HAL
open science

Low-complexity Computational Units for the Local-SOVA Decoding Algorithm

Stefan Weithoffer, Rami Klaimi, Charbel Abdel Nour, Norbert Wehn, Catherine Douillard

To cite this version:

Stefan Weithoffer, Rami Klaimi, Charbel Abdel Nour, Norbert Wehn, Catherine Douillard. Low-complexity Computational Units for the Local-SOVA Decoding Algorithm. PIMRC 2020: IEEE International Symposium on Personal, Indoor and Mobile Radio Communications - ETTCOM Workshop, Aug 2020, London (Virtual), United Kingdom. 10.1109/PIMRC48278.2020.9217318 . hal-02545582v4

HAL Id: hal-02545582

https://hal.science/hal-02545582v4

Submitted on 12 May 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Low-complexity Computational Units for the Local-SOVA Decoding Algorithm

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

***Abstract*—Recently the Local-SOVA algorithm was suggested as an alternative to the max-Log MAP algorithm commonly used for decoding Turbo codes. In this work, we introduce new complexity reductions to the Local-SOVA algorithm, which allow an efficient implementation at a marginal BER penalty of 0.05 dB. Furthermore, we present the first hardware architectures for the computational units of the Local-SOVA algorithm, namely the add-compare select unit and the soft output unit of the Local-SOVA for radix orders 2, 4 and 8 are proposed.**

We provide place & route implementation results for 28nm technology and demonstrate an area reduction of $46-75\%$ for the soft output unit for radix orders ≥ 4 in comparison with the respective max-Log MAP soft output unit. These area reductions compensate for the overhead in the add compare select unit, resulting in overall area saving of around $27-46\%$ compared to the max-Log-MAP. These savings simplify the design and implementation of high throughput Turbo decoders.

***Keywords*—Local-SOVA, Turbo codes, Soft-input soft-ouput decoding, Soft-ouput Viterbi algorithm, High-radix decoding**

I. Introduction

Today the interest in increased connectivity goes well beyond entertainment applications towards population-wide distance learning and home office programs. The associated requirements for reliable communications call for efficient *Forward Error Correction* (FEC).

Turbo codes [1] were adopted as FEC codes in 3G and 4G standards and will continue to be part of 5G through the evolution of LTE [2] complementing the 5G New Radio. A Turbo decoder consists in its basic structure of two component decoders, an interleaver Π and a de-interleaver Π^{-1} connected in an iterative loop in which the two component decoders exchange extrinsic information Λ^e [3].

For the component decoding, the BCJR algorithm (named after Bahl-Cocke-Jelinek and Raviv) [4] is used. However, in order to support efficient hardware implementation, its counterpart in the logarithmic domain, the Log-MAP algorithm, and its simplified version max-Log MAP (MLM) algorithm [5] are typically used in conjunction with a scaling of the extrinsic information with an *extrinsic scaling factor* (ESF, [6]) to avoid an overestimate of the calculated extrinsics.

The BCJR and its derivatives are inherrently serial in nature due to the recursive calculation of state metrics (see II), which limits the decoder throughput. In order to achieve a throughput in the order of Gb/s, decoder architecture archetypes achieve a high throughput by employing either *spatial* parallelism [7], [8] or *functional* parallelism [9]–[12] at decoder level. The code trellis is split into smaller sub-trellises, which are then decoded in parallel on several sub-decoder cores (spatial parallelism) or in a pipeline (functional parallelism).

Decoding with a higher radix is an essential technique for increasing the decoder throughput [13]. Higher radices process several consecutive trellis steps in parallel, which leads to a reduction in decoding latency, which in turn, increases the decoder throughput. Furthermore the state metric memory is reduced which translates into an area reduction.

For teraherz applications, a very high throughput towards Tb/s is required [14]. This throughput requirement is particularly challenging for Turbo decoders and requires improvements from architecural and algorithmic side. From an architectural perspective high radix decoding has a particular impact on fully pipelined iteration unrolled decoders as demonstrated in [11], [12], [15]. There, a higher radix reduces the length of the decoder pipeline, thereby reducing the area overhead in the pipelines, in turn allowing for larger frame sizes. Therefore, a high radix decoding is highly desirable. However, the increased complexity and long critical path for higher radix orders make radix orders ≥ 8 inefficient and Turbo decoders typically use radix-2 or radix-4 computations.

To this end, in [16], the *Local-SOVA* algorithm was recently proposed as a path based formulation of the forward and backward recursions of the MLM algorithm together with the extrinsic update rules of Hagenauer [17] and Battail [18] inspired by their use in the modified *Soft Output Viterbi Algorithm* (SOVA) [19]. The path based state metric recursion and the computation of the extrinsics using the update rules yield the same extrinsic value as the MLM. For radix orders ≥ 2, the Local-SOVA was shown to exhibit a lower computational complexity compared to the MLM. The complexity analysis, however, was only performed in terms of the number of additions and comparisons which does not directly translate to hardware implementation.

