Bit-by-Bit Iterative Decoding Expedites the Convergence of Repeat Accumulate Decoders

Jinhui Chen, Wenbo Zhang, Robert G. Maunder and Lajos Hanzo
School of ECS, University of Southampton, SO17 1BJ, United Kingdom.
Email: {jc9e12, wz4g11, rm, lh}@ecs.soton.ac.uk, http://www-mobile.ecs.soton.ac.uk

***Abstract*—In this paper, we propose bit-by-bit iterative decoding for expediting the convergence of Repeat Accumulate (RA) decoders. In a conventional RA decoder, the repeat and accumulate component decoders are operated iteratively, in order to facilitate near-capacity communication. However, whenever one decoder is activated, the other is kept idle. The outputs of the active component decoder are stored until its operation is completed, whereupon the outputs are forwarded to the other decoder all at once and the activation of the decoders is swapped. The proposed bit-by-bit RA decoder expedites this process by allowing both component decoders to operate simultaneously, continuously exchanging outputs without buffering. We present both EXtrinsic Information Transfer (EXIT) charts and Bit Error Ratio (BER) results, which demonstrate that the proposed bit-by-bit RA decoder requires fewer decoding iterations to converge, at the cost of a slightly increased complexity per decoding iteration. Overall, we demonstrate that in a range of practical scenarios, the proposed bit-by-bit RA offers gains of up to 0.86 dB, without imposing any additional decoding complexity and without requiring any additional transmission-energy, -bandwidth or -duration.**

***Index Terms*—Repeat accumulate code, channel coding, iterative decoding, EXIT chart, decoding convergence.**

I. Introduction

REPEAT Accumulate (RA) codes [1] have been proposed as a class of low-complexity "turbo-like" channel codes, which facilitate near-capacity communication. An RA encoder may be considered to be a special case of a self-concatenated convolutional encoder [2], [3], which comprises a serial concatenation of a repetition encoder, a pseudo-random interleaver [4] and a Unity-rate Recursive Convolutional (URC) encoder [5], [6]. More specifically, this URC encoder is also known as an accumulator in the special case, where its output bits are obtained as a cumulative modulo-2 sum of its input bits [7]. However, regardless of the URC encoder design, the term 'RA code' is used instead of 'self-concatenated code' throughout this paper, owing to its greater familiarity. Accordingly, the RA decoder comprises a corresponding serial concatenation of Soft-Input Soft-Output (SISO) repeat and URC decoders [8]–[10], which iteratively exchange increasingly-reliable soft information through an interleaver and a deinterleaver [1].

Owing to its relatively low complexity and its suitability to parallel processing, the operation of the repeat decoder can typically be completed in significantly less time than that required for the URC decoder. This is because the URC decoder employs the trellis-based Logarithmic Maximum *A posteriori* Probability (Log-MAP) algorithm [11], which has a relatively high complexity and is not well suited to parallel processing. A number of techniques have been proposed for increasing the parallelism of the Log-MAP algorithm, which could be applied to RA decoders in order to mitigate the corresponding bottleneck. In [12], the parallelism of the Log-MAP algorithm was increased by decomposing the trellis into K number of sub-trellises and performing the Log-MAP algorithm on each in parallel, hence reducing the corresponding processing time by a factor of $1/K$. Other approaches include pipelining the operations of the Log-MAP algorithm [13], so that different operations are performed in parallel, in different parts of the decoding hardware. Similarly, the radix-4 technique can be employed for processing two trellis stages at a time [13]. Furthermore, some of the Log-MAP calculations can be replaced with lower-complexity approximations [11], so that a greater number of these calculations can be performed in parallel using the same amount of hardware. However, the increased parallelism and reduced processing time offered by these techniques is typically achieved at the cost of either increasing the number of decoding iterations required or degrading the error correction performance [8], [9].

While the above-mentioned papers have proposed techniques for increasing the parallelism of the Log-MAP algorithm, they all consider the corresponding SISO component decoder to be activated alternately with the turbo-like code's other SISO component decoder. More specifically, although soft information outputs may be generated throughout the operation of the active SISO component decoder, these are buffered for forwarding to the other SISO component decoder on a frame-by-frame basis, rather than on a bit-by-bit basis. As a result, this other decoder must remain idle throughout the operation of the active decoder. The other decoder is only activated when the operation of the first SISO component decoder is completed, whereupon it is provided with all of the soft information within the buffer at once. Likewise, the first decoder must remain idle, while the second decoder is active, for the same reason. This motivated the bit-by-bit exchange of soft information in the work of [14]–[17]. More specifically, [14] proposed a decoder for turbo product codes, which performs the decoding of the code's rows in parallel with the decoding of its columns. Soft information is exchanged between the two decoding processes following the completion of each row and each column, rather than following the completion of all rows and all columns, in a serial manner. Similarly, [15], [16] proposed a decoder for turbo convolutional codes, which performs the decoding of both SISO component decoders in parallel. Again, soft information is exchanged between the two components on a bit-by-bit basis, rather than a frame-by-frame basis. This work was

The authors wish to gratefully acknowledge the financial support of the EPSRC, Swindon UK under the auspices of grant EP/J015520/1 and EP/L010550/1, as well as the TSB, Swindon UK under the auspices of grant TS/L009390/1.

further refined in [17], which proposed an interleaver design that mitigates contention during the bit-by-bit exchange of soft information. Each of the above-mentioned approaches was successful in increasing the degree of parallelism of the various turbo-like codes, hence reducing the associated processing time required. However, in each case, this was achieved at the cost of incurring a reduced error correction performance and/or an increased memory or complexity requirement.

