K	128/64/32	128
Spacial Parallelism	4	
Radix	4	
Algorithm	max-Log-MAP with ESF = 0.75	
State Metric Initialization	Next Iteration Initialization [22]	
Input Quantization	6 bit	
Termination	Tailbiting	
Stopping Criterion	**n/a**	HDA

Table II: Algorithmic properties for the implementations.

The post place & route results are shown in Table III alongside published results for decoders with FPMAP, PMAP and XMAP architectures. These report a throughput ranging from 1 to tens of Gb/s. Since results from literature are reported for different technology nodes, we provide a scaling to 28 nm technology in the caption of Table III. For this, we cap the frequency scaling to 1000 MHz, which is the expected frequency cap necessary to preserve single cycle accesses to SRAM. Note, that in Table III "parallelism" refers to the amount of bits decoded in parallel.

A. Area and Throughput

To the best of our knowledge, no other reported implementations besides the previous work from [8] use a fully pipelined decoder architecture. Consequently, none of the compared implementations except [8] achieves a throughput of more than 16 Gb/s. Even when scaled to 28 nm technology, only [7] is expected to achieve a throughput of around 40 Gb/s. The presented architectures improve on previous work from [8] by reducing the area consumption by 30% and 40% respectively to 16.54 mm^2 and 14.32 mm^2. This, in turn increases the area efficiency to 6.19 $Gb/s/mm^2$ and 7.15 $Gb/s/mm^2$, thereby outperforming the most efficient reference implementation from [14] by 37% and 59%, even when scaling its results to 28 nm.

B. Layout

Layout pictures for our placed & routed designs are shown in Figure 6. Note that they are at the same scale, showing the correct relative sizes for both designs. The HI-Stages and the Afterburner Control (red in Figure 6 (b)) are clearly separated due to the hierarchical place & route process, but also the individual X-Elements can be clearly identified for both designs. Also note the iterative afterburner on the left in Figure 6 (b), where the impact of the internal (de-)interleaving is visible.

VI. Conclusion

In this work, we presented two new fully pipelined iteration unrolled hardware architectures for Turbo Decoding along with place & route results and power numbers for the respective case study implementations. Based on our investigations

	This Work		[8]	[12]	[14]	[7]	[16]	[15]
Architecture	FF-UXMAP	AB-UXMAP	UXMAP	PMAP		FPMAP	XMAP	
K	128/64/32	128	128	6144	6144	6144	6144	6144
Parallelism	128		128	64	32	6144	64	32
n_{IT}	4	3 + 4	4	5.5	5.5	39	5.5	7
Technology	28 nm		28 nm	90 nm$^{\natural}$ $^{\sharp}$	65 nm† ‡	65 nm† ‡	45 nm $\clubsuit$ $\spadesuit$	28 nm
Freq. $[MHz]$	800		800	625 (1000)	410 (1000)	100 (252)	600 (1000)	625
Throughput $[Gb/s]$	102.4		102.4	3.3 (5.29)	1.01 (2.47)	15.8 (39.86)	1.67 (3.2)	1.13
Area $[mm^2]$	16.54	14.32	23.61	19.75 (2.44)	2.49 (0.55)	109 (24.09)	2.43 (1.04)	0.49
Area Eff. $[Gb/s/mm^2]$	6.19	7.15	4.34	0.17 (2.17)	0.41 (4.49)	0.14 (1.65)	0.69 (2.68)	2.32

Table III: Comparison of implementation results for different turbo decoder architectures
Frequency scaling to 28 nm (Capped at 1000 MHz): $^{\natural}$: 2.52; †: 1.95; $\clubsuit$: 1.46; Area scaling to 28 nm: $^{\sharp}$: 0.40; ‡:0.51; $\spadesuit$: 0.69

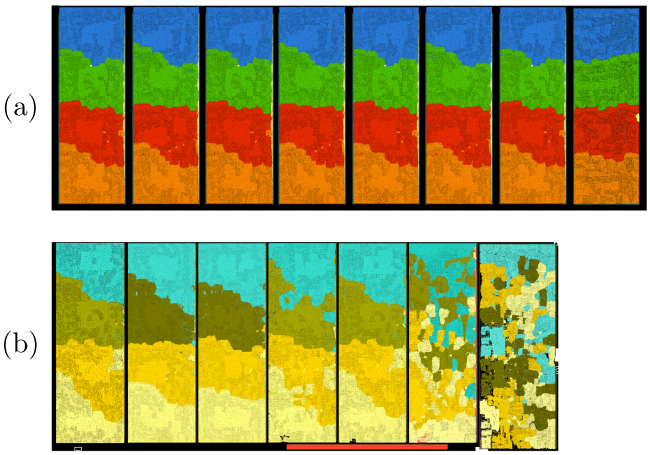


Figure 6: Layout of (a) FF-UXMAP and (b) AB-UXMAP decoders.