In this work, we present new complexity reductions to the Local-SOVA algorithm as well as the first hardware architectures and implementation results for Local-SOVA computational units with radix orders $\in \{2, 4, 8\}$. The implementation results are then compared with their respective MLM counterparts. Our results show an area complexity reduction of up to 75%, well beyond the $28-46\%$ predicted in [16].

The rest of this paper is structured as follows: First, section II and III recount the MLM and Local-SOVA algorithms. Then, we present our proposed new algorithmic complexity reductions in section IV along with BER simulation results. The efficient hardware architectures and the placement & routing results of the implementations are described and discussed in sections V and VI before section VII concludes the paper.

II. MAP ALGORITHM

In the following, we will focus on the MLM algorithm, but the results are equally applicable to the Log-MAP algorithm. In the MLM algorithm, a *forward state metric* α_i^s, *backward state metric* $\beta_{i+1}^{s'}$ and a *branch metric* $\gamma_{i,i+1}^{s,s'}$ are assigned to each state s, s' and the branch (s, s') respectively. The forward and backward metrics are calculated recursively:

$$\alpha_{i+1} = \max_{\forall s} \left(\alpha_i + \gamma_{i,i+1}\right) \tag{1}$$

$$\beta_i = \max_{\forall s'} \left(\beta_{i+1} + \gamma_{i,i+1}\right). \tag{2}$$

Note that for better readability in the following, we will sometimes omit the explicit labeling via s, s' and (s, s'). From these metrics, an *a Posteriori Probability* (APP) *Logarithmic Likelihood Ratio* (LLR) Λ_i is calculated by considering the "best" branch carrying the hard decision $u = 0$ and the best branch carrying the hard decision $u = 1$:

$$\begin{aligned}\Lambda_i = &\max_{\forall(s,s')} \left(\gamma_{i,i+1}^{u=0} + \alpha_i + \beta_{i+1}\right) \\ &- \max_{\forall(s,s')} \left(\gamma_{i,i+1}^{u=1} + \alpha_i + \beta_{i+1}\right).\end{aligned} \tag{3}$$

Processing k trellis steps at the same time, referred to as *radix-2^k* decoding [13], reduces the state metric memory and the decoding latency.

III. LOCAL-SOVA ALGORITHM

Local-SOVA algorithm [16] operates on paths in the trellis diagram stretching along k trellis steps $i, i+1, \ldots i+(k-1)$ for a radix order $R = 2^k$. A path in the trellis is defined as a 3-tuple consisting of a real-valued path metric M, a set of binary hard decisions $\mathbf{u} = \{u_i, u_{i+1}, \ldots, u_{i+(k-1)}\}$ and a set of positive real-valued reliability values $\mathbf{L} = \{L_i, L_{i+1}, \ldots, L_{i+(k-1)}\}$ (see also Fig. 1):

$$P = \{M, \mathbf{u}, \mathbf{L}\} \in \mathbb{R} \times \{0,1\}^k \times (\mathbb{R}^+)^k. \tag{4}$$

The path metric M along a branch (s, s') in a radix-2^k trellis is composed of α and β state metrics which themselves are calculated following Eq. 1 and 2 as well as the *branch* metrics:

$$M = \alpha_k(s) + \gamma_k(s, s') + \beta_{k+1}(s'). \tag{5}$$

The hard decisions $\mathbf{u}$ correspond to the data bit carried by the corresponding trellis branches in the trellis diagram (0 for a dashed line or 1 for a solid line in Figure 1). The reliabilities $\mathbf{L}$ associated with the respective hard decision are initialized to ∞ (or the largest quantization value) [16]. Based on the path structure, the Local-SOVA defines a merge operation, which merges two paths $P_a = \{M_a, \mathbf{u^a}, \mathbf{L^a}\}$ and

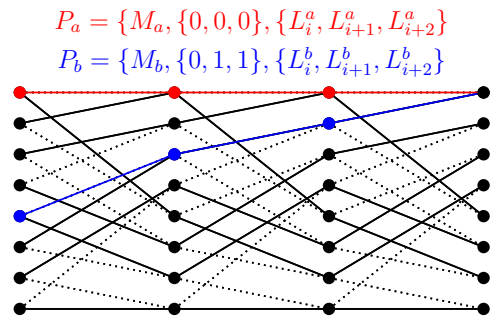

Figure 1: Two radix-8 ($k = 3$) paths in the trellis (Eq. 4)