Against this background, this paper proposes a technique for expediting the iterative decoding convergence of RA decoders. More specifically, we propose a bit-by-bit iterative decoding algorithm, which allows the repeat decoder and the URC decoder to operate in parallel. In contrast to the bit-by-bit and parallel iterative decoding algorithms mentioned above, ours actually reduces the number of decoding iterations required for achieving convergence, as well as improving the associated error correction performance. This is achieved using a novel technique, which allows the URC decoder to generate soft information at every stage of both its Forward Recursion (FR) and its Backward Recursion (BR), in contrast to the approach of [15], [16]. Furthermore, we enable the repeat decoder to process each piece of soft information as soon as it is generated by the URC decoder, rather than buffering it until the operation of the URC decoder is completed, as in a conventional frame-by-frame RA decoder. Likewise, we provide the URC decoder with each resultant piece of soft information as soon as it is generated by the repeat decoder, potentially to be used by the URC decoder within the same FR or BR. A novel technique is proposed for significantly reducing the complexity of each decoding iteration, which avoids generating soft information that will not be used before it is replaced with more reliable updated soft information. Furthermore, we propose a novel EXtrinsic Information Transfer (EXIT) chart based analysis [18] technique, which demonstrates that each decoding iteration of our proposed bit-by-bit RA decoder is equivalent to two iterations of the conventional frame-by-frame RA decoder.

Note that rather than using the above-described serially concatenated interpretation of RA decoders [1], many previous contributions [19]–[21] have used an alternative factor graph based interpretation, which operates on the basis of message passing [22]. Furthermore, serial update schedules [23]–[28] have been proposed for performing URC decoding within the factor graph using a FR and a BR. This approach has been shown to reduce the required number of decoding iterations by up to 50% [26, Figure 6], compared to the parallel update schedule that is conventionally adopted for message passing. However, these serial update schedules do not benefit from the further complexity reductions that are offered by bit-by-bit decoding, as in the proposed approach. Instead, these schedules wait until the URC decoder's FR and BR have been completed, before operating the repeat decoder on the whole frame at once, in a frame-by-frame manner. Furthermore, these schedules have only been applied to the special case of URC decoders, where the corresponding encoder is a two-state accumulator. This is because accumulators avoid short cycles in the factor graph, which severely degrade the message passing performance [22]. By contrast, the proposed approach imposes no restrictions on the design of the URC decoder.

The rest of this paper is organized as follows. In Section II, we contrast the operation of the proposed bit-by-bit RA decoder to that of a conventional frame-by-frame RA decoder. Section III proposes a technique for reducing the complexity of the proposed bit-by-bit RA decoder, although we show that this still results in a slightly higher decoding complexity per iteration than the conventional frame-by-frame RA decoder. Section IV provides our EXIT chart analysis [18] for characterizing the reduced number of decoding iterations required for achieving convergence in the proposed bit-by-bit RA decoder. Owing to its reduced number of iterations, we show that the proposed bit-by-bit RA decoder has a lower complexity than the conventional frame-by-frame RA decoder, despite having a slightly higher per-iteration complexity. Overall, we demonstrate that in a range of practical scenarios, the proposed bit-by-bit RA offers gains of up to 0.86 dB, without imposing any additional decoding complexity and without requiring any additional transmission-energy, -bandwidth or -duration. Finally, we conclude in Section V that improved RA decoding is facilitated by our novel bit-by-bit iterative decoding algorithm, our novel technique for complexity reduction and our novel EXIT chart analysis technique.

II. Bit-by-Bit Iterative Decoding

This section commences by providing an overview of RA encoding and decoding in Section II-A. Following this, our bit-by-bit RA decoding algorithm is proposed in Section II-B.