on interleaver sets, which can be implemented with a low routing complexity, we presented a frame flexible architecture supporting three different frame sizes and in total four different operating modes as well as the first iteration unrolled architecture for Turbo Decoding with an iterative afterburner. Both case study implementations achieve a throughput of 102.4 Gb/s and outstanding area efficiency numbers in 28 nm technology outperforming other state-of-the-art Turbo Decoder hardware architectures.

Acknowledgment

We gratefully acknowledge financial support by the EU (project-ID: 760150-EPIC)

References

[1] E. Dahlman, S. Parkvall, J. Peisa, and H. Tullberg. 5g evolution and beyond. In *2019 IEEE 20th Int. Workshop on Sig. Proc. Adv. in Wireless Commun. (SPAWC)*, pages 1–5, July 2019.

[2] C. Berrou and A. Glavieux. Near optimum error correcting coding and decoding: turbo-codes. *IEEE Trans. on Commun.*, 44(10):1261–1271, Oct 1996.

[3] D. J. C. MacKay and R. M. Neal. Near shannon limit performance of low density parity check codes. *Electronics Letters*, 33(6):457–458, March 1997.

[4] 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)*, pages 153–158, Oct 2013.

[5] 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)*, pages 1–6, Oct 2015.

[6] R. Garzón-Bohórquez, C. Abdel Nour, and C. Douillard. Protograph-based interleavers for punctured turbo codes. *IEEE Trans. on Commun.*, 66(5):1833–1844, May 2018.

[7] 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*, 4:323–346, 2016.

[8] 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. proc. (ISTC)*, pages 1–6, Dec 2018.

[9] C. Berrou, Y. Saouter, C. Douillard, S. Kerouedan, and M. Jezequel. Designing good permutations for turbo codes: towards a single model. In *IEEE Int. Conf. on Commun. (ICC)*, volume 1, pages 341–345, June 2004.

[10] S. Scholl, P. Schläfer, and N. Wehn. Saturated min-sum decoding: An "afterburner" for ldpc decoder hardware. In *Design, Autom.and Test in Eu. Conf. (DATE)*, pages 1219–1224, March 2016.

[11] 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. proc. (ISTC)*, pages 21–25, Aug 2012.

[12] R. Shrestha and R. P. Paily. High-throughput turbo decoder with parallel architecture for lte wireless communication standards. *IEEE TCAS I: Regular Papers*, 61(9):2699–2710, Sep. 2014.

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

[14] C. Roth, S. Belfanti, C. Benkeser, and Q. Huang. Efficient parallel turbo-decoding for high-throughput wireless systems. *IEEE TCAS I: Regular Papers*, 61(6):1824–1835, June 2014.

[15] 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. proc. (ISTC)*, pages 61–65, Sep. 2016.

[16] 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*, 61(5):1376–1389, May 2014.

[17] M. May, T. Ilnseher, N. Wehn, and W. Raab. A 150mbit/s 3gpp lte turbo code decoder. In *Design, Autom.and Test in Eu. Conf. (DATE)*, pages 1420–1425, March 2010.

[18] Juntan Zhang and M. P. C. Fossorier. Shuffled iterative decoding. *IEEE Trans. on Commun.*, 53(2):209–213, Feb 2005.

[19] L. Xiang, M. F. Brejza, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo. Arbitrarily parallel turbo decoding for ultra-reliable low latency communication in 3gpp lte. *IEEE Journ. on Selec. Areas in Commun.*, 37(4):826–838, April 2019.

[20] R. Garzón Bohórquez, C. A. Nour, and C. Douillard. On the equivalence of interleavers for turbo codes. *IEEE Wireless Commun. Letters*, 4(1):58–61, Feb 2015.

[21] R. Y. Shao, Shu Lin, and M. P. C. Fossorier. Two simple stopping criteria for turbo decoding. *IEEE Trans. on Commun.*, 47(8):1117–1120, Aug 1999.

[22] J. Dielissen and J. Huiskens. State Vector Reduction for Initialization of Sliding Windows MAP. In *Int. Symp. on Turbo codes and iter. proc. (ISTC)*, pages 387–390, Brest, France, September 2000.