$P_b = \{M_b, \mathbf{u^b}, \mathbf{L^b}\}$ into one path $P_c = \{M_c, \mathbf{u^c}, \mathbf{L^c}\}$ by three functions:

$$M_c = f_0(M_a, M_b) = \max(M_a, M_b) \tag{6}$$

$$\mathbf{u^c} = f_1(\mathbf{u^a}, \mathbf{u^b}) = \begin{cases} \mathbf{u^a} & \text{, if } f_0(M_a, M_b) = M_a \\ \mathbf{u^b} & \text{, if } f_0(M_a, M_b) = M_b \end{cases} \tag{7}$$

$$\mathbf{L^c} = \left\{L_i^c | L_i^c = f_2(L_i^a, L_i^b)\right\}. \tag{8}$$

If all paths at a trellis stage are merged through application of f_0 and f_1, the resulting path P_c is considered the *maximum likelihood* (ML) path and it carries the hard decision provided by the decoder. The update of the reliability values $\mathbf{L}$ through f_2 is performed on the basis of the Hagenauer rule (HR) [17] and the Battail rule (BR) [18], [20].

Following [16], let P_a and P_b be two paths to be merged and M_p and $M_{p'}$ be defined as

$$M_p = \max(M_a, M_b) \qquad M_{p'} = \min(M_a, M_b). \tag{9}$$

The metric difference between P_a and P_b is then defined as $\Delta_{p,p'} = M_p - M_{p'}$ and is always positive. Then the updates on L_i^c are performed as follows:

$$\begin{aligned} L_i^c =& f_2(L_i^a, L_i^b) \\ =& \begin{cases} \min\left(L_{p'}^i, \Delta_{p,p'}\right) & \text{, if } u_i^a \neq u_i^b \text{ (HR)} \\ \min\left(L_{p'}^i, \Delta_{p,p'} + L_p^i\right) & \text{, if } u_i^a = u_i^b \text{ (BR).} \end{cases} \end{aligned} \tag{10}$$

Using a Dirac measure $\delta(u_i^a, u_i^b)$, Eq. 10 can be written as

$$\begin{aligned} L_i^c =& \phi(L_i^{p'}, L_i^p, u_i^a, u_i^b) \\ =& \min\left(L_{p'}^i, \Delta_{p,p'} + \delta(u_i^a, u_i^b \cdot L_p^i\right), \\ \text{with: } \delta(u_i^a, u_i^b) =& \begin{cases} 1 & \text{, if } u_i^a = u_i^b \\ 0 & \text{, if } u_i^a \neq u_i^b. \end{cases} \end{aligned} \tag{11}$$

With Eqs. 6, 7 and 11, the MLM algorithm can be expressed on the basis of paths and the merge operation. For each trellis stage i, for the set of paths carrying $u_i = 0$ and the set of paths carrying $u_i = 1$, the respective ML paths $P_{ML}^{0,i}$ and $P_{ML}^{1,i}$ are found via a tree of merge operations. Note that this always requires the use of BR for the update of the reliability values $\mathbf{L}$. Then $P_{ML}^{0,i}$ and $P_{ML}^{1,i}$ are merged together, which gives the hard decision of the MLM algorithm for trellis step i, u_i as well as its reliability value L_i via HR:

$$L_i = \min\left(\infty, |M_1 - M_0|\right). \tag{12}$$

Note, that the left operand of the min-operation in Eq. 12 is ∞, since all reliability values are initialized to ∞. From the hard decision u_i and its reliability value L_i, we can calculate the output LLR for trellis step i:

$$\Lambda_k = (2 \cdot u_k - 1) \cdot L_k. \tag{13}$$

In [16] the commutativity and associativity of the merge operation were proven which allows merging in an arbitrary order. In particular, the merges can be rearranged so that paths with the same ending state are merged first. The merge operation for two paths P_a and P_b then yields the path metric

$$\begin{aligned} M_c &= \max\left(M_a, M_b\right) && (14)\\ &= \max\left(\alpha_k^{s_0} + \gamma_{k,k+1}^{s_0,s'} + \beta_{k+1}^{s'}, \alpha_k^{s_1} + \gamma_{k,k+1}^{s_1,s'} + \beta_{k+1}^{s'}\right) \\ &= \max\left(\alpha_k^{s_0} + \gamma_{k,k+1}^{s_0,s'}, \alpha_k^{s_1} + \gamma_{k,k+1}^{s_1,s'}\right) + \beta_{k+1}^{s'} && (15)\\ &= \alpha_{k+1}^{s'} + \beta_{k+1}^{s'}, && (16) \end{aligned}$$

The reordering of merges leads to an implicit computation of the forward state metrics, as can be seen from Eqs. 15 and 16. Thus, the forward recursion can be performed through merges via Eqs. 6, 7 and 11.