A. System Overview

As shown in Figure 1, the RA encoder comprises a repeat encoder, a pseudo-random interleaver $\boldsymbol{\pi}$ [4] and a URC encoder [6]. During the encoding process, each of the L number of bits in the message vector $\mathbf{x} = [x_j]_{j=1}^{L}$ is repeated N number of times by the repeat encoder, which has a coding rate of $R = 1/N$. This results in the vector of LN bits $\mathbf{y} = [y_k]_{k=1}^{LN}$, which are obtained according to $y_k = x_{\lceil k/N \rceil}$. The bit vector $\mathbf{y}$ is permuted by the interleaver $\boldsymbol{\pi} = [\pi_k]_{k=1}^{LN}$ in order to obtain the bit vector of equal length $\mathbf{z} = [z_k]_{k=1}^{LN}$, where $z_{\pi_k} = y_k$. Following this, a URC encoder employing M number of states [6] is used for converting the bit vector $\mathbf{z}$ into the bit vector of equal length $\mathbf{w} = [w_k]_{k=1}^{LN}$, as shown in Figure 1. As it will be justified in Section IV-A, three different RA code parametrizations are considered in this paper, namely the combinations of $R = 1/2$-, $1/3$- and $1/4$-rate repetition codes with $M = 8$-, 4- and 2-state URC codes, respectively. The $M = 8$-, 4- and 2-state URC codes generate the bits of $\mathbf{w}$ according to $w_k = w_{k-3} \oplus w_{k-2} \oplus w_{k-1} \oplus z_k$, $w_k = w_{k-2} \oplus w_{k-1} \oplus z_k$ and $w_k = w_{k-1} \oplus z_k$ respectively, where $\oplus$ represents modulo-2 addition and we assume $w_{k|k<1} = 0$. Note that these $M = 8$- and 4-state URC codes would result in short cycles within a factor graph interpretation of the RA decoder, resulting in poor message passing performance [22]. Figure 1 demonstrates the combination of the $R = 1/3$-rate repetition encoder with the $M = 4$-state URC code, for the case of an $L = 2$-bit message vector and the interleaver $\boldsymbol{\pi} = [2, 4, 1, 6, 5, 3]$.

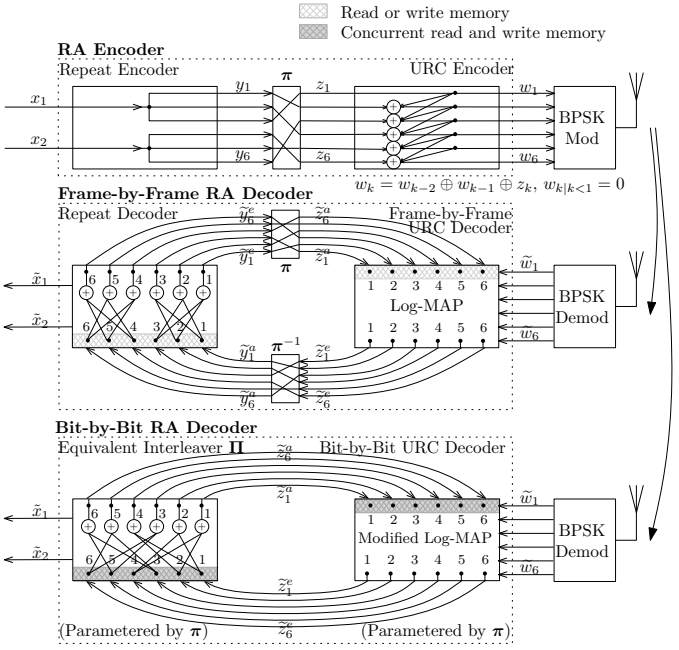


Fig. 1. Schematic of an example RA code having a message length of $L = 2$ bits, a $R = 1/3$-rate repetition code and an $M = 4$-state URC code. The RA encoder and frame-by-frame decoder employ the interleaver $\boldsymbol{\pi} = [2, 4, 1, 6, 5, 3]$, while the proposed bit-by-bit RA decoder employs the corresponding equivalent interleaver $\boldsymbol{\Pi} = [2, 1, 5, 1, 3, 3; 4, 4, 6, 2, 6, 5]$.

We use Binary Phase Shift Keying (BPSK) for modulating the bit vector $\mathbf{w}$ onto an Additive White Gaussian Noise (AWGN) channel. In the receiver, soft decision BPSK demodulation [6] is employed for recovering the vector of Logarithmic Likelihood Ratios (LLRs) $\tilde{\mathbf{w}} = [\tilde{w}_k]_{k=1}^{LN}$, which pertain to the bits of $\mathbf{w}$. As shown in Figure 1, these LLRs are provided as inputs to the Log-MAP URC decoder. In both frame-by-frame and bit-by-bit RA decoders, the Log-MAP URC decoder and the repeat decoder are operated iteratively. In each iteration, the Log-MAP URC decoder uses an FR and a BR for generating the vector of extrinsic LLRs (eLLRs) $\tilde{\mathbf{z}}^e = [\tilde{z}_k^e]_{k=1}^{LN}$ [5], [9], which pertain to the bits of $\mathbf{z}$. The eLLRs in this vector are rearranged by the deinterleaver $\boldsymbol{\pi}^{-1}$, in order to obtain the vector of *a priori* LLRs (aLLRs) $\tilde{\mathbf{y}}^a = [\tilde{y}_k^a]_{k=1}^{LN}$, where $\tilde{y}_k^a = \tilde{z}_{\pi_k}^e$. The repeat decoder adds together particular combinations of the aLLRs in $\tilde{\mathbf{y}}^a$ in order to generate the vector of eLLRs $\tilde{\mathbf{y}}^e = [\tilde{y}_k^e]_{k=1}^{LN}$, which pertain to the bits of $\mathbf{y}$. More specifically, the eLLR $\tilde{y}_k^e$ pertaining to a particular bit y_k is obtained as the sum of the aLLRs that pertain to the *other* repetitions of that bit, according to

$$\tilde{y}_k^e = \sum_{\substack{k' = (\lceil k/N \rceil - 1)N + 1 \\ k' \neq k}}^{\lceil k/N \rceil N} \tilde{y}_{k'}^a. \tag{1}$$