This first phase of merges uses the sum of forward state metrics and branch metrics as *preliminary path metrics*, yielding one resulting path for each trellis state. In the following, we will speak of this first phase as *Add-Compare-Select* (ACS) phase. Note, however, that in this phase (and different from Eq. 1), also updates of $\mathbf{u}$ and $\mathbf{L}$ are performed.

In the second phase *Soft Output Unit phase* (SOU phase), the remaining paths are merged in a tree-like fashion after the preliminary path metrics for each state have been summed up with the corresponding backward state metric to compute the ML path, the according hard decisions and soft outputs. Here, the amount of merge tree layers solely depends on the memory length ν of the underlying convolutional code. Thus, for a memory length ν, the SOU merges require a binary tree of ν layers (see Figure 2). This is independent of the radix order, since for each trellis state one path is output from the ACS phase. In the case of $R = 2$, the ACS operation of the

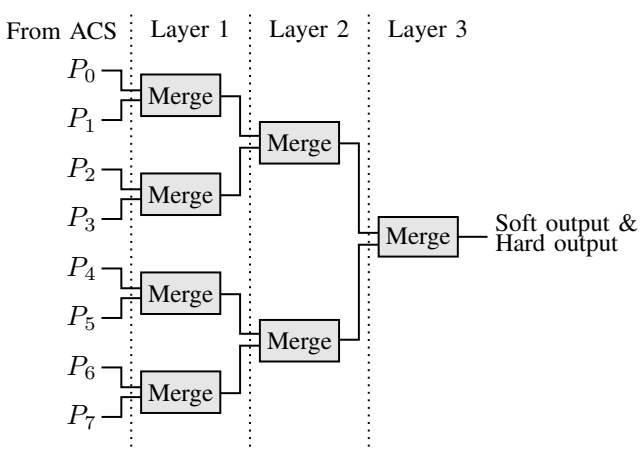

Figure 2: SOU phase example for a $\nu = 3$.

Local-SOVA merges for any state in each trellis step i two paths which are guaranteed to carry a different hard decision, allowing reliability updates solely using the HR.

For a radix order k, however, there are 2^k paths to be merged for each state in the radix-2^k trellis. Then, it is no longer guaranteed that paths intended to be merged carry different hard decisions. It was also proposed in [16] to use an arrangement of path merges in the ACS operation that minimizes the computational complexity for $R \geq 4$. In this

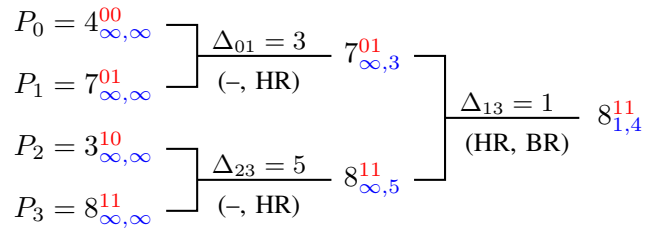

Figure 3: Radix-4 ACS merge operation example.

ordering, which is illustrated with an example for $R = 4$ in Fig. 3, the use of BR is minimized, since through the path ordering, in the first layer the BR update of the reliability value L for the second bit can be omitted (i.e. $\min(\infty, \infty + 3) = \infty$ for the merging of P_0 and P_1), while for the first bit a simplified HR can be used (i.e. $\min(\infty, 3) = 3$ for the merging of P_0 and P_1). It becomes obvious, however that the use of BR cannot entirely be avoided by a mere reordering of merges.

IV. New Complexity Reductions

Consider the update of the reliability value for the first bit in the second merge layer in Fig. 3:

$$L_0^{ML} = \min\left(\underbrace{|3-8|}_{\Delta_{23}=5}, \underbrace{|\min(4,7) - \min(3,8)|}_{\Delta_{13}=1} + \underbrace{|4-7|}_{\Delta_{01}=3}\right)$$

Here, BR needs to be used, since in the first merges P_1 and P_3 were selected which carry the same hard decision for the first bit. Therefore Δ_{13} alone does not give us information about the reliability. By adding to it Δ_{01}, we get the reliability of the hard decision in relation to the "*challenging hard decision*" of path P_0. This sum is then compared to the reliability w.r.t. the challenging hard decision of P_2, i.e. Δ_{23}.

In other words, the BR allows to get a reliability w.r.t. the challenging path from the other sub-tree at the cost of one addition. This addition, however, lies in the critical path of the ACS operation. Moreover, for large R, updates via nested BR cannot be avoided, which further increases the critical path.

ACS Phase: In order to avoid a long critical path resulting from the nesting of BR updates, we propose a modified ACS operation. As outlined in the previous paragraph, finding the individual reliability values L_i^{ML} for the hard decisions $\mathbf{u}^{ML}$ carried by the ML path P_{ML} surmounts to finding the best challenging path w.r.t. this bit, i.e. the best path which has a different hard decision from the ML path *for this bit*.

Therefore, it is enough to find for each bit the *best-0 path* and the *best-1 path* and selecting this path as preliminary reliability or challenging metric value l_i^{ML}. The full reliability value can then be generated by calculating $L_i^{ML} = |M^{ML} - l_i^{ML}|$ in the SOU phase.

Consequently, the ACS operation uses a slightly modified *ACS-Path* definition $P^{ACS} = \{M, \mathbf{u}, \mathbf{l}\}$ where the set of reliability values $\mathbf{L}$ is replaced with the set of *best challenging* metrics $\mathbf{l}$, which are calculated by

$$l_i^{ML} = \begin{cases} M^{best-0} & \text{, if } u_i^{ML} = 1 \\ M^{best-1} & \text{, if } u_i^{ML} = 0, \end{cases} \tag{17}$$

where M^{best-0} and M^{best-1} are given by

$$M^{best-0} = \max\{M_p | u_i = 0\} \tag{18}$$

$$M^{best-1} = \max\{M_p | u_i = 1\}. \tag{19}$$

Eqs. 6 and 7 remain unchanged. Note that Eqs. 6, 18 and 19 can not only be processed in parallel as opposed to the original nested calculations, but also share a number of comparisons. Thus, they can be obtained as a side-result of Eq. 6 with minimal overhead (see Section V).

SOU Phase: First, the path metrics $M_{s'}$ have to be summed up with the respective $\beta_{s'}$ and, due to the changed ACS update rule (Eq. 17), the reliability values have to be computed via $L_i^{s'} = |M^{s'} - l_i^{s'}|$. As discussed in Section III, the merges of the SOU phase are then processed in a binary tree. In contrast to the ACS phase, however, the merges cannot be reordered to avoid the use of BR since it is not known beforehand which hard decisions are carried by the ACS paths. In [16], it was therefore proposed to force the use of HR

$$L_i^c = \min\left(L_{p'}^i, \Delta_{p,p'}\right). \tag{20}$$

This resulted in a reduction of complutational complexity of about 10% at the cost of a BER performance loss of 0.3dB at BER 10^{-6}. This degradation stems from the fact that by forcing the HR for merges of paths carrying the same hard decision the real competing reliability is not recovered leading on an over estimation of the reliability value.

Therefore, we propose a reliability update that replaces the application of the BR by keeping the previous reliability value of the winning path:

$$L_i^c = \begin{cases} \min\left(L_{p'}^i, \Delta_{p,p'}\right) & \text{, if } u_i^a \neq u_i^b \text{ (HR)} \\ L_a^i & \text{, if } u_i^a = u_i^b \text{ and } M^a > M^b \\ L_b^i & \text{, if } u_i^a = u_i^b \text{ and } M^b > M^a \end{cases} \tag{21}$$

This simplification is intuitively motivated by the reasoning that the paths arriving to the SOU phase are already the result of the merging of 2^k paths for a radix-k. Thus, the reliability values resulting from the updates in the ACS phase already provide close estimates and therefore only need to be updated, if a merge with a contending path is performed in the SOU phase. Note, however, that this simplified update also only estimates the reliability value. Moreover, if applied to a tree of merges, highly reliable contending paths originating from a different bracket of the SOU merge tree might not get considered for the update of the reliability values.

However, we can explicitly consider them with a HR update. This is illustrated in Fig. 4, where four paths are merged (corresponding, for example to the first two layers in the SOU phase for an 8-state code). With the explicit consideration of

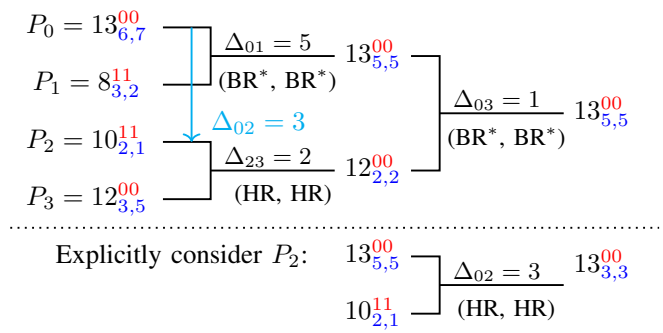

Figure 4: Merge tree for 4 paths. BR* corresponds to the simplified update via Eq. 21.