The eLLRs in the vector $\tilde{y}_k^e$ are rearranged by the interleaver $\boldsymbol{\pi}$ in order to obtain the vector of aLLRs $\tilde{\mathbf{z}}^a = [\tilde{z}_k^a]_{k=1}^{LN}$, where $\tilde{z}_{\pi_k}^a = \tilde{y}_k^e$. As shown in Figure 1, these aLLRs are then input to the Log-MAP URC decoder. Once the iterative decoding process has converged, the repeat decoder adds together particular combinations of the aLLRs to produce the vector of *a posteriori* LLRs $\tilde{\mathbf{x}} = [\tilde{x}_j]_{j=1}^{L}$, which pertain to the bits of $\mathbf{x}$. More specifically, the *a posteriori* LLR $\tilde{x}_j$ pertaining to a particular bit x_k is obtained as the sum of the aLLRs that pertain to *all* repetitions of that bit, according to

$$\tilde{x}_j = \sum_{k'=(j-1)N+1}^{jN} \tilde{y}_{k'}^a. \tag{2}$$

Note that in the bit-by-bit decoder, the deinterleaver $\boldsymbol{\pi}^{-1}$, the repeat decoder and the interleaver $\boldsymbol{\pi}$ are amalgamated in order to form an equivalent interleaver $\boldsymbol{\Pi}$, as will be detailed in Section II-B.

B. Bit-by-Bit RA Decoding Algorithm

In contrast to the alternated operation of the two SISO component decoders in the frame-by-frame RA decoder, the proposed bit-by-bit RA decoder enables both component decoders to operate simultaneously. This is facilitated by using concurrent read and write memory, the equivalent interleaver $\boldsymbol{\Pi}$ and an bit-by-bit Log-MAP URC decoder, as discussed in the following subsections. These discussions are aided by the timing diagrams of Figures 2 and 3, which exemplify the values that are input, stored, loaded and output by each component during one iteration of the frame-by-frame and bit-by-bit RA decoders of Figure 1, respectively. Note that in the frame-by-frame RA decoder timing diagram of Figure 2, the Log-MAP URC decoder is active in the first eleven Time Slots (TSs), while the repeat decoder is active in the twelfth TS, when it performs all processing of (1) in parallel. By contrast, in the bit-by-bit RA decoder timing diagram of Figure 3, the Log-MAP URC decoder and the equivalent interleaver $\boldsymbol{\Pi}$ are both active in all ten TSs.

	Time	1	2	3	4	5	6	7	8	9	10	11	12
Frame-by-frame URC	Input	$\tilde{z}_1^a$-$\tilde{z}_6^a$											
	Store	$\tilde{z}_2^a$-$\tilde{z}_6^a$											
	Load	$\tilde{z}_1^a$	$\tilde{z}_2^a$	$\tilde{z}_3^a$	$\tilde{z}_4^a$	$\tilde{z}_5^a$	$\tilde{z}_6^a$	$\tilde{z}_5^a$	$\tilde{z}_4^a$	$\tilde{z}_3^a$	$\tilde{z}_2^a$	$\tilde{z}_1^a$	
	Load	$\tilde{w}_1$ $\boldsymbol{\alpha}_0$	$\tilde{w}_2$ $\boldsymbol{\alpha}_1$	$\tilde{w}_3$ $\boldsymbol{\alpha}_2$	$\tilde{w}_4$ $\boldsymbol{\alpha}_3$	$\tilde{w}_5$ $\boldsymbol{\alpha}_4$	$\tilde{w}_6$ $\boldsymbol{\alpha}_5$ $\boldsymbol{\beta}_6$	$\tilde{w}_5$ $\boldsymbol{\alpha}_4$ $\boldsymbol{\beta}_5$	$\tilde{w}_4$ $\boldsymbol{\alpha}_3$ $\boldsymbol{\beta}_4$	$\tilde{w}_3$ $\boldsymbol{\alpha}_2$ $\boldsymbol{\beta}_3$	$\tilde{w}_2$ $\boldsymbol{\alpha}_1$ $\boldsymbol{\beta}_2$	$\tilde{w}_1$ $\boldsymbol{\alpha}_0$ $\boldsymbol{\beta}_1$	
	Store	$\boldsymbol{\alpha}_1$	$\boldsymbol{\alpha}_2$	$\boldsymbol{\alpha}_3$	$\boldsymbol{\alpha}_4$	$\boldsymbol{\alpha}_5$	$\boldsymbol{\beta}_5$	$\boldsymbol{\beta}_4$	$\boldsymbol{\beta}_3$	$\boldsymbol{\beta}_2$	$\boldsymbol{\beta}_1$		
	Output						$\tilde{z}_6^e$	$\tilde{z}_5^e$	$\tilde{z}_4^e$	$\tilde{z}_3^e$	$\tilde{z}_2^e$	$\tilde{z}_1^e$	
$\boldsymbol{\pi}^{-1}$	Input						$\tilde{z}_6^e$	$\tilde{z}_5^e$	$\tilde{z}_4^e$	$\tilde{z}_3^e$	$\tilde{z}_2^e$	$\tilde{z}_1^e$	
	Output						$\tilde{y}_4^a$	$\tilde{y}_5^a$	$\tilde{y}_2^a$	$\tilde{y}_6^a$	$\tilde{y}_1^a$	$\tilde{y}_3^a$	
Repeat	Input						$\tilde{y}_4^a$	$\tilde{y}_5^a$	$\tilde{y}_2^a$	$\tilde{y}_6^a$	$\tilde{y}_1^a$	$\tilde{y}_3^a$	
	Store						$\tilde{y}_4^a$	$\tilde{y}_5^a$	$\tilde{y}_2^a$	$\tilde{y}_6^a$	$\tilde{y}_1^a$	$\tilde{y}_3^a$	
	Load												$\tilde{y}_1^a$-$\tilde{y}_6^a$
	Output												$\tilde{y}_1^e$-$\tilde{y}_6^e$
$\boldsymbol{\pi}$	Input												$\tilde{y}_1^e$-$\tilde{y}_6^e$
	Output												$\tilde{z}_1^a$-$\tilde{z}_6^a$