all contending paths (i.e. paths with different hard decision), we can define the merge operation for k paths $P_0, \ldots, P_{k-1}$ into one output path P_C:

$$M^c = \max(M^0, \ldots, M^{k-1}) \tag{22}$$

$$\mathbf{u^c} = \mathbf{u^P}, \text{ if } M^c = M^p \tag{23}$$

$$\mathbf{L_i^c} = \min\left(L_i^p, \min_{\forall P^j : u_i^j \neq u_i^p} \{\Delta_{jp}\}\right) \tag{24}$$

Equations 22 and 23 directly correspond to Eqs. 6 and 7 for nested merging of multiple paths. Note that the number of Δ_{jp} terms considered for the individual reliability updates can vary depending on i and the carried hard decisions of the paths arriving at the SOU phase. However, some Δ_{jp} terms will generally occur in more than one reliability update. In Fig. 3, for example, Δ_{02} is used to update the reliability values for both bits, since $\min(\Delta_{01} = 5, \Delta_{02} = 3) = 3$.

Simulation results: This complexity reduction can be applied to the whole merge tree or to sub-trees (i.e. a subset of layers in Fig. 2), Figure 5 shows the fixed-point BER performance simulation results for 8-state LTE Turbo codes with frame sizes $K \in \{128, 512, 1024\}$ in comparison with the MLM. The number of decoding iterations was fixed to 6, a windowed decoding with a window size of 32 and a channel value quantization of 6 bits were used. Applying the simplifications on the whole merge tree (L-SOVA I) leads to a penalty of about 0.2 dB at a BER of 10^{-7} in comparison to the MLM. Using Eq. 11 for the last merge (Layer 3 in Fig. 2) and performing a low complexity merging of the first two layers (L-SOVA II), however, shows a negligible performance penalty of < 0.05 dB.

V. Proposed Architectures

ACSU: In comparison to the MLM ACSU, which only computes the maximum state metric for each state, the Local-

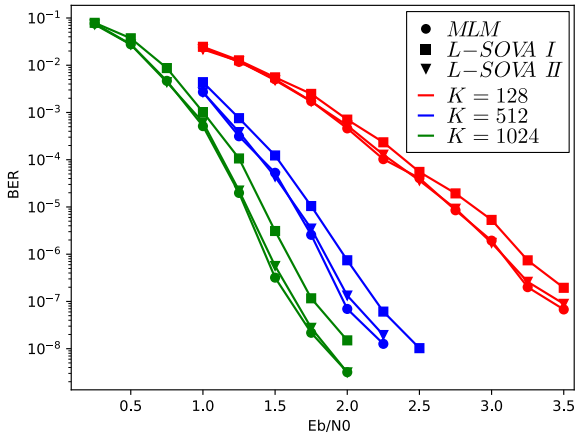

Figure 5: BER performance of the proposed low complexity Local-SOVA decoding for different LTE-A Turbo codes.

SOVA ACSU additionally outputs the best contending metrics for each hard decision. To achieve a short critical path, the ACSU for the MLM can be implemented by performing $\sum_{j=0}^{j=R-1} j$ comparisons and then selecting the maximum out of $R = 2^k$ (i.e. radix-2^k) inputs based on a lookup table (LUT) [21]. This is illustrated in Table I for the example of radix-4, where the maximum selection can be performed based on six comparison results. The l_i can be found based

max	$a-b$	$a-c$	$a-d$	$b-c$	$b-d$	$c-d$
a	-	-	-	?	?	?
b	+	?	?	-	-	?
c	?	+	?	+	?	-
d	?	?	+	?	+	+

Table I: Lookup table for maximum selection of four values.

on the same comparison results together with the carried hard decisions. Figure 6 shows the resulting hardware architecture schematic for radix order 2^k. Marked in blue are the LUTs and multiplexers (MUXes) that have to be added for the Local-SOVA ACSU.

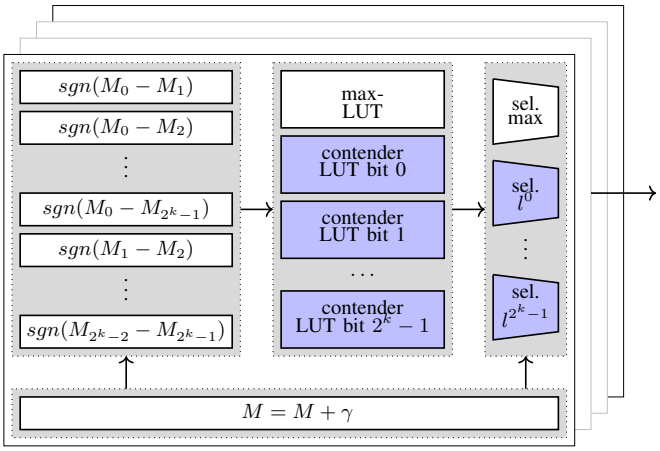

Figure 6: Schematic of the ACS unit.

SOU: The architecture of the SOU is illustrated in Fig. 7. It can be separated into three functional blocks: The simplified merging of the upper and lower sub-tree, the merge of the last tree layer and the calculation of the extrinsic output including scaling (with an ESF of 0.75 [6]). After the summation of

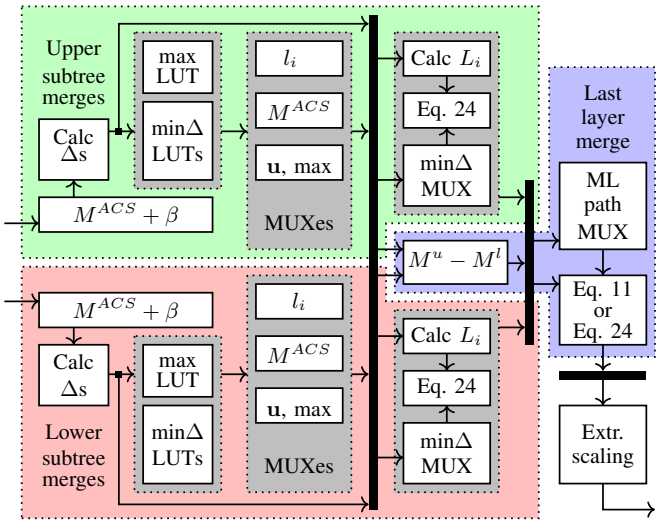

Figure 7: Schematic of the SOU.

$M_{s'}^{ACS} + \beta_{s'}$, the simplified merging of the subtrees realizes in the first step Eqs. 22 and 23 via table lookup and MUXes in the same way as the ACSU. In addition, the ACS path metric M^{ACS} and the preliminary reliabilities of the l_i and the Δ values calculated for the maximum selection are kept for the next step.

In the second step, the contender reliability values for the respective ML path for the upper and lower subtrees are computed, which further reduces the complexity, since it avoids calculating reliability values for the loosing paths, which would then be discarded. Then follows an update of the reliabilities via Eq. 24, for which the minimum Δ corresponding to a contender with a different hard decision is chosen via lookup. In parallel, the difference of the path metrics of the winning paths from the upper and lower subtrees is computed. In the last steps, the winning paths from upper and lower subtrees are merged via Eq. 24 (L-SOVA I) or Eq. 11 (L-SOVA II), and the extrinsics are calculated and scaled.

VI. Implementation Results

We implemented the hardware architectures presented in Section V in VHDL for radix orders $2, 4$ and 8 for the LTE Turbo code with a channel value quantization over 6 bits. The designs were placed & routed alongside the MLM ACSU and SOU with *Synopsis IC Compiler* for a 22 nm process under worst case PVT *(Process/Voltage/Temperature)* constraints and a target clock frequency of 833 MHz. For comparison with the MLM, the respective ACSU and SOU were as well implemented, placed & routed. The results are listed in Table II, where the area complexity is given as cell area consumption in [μm^2].

For the radix-2 case, the area complexity of ACSU and SOU is almost identical for the MLM and the Local-SOVA

	MLM	L-SOVA I	L-SOVA II	Area Saving	
ACSU	Cell area [μm^2]			[%]	
Radix-2	826	870		-5.33	
Radix-4	1572	1917		-21.95	
Radix-8	4101	5245		-27.90	
SOU	Cell area [μm^2]			LSOVA I	LSOVA II
Radix-2	1718	1598	1680	6.98	2.21
Radix-4	4259	2127	2295	**50.06**	**46.11**
Radix-8	10548	2658	2904	**74.80**	**72.47**
Sum	Cell area [μm^2]				
Radix-2	2544	2468	2550	2.99	-0.24
Radix-4	5831	4043	4212	**30.66**	**27.77**
Radix-8	14649	7947	8149	**45.75**	**44.37**

Table II: Post place & route cell area comparison in 22nm technology @ 833MHz target frequency.

which is in line with the complexity analysis of [16]. For the higher radixes, there is a overhead of 21.95% for the radix-4 ACSU and an area overhead of 27.9% for the radix-8 ACSU on the side of the Local-SOVA. This increase is attributed to the increased complexity of the table lookup for the contenders.