Fig. 2. Timing diagram for each decoding iteration of the frame-by-frame RA decoder shown in Figure 1.

1) Concurrent Read and Write Memory: In an RA decoder, each SISO component decoder uses a memory to store LLRs until its decoding algorithm is ready to use them. In the frame-by-frame RA decoder of Figure 1, only one SISO component decoder is active at a time, as exemplified in Figure 2. During this time, the active SISO component decoder reads aLLRs

	Time	1	2	3	4	5	6	7	8	9	10
Bit-by-bit URC	Input	$\tilde{z}_1^a$	$\tilde{z}_2^a$	$\tilde{z}_4^a$	$\tilde{z}_5^a$		$\tilde{z}_6^a$	$\tilde{z}_5^a$	$\tilde{z}_3^a$	$\tilde{z}_2^a$	
	Store	$\tilde{z}_1^a$	$\tilde{z}_2^a$	$\tilde{z}_4^a$	$\tilde{z}_5^a$		$\tilde{z}_6^a$	$\tilde{z}_5^a$	$\tilde{z}_3^a$	$\tilde{z}_2^a$	
	Load	$\tilde{z}_1^a$	$\tilde{z}_2^a$	$\tilde{z}_3^a$	$\tilde{z}_4^a$	$\tilde{z}_5^a$	$\tilde{z}_6^a$	$\tilde{z}_5^a$	$\tilde{z}_4^a$	$\tilde{z}_3^a$	$\tilde{z}_2^a$
	Load	$\widetilde{w}_1$	$\widetilde{w}_2$	$\widetilde{w}_3$	$\widetilde{w}_4$	$\widetilde{w}_5$	$\widetilde{w}_6$	$\widetilde{w}_5$	$\widetilde{w}_4$	$\widetilde{w}_3$	$\widetilde{w}_2$
		$\boldsymbol{\alpha}_0$	$\boldsymbol{\alpha}_1$	$\boldsymbol{\alpha}_2$	$\boldsymbol{\alpha}_3$	$\boldsymbol{\alpha}_4$	$\boldsymbol{\alpha}_5$	$\boldsymbol{\alpha}_4$	$\boldsymbol{\alpha}_3$		$\boldsymbol{\alpha}_1$
		$\boldsymbol{\beta}_1$	$\boldsymbol{\beta}_2$	$\boldsymbol{\beta}_3$		$\boldsymbol{\beta}_5$	$\boldsymbol{\beta}_6$	$\boldsymbol{\beta}_5$	$\boldsymbol{\beta}_4$	$\boldsymbol{\beta}_3$	$\boldsymbol{\beta}_2$
	Store	$\boldsymbol{\alpha}_1$	$\boldsymbol{\alpha}_2$	$\boldsymbol{\alpha}_3$	$\boldsymbol{\alpha}_4$	$\boldsymbol{\alpha}_5$	$\boldsymbol{\beta}_5$	$\boldsymbol{\beta}_4$	$\boldsymbol{\beta}_3$	$\boldsymbol{\beta}_2$	$\boldsymbol{\beta}_1$
	Output	$\tilde{z}_1^e$	$\tilde{z}_2^e$	$\tilde{z}_3^e$		$\tilde{z}_5^e$	$\tilde{z}_6^e$	$\tilde{z}_5^e$	$\tilde{z}_4^e$		$\tilde{z}_2^e$
$\boldsymbol{\Pi}$	Input	$\tilde{z}_1^e$	$\tilde{z}_2^e$	$\tilde{z}_3^e$		$\tilde{z}_5^e$	$\tilde{z}_6^e$	$\tilde{z}_5^e$	$\tilde{z}_4^e$		$\tilde{z}_2^e$
	Store	$\tilde{z}_1^e$	$\tilde{z}_2^e$	$\tilde{z}_3^e$		$\tilde{z}_5^e$	$\tilde{z}_6^e$	$\tilde{z}_5^e$	$\tilde{z}_4^e$		$\tilde{z}_2^e$
	Load	$\tilde{z}_1^e$	$\tilde{z}_1^e$	$\tilde{z}_3^e$		$\tilde{z}_3^e$	$\tilde{z}_3^e$	$\tilde{z}_5^e$	$\tilde{z}_1^e$		$\tilde{z}_2^e$
		$\tilde{z}_4^e$	$\tilde{z}_2^e$	$\tilde{z}_6^e$		$\tilde{z}_5^e$	$\tilde{z}_6^e$	$\tilde{z}_6^e$	$\tilde{z}_4^e$		$\tilde{z}_4^e$
	Output	$\tilde{z}_2^a$	$\tilde{z}_4^a$	$\tilde{z}_5^a$		$\tilde{z}_6^a$	$\tilde{z}_5^a$	$\tilde{z}_3^a$	$\tilde{z}_2^a$		$\tilde{z}_1^a$

Fig. 3. Timing diagram for each decoding iteration of the bit-by-bit RA decoder shown in Figure 1.

from its memory, as required by its decoding algorithm. In response, it generates eLLRs, which are interleaved and written to the memory of the other SISO component decoder, as shown in Figure 1. Once the operation of the first SISO component decoder is completed, the other decoder becomes active and the roles of the memories are swapped. Owing to this, the memories in the frame-by-frame RA decoder are never read and written at the same time, allowing *read or write memory* to be used. This is exemplified in Figure 2, where the aLLRs are never stored in a particular component's memory during a TS, in which aLLRs are read from that memory.

By contrast, both SISO component decoders are active all of the time in the bit-by-bit RA decoder of Figure 1, requiring both decoders to use *concurrent read and write memory*. More specifically, throughout the bit-by-bit RA decoding process, one SISO component decoder reads the aLLRs from its own memory and writes the eLLRs into the memory of the other SISO component decoder. Meanwhile, the other SISO component decoder will be doing the same, causing both memories to be read and written concurrently. This is exemplified in Figure 3, where the aLLRs are frequently stored in a particular component's memory during a TS, in which the aLLRs are read from that memory. Note that at the beginning of iterative decoding, the concurrent read and write memories of both SISO component decoders are filled with zero-valued LLRs.

2) Equivalent Interleaver: In the proposed bit-by-bit RA decoder, the deinterleaver $\boldsymbol{\pi}^{-1}$, the repeat decoder and the interleaver $\boldsymbol{\pi}$ are combined in order to form an equivalent interleaver $\boldsymbol{\Pi}$, as mentioned above. In contrast to a stand-alone repeat decoder, the equivalent interleaver is aware of where each aLLR is positioned within the vector $\tilde{\mathbf{z}}^a$. As it will be described below, this knowledge may be exploited in order to reduce the complexity of the equivalent interleaver.

The equivalent interleaver is specifically configured to maintain the relationships that are created between the aLLRs of $\tilde{\mathbf{z}}^a$ and the eLLRs of $\tilde{\mathbf{z}}^e$, when employing a separate deinterleaver, repeat decoder and interleaver. More specifically, each aLLR in the vector $\tilde{\mathbf{z}}^a$ is obtained by deinterleaving, adding and interleaving $(N-1)$ of the eLLRs in the vector $\tilde{\mathbf{z}}^e$, as exemplified in Figure 1. Note that the particular set of $(N-1)$ eLLRs depends on the design of the interleaver $\boldsymbol{\pi}$. Conversely, each eLLR in the vector $\tilde{\mathbf{z}}^e$ contributes to the summations that provide $(N-1)$ of the aLLRs in the vector $\tilde{\mathbf{z}}^a$, again depending on the design of the interleaver $\boldsymbol{\pi}$. For example, both the frame-by-frame and bit-by-bit RA decoders of Figure 1 are configured for ensuring that the first aLLR $\tilde{z}_1^a$ is obtained as the sum of the $(N-1)=2$ eLLRs $\tilde{z}_2^e$ and $\tilde{z}_4^e$. Conversely, the first eLLR $\tilde{z}_1^e$ contributes to the summations that provide the $(N-1)=2$ aLLRs $\tilde{z}_2^a$ and $\tilde{z}_4^a$.

The configuration of the equivalent interleaver may be described by a matrix $\boldsymbol{\Pi}$, having $(N-1)$ rows and LN columns. The k^{th} column of the matrix lists the indices of the eLLRs that are added together in order to obtain the k^{th} aLLR $\tilde{z}_k^a$, according to

$$\tilde{z}_k^a = \sum_{n=1}^{N-1} \tilde{z}_{\Pi_{n,k}}^e. \tag{3}$$

For example, the equivalent interleaver of Figure 1 is configured according to $\boldsymbol{\Pi} = \begin{bmatrix} 2,1,5,1,3,3 \\ 4,4,6,2,6,5 \end{bmatrix}$, where the first column contains the indices 2 and 4 because the first aLLR is obtained according to $\tilde{z}_1^a = \tilde{z}_2^e + \tilde{z}_4^e$. Equivalently, the k^{th} column of the matrix $\boldsymbol{\Pi}$ can be considered to list the indices of the aLLRs that are contributed to by the eLLR $\tilde{z}_k^e$. For example, the first column of $\boldsymbol{\Pi}$ in Figure 1 contains the indices 2 and 4, because the first eLLR $\tilde{z}_1^e$ contributes to the aLLRs $\tilde{z}_2^a$ and $\tilde{z}_4^a$. Note that the calculation of (3) is reminiscent of the operation of a variable node in a factor graph interpretation of the RA code. More specifically, the k^{th} aLLR $\tilde{z}_k^a$ is obtained by summing all of the relevant eLLRs, except for the corresponding one $\tilde{z}_k^e$ [22].

In the proposed bit-by-bit RA decoder, the equivalent interleaver $\boldsymbol{\Pi}$ operates in a reactive manner. More explicitly, whenever an updated eLLR value $\tilde{z}_k^e$ is received, the equivalent interleaver stores it in its concurrent read and write memory. At the same time, the $(N-1)$ eLLRs with the indices in the k^{th} column of $\boldsymbol{\Pi}$ may be loaded from the concurrent read and write memory. Together with the eLLR $\tilde{z}_k^e$, these $(N-1)$ eLLRs allow the computation of new values for the $(N-1)$ aLLRs having the indices in the k^{th} column of $\boldsymbol{\Pi}$. These $(N-1)$ aLLRs may then be input to the Log-MAP URC decoder and stored in its concurrent read and write memory. In the example of Figure 1, whenever an updated value is received for the first eLLR $\tilde{z}_1^e$, the eLLRs $\tilde{z}_2^e$ and $\tilde{z}_4^e$ may be loaded from the concurrent read and write memory. These eLLRs may be used for producing updated values for the aLLRs $\tilde{z}_2^a = \tilde{z}_1^e + \tilde{z}_4^e$ and $\tilde{z}_4^a = \tilde{z}_1^e + \tilde{z}_2^e$.

However, in the bit-by-bit RA decoder, the eLLRs $\tilde{z}_k^e$ are generated in a particular order by the Log-MAP URC decoder, which is dictated by its FR and BR, as it will be described in Section II-B3. Likewise, the aLLRs $\tilde{z}_k^a$ are read from the Log-MAP URC decoder's concurrent read and write memory in that same order. Owing to this, when the equivalent interleaver receives an eLLR, it is unnecessary for it to update all $(N-1)$ of the corresponding aLLRs, because some of these will be replaced before they are used by the Log-MAP URC decoder.

In the example of Figure 3, the Log-MAP URC decoder loads the aLLR $\tilde{z}_1^a$ in TS 1, it loads $\tilde{z}_2^a$ in TSs 2 and 10, and it loads $\tilde{z}_4^a$ in TSs 4 and 8. Likewise, it outputs an updated value for the eLLR $\tilde{z}_1^e$ in TS 1, it outputs $\tilde{z}_2^e$ in TSs 2 and 10, and it outputs $\tilde{z}_4^e$ in TS 8, as shown in Figure 3. During TS 1, the equivalent interleaver uses $\tilde{z}_1^e$ to generate only $\tilde{z}_2^a$, without generating $\tilde{z}_4^a$. This is because $\tilde{z}_4^a$ will not be used by the Log-MAP URC decoder until TS 4 and because in TS 2, the equivalent interleaver uses $\tilde{z}_2^e$ to generate a superior value for $\tilde{z}_4^a$. Likewise, $\tilde{z}_1^a$ is not generated during TS 2, because it will not be used by the Log-MAP URC decoder until TS 1 of the next decoding iteration. In this way, the complexity associated with generating unnecessary aLLRs can be eliminated from the equivalent interleaver. As mentioned above, this complexity reduction is facilitated by informing the equivalent interleaver of where each aLLR that it generates is positioned within the vector $\tilde{\mathbf{z}}^a$, as well as by informing the equivalent interleaver of the order in which the aLLRs are used by the Log-MAP URC decoder.

3) Bit-by-Bit Log-MAP URC Decoder: Similarly to the Log-MAP URC decoder of a conventional frame-by-frame RA decoder, the proposed bit-by-bit RA decoder operates on the basis of an FR and a BR. More specifically, an FR is performed during the first half of each decoding iteration, which corresponds to TSs 1 to 5 in the examples of Figures 2 and 3. During this time, the Log-MAP URC decoder loads and processes a single aLLR $\tilde{z}_k^a$ and a single demodulated LLR $\tilde{w}_k$ in each TS, in ascending order of the index k, as exemplified in Figures 2 and 3. During each TS, a vector $\boldsymbol{\alpha}_{k-1}$ of M forward state metrics is loaded from memory and combined with the LLRs $\tilde{z}_k^a$ and $\tilde{w}_k$ to produce a new vector $\boldsymbol{\alpha}_k$ of M forward state metrics [11], which is stored in memory. Note that these data dependencies of $\boldsymbol{\alpha}_k$ upon $\boldsymbol{\alpha}_{k-1}$ dictate the requirement for the forward state metrics to be calculated using an FR. Likewise, a BR is performed during the second half of each decoding iteration, when the LLRs are loaded and processed in descending order of index k, as exemplified in TSs 6 to 10 of Figures 2 and 3. Each step of the BR has to calculate and store the vector $\boldsymbol{\beta}_{k-1}$ of M backward state metrics, which are obtained by combining $\boldsymbol{\beta}_k$ with $\tilde{z}_k^a$ and $\tilde{w}_k$. Note that depending on how the URC code is terminated, $\boldsymbol{\alpha}_0$ and $\boldsymbol{\beta}_{LN}$ are initialized with values that remain constant throughout the iterative decoding process [11].

In the frame-by-frame RA decoder, the Log-MAP URC decoder is only able to generate and output eLLRs during the BR, as shown in Figure 2. This is because each eLLR $\tilde{z}_k^e$ is obtained by loading and combining $\tilde{w}_k$, $\boldsymbol{\alpha}_{k-1}$ and $\boldsymbol{\beta}_k$ [11], but the backward state metrics $\boldsymbol{\beta}_k$ are not available until the BR is started. By contrast, the Log-MAP URC decoder of the proposed bit-by-bit RA decoder is able to generate and output eLLRs during both the FR and the BR, as shown in Figure 3. This is made possible in the FR because we propose to generate the eLLR $\tilde{z}_k^e$ by loading and combining $\tilde{w}_k$ and $\boldsymbol{\alpha}_{k-1}$ with the backward state metrics $\boldsymbol{\beta}_k$ that were generated and stored during the previous decoding iteration. Note that before iterative decoding begins, all backward state metrics are initialized with zero values.

Note that in the proposed bit-by-bit RA decoder, the Log-MAP URC decoder has the opportunity to generate and output each eLLR $\tilde{z}_k^e$ during each FR and during each BR. Therefore, it can be said that the eLLR vector $\tilde{\mathbf{z}}^e$ is updated in each half-iteration of the bit-by-bit RA decoder, where alternate half-iterations comprise a FR or a BR. Likewise, since the equivalent interleaver operates in a reactive manner, generating aLLRs as soon as it is provided with eLLRs, it can be said that the aLLR vector $\tilde{\mathbf{z}}^a$ is also updated in each half-iteration. By contrast, the vectors $\tilde{\mathbf{z}}^e$ and $\tilde{\mathbf{z}}^a$ are only updated once per iteration in the frame-by-frame RA decoder. Therefore, each bit-by-bit half-iteration may be deemed to correspond to one frame-by-frame iteration and hence we surmise that each bit-by-bit iteration corresponds to two frame-by-frame iterations. These relationships will be demonstrated with the aid of our EXIT chart analysis in Section IV-A.

Note that as in the equivalent interleaver $\boldsymbol{\Pi}$ of Section II-B2, the Log-MAP URC decoder can facilitate a complexity reduction by exploiting the knowledge of the interleaver design $\boldsymbol{\pi}$. More specifically, it is unnecessary for the Log-MAP URC decoder to generate a particular eLLR $\tilde{z}_k^e$ during its FR, if it would be superseded by the updated version of that eLLR $\tilde{z}_k^e$ generated during the BR. This happens when the corresponding aLLRs generated by the equivalent interleaver $\boldsymbol{\Pi}$ in response to $\tilde{z}_k^e$ would not be used by the Log-MAP URC decoder before they are replaced in response to the generation of $\tilde{z}_k^e$ during the BR. Likewise, it is unnecessary for the Log-MAP URC decoder to generate a particular eLLR $\tilde{z}_k^e$ during its BR, if it would be superseded by the eLLR $\tilde{z}_k^e$ generated during the FR of the next decoding iteration. For example, in TS 4 of Figure 3, it is unnecessary for the Log-MAP URC decoder to generate the eLLR $\tilde{z}_4^e$. This is because this eLLR would allow the equivalent interleaver $\boldsymbol{\Pi}$ to update the aLLRs $\tilde{z}_2^a$ and $\tilde{z}_1^a$, as described in Section II-B2. However, these aLLRs would not be used by the Log-MAP URC decoder until the end of its BR, in TS 10 and TS 1 of the next decoding iteration, respectively. However before this, the Log-MAP URC decoder has a second opportunity to update $\tilde{z}_4^e$ in TS 8. It is this version of the eLLR $\tilde{z}_4^e$ that is used by the equivalent interleaver $\boldsymbol{\Pi}$ to update the aLLRs $\tilde{z}_1^a$ and $\tilde{z}_2^a$, ready for TSs 10 and 1. Note that since there is no need to generate $\tilde{z}_4^e$ during TS 4, there is also no need to load $\boldsymbol{\beta}_4$ for that purpose during that TS. Furthermore, since no eLLR is output during TS 4, the equivalent interleaver $\boldsymbol{\Pi}$ is not operated during that TS and there is no new input for the Log-MAP URC decoder to store in TS 5, as shown in Figure 3. Likewise, there is no need for the Log-MAP URC decoder to generate the eLLR $\tilde{z}_3^e$ during TS 9 because it is superseded by the $\tilde{z}_3^e$ generated during TS 3 of the next decoding iteration.

III. Complexity Analysis

In this section, we characterize and compare the computational complexity per iteration of the frame-by-frame and bit-by-bit RA decoders. This analysis will be used for making fair comparisons between the frame-by-frame and bit-by-bit RA decoders in Section IV. As described in Section II-B, a complexity reduction is facilitated for the proposed bit-by-bit RA decoder, if the equivalent interleaver $\boldsymbol{\Pi}$ and Log-MAP URC decoder are able to exploit knowledge of the interleaver

design $\boldsymbol{\pi}$. Owing to this, it is not necessary to generate all aLLRs