On the other hand, an area reduction of $72 - 75\%$ can be observed for the radix-8 case of the SOU and for the radix-4 configuration of the SOU, the area complexity is still halved in comparison to the respective MLM SOU. Comparing the sum of ACSU and SOU, area savings of $27 - 31\%$ and around 46% are achieved for the radix-4 and radix-8 case respectively.

VII. Conclusion

In this work, we proposed additional new complexity reductions for decoding with the Local-SOVA algorithm for radix orders $\in \{2, 4, 8\}$. The proposed simplifications result in a marginal loss in BER performance of 0.05dB if Eq. 11 is used in the last merge layer of the SOU. The corresponding efficient hardware architectures demonstrated a reduction of the area complexity of the soft output computation by around 50% for the radix-4 case and up to 75% for the radix-8 case in comparison with the respective max-Log-MAP units. These area savings outweigh the area overhead in the Local-SOVA ACSU of around 22% and 28% and thus lead to an overall area saving of up to 46% in comparison with the max-Log-MAP. In this work (as in [16]), we applied the Local-SOVA algorithm to Turbo decoding. Our results show it to be on par with the MLM, which is used in SoA, while exhibiting a significantly lower computational and implementation complexity.

Moreover, the complexity reductions become even more interesting for convolutional codes with ≥ 8 states. There, the Local-SOVA is contending with the SOVA, which is used for demodulation, decoding, equalization, etc. and has been widely applied in a range of communication and storage systems [22]. With a higher number of trellis states, the area savings through Eq. 21 become more pronounced and the absence of a traceback in the Local-SOVA can lead to a reduced decoding latency compared to the SOVA.

Acknowledgment

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

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. In *Proceedings of ICC '93 - IEEE Intern. Conf. on Commun.*, volume 2, pages 1064–1070 vol.2, May 1993.

[2] 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)* , April 2016.

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

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

[5] P. Robertson, E. Villebrun, and P. Hoeher. A Comparison of Optimal and Sub-Optimal MAP decoding Algorithms Operating in the Log-Domain. In *Proceedings of ICC '95 - IEEE Intern. Conf. on Commun.*, pages 1009–1013, Seattle, Washington, USA, June 1995.

[6] J. Vogt and A. Finger. Improving the max-log-MAP turbo decoder. *Electronics Letters*, 36(23):1937–1939, Nov 2000.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] S. Weithoffer, C. Abdel 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.

[12] 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, October 2019.

[13] G. Fettweis and H. Meyr. Parallel Viterbi algorithm implementation: breaking the ACS-bottleneck. *IEEE Trans. on Commun.*, 37(8):785–790, Aug 1989.

[14] EPIC Project. Enabling Practical Wireless Tb/s Communications with Next Generation Channel Coding (EPIC), 2020. https://epic-h2020.eu/.

[15] S. Weithoffer, R. Klaimi, C. Abdel Nour, N. Wehn, and C. Douillard. Fully Pipelined Iteration Unrolled Decoders – The Road to Tb/s Turbo Decoding. In *ICASSP 2020 : 45th Int. Conference on Acoustics, Speech, and Signal Processing*, Barcelona, Spain, May 2020.

[16] V. H. S. Le, C. Abdel Nour, E. Boutillon, and C. Douillard. Revisiting the Max-Log-Map Algorithm With SOVA Update Rules: New Simplifications for High-Radix SISO Decoders. *IEEE Trans. on Commun.*, 68(4):1991–2004, 2020.

[17] J. Hagenauer and P. Hoeher. A Viterbi algorithm with soft-decision outputs and its applications. In *1989 IEEE Global Telecomm. Conf. and Exhibition 'Commun. Techn. for the 1990s and Beyond'*, pages 1680–1686 vol.3, Nov 1989.

[18] Gérard Battail. Pondération des symboles décodés par l'algorithme de Viterbi. In *Annales des telecommunications*, volume 42, pages 31–38. Springer, 1987.

[19] M. P. C. Fossorier, F. Burkert, Shu Lin, and J. Hagenauer. On the equivalence between SOVA and max-log-MAP decodings. *IEEE Communications Letters*, 2(5):137–139, 1998.

[20] Lang Lin and R. S. Cheng. Improvements in SOVA-based decoding for turbo codes. In *Proceedings of ICC'97 - Intern. Conf. on Commun.*, volume 3, pages 1473–1478 vol.3, June 1997.

[21] W. Byun and J. Kim. High-speed radix-4 Add-Compare-Select unit for next generation communication systems. In *2013 Int. SoC Design Conf. (ISOCC)*, pages 1–2, 2013.

[22] Q. Huang, Q. Xiao, L. Quan, Z. Wang, and S. Wang. Trimming Soft-Input Soft-Output Viterbi Algorithms. *IEEE Trans. on Commun.*, 64(7):2952–2960, 2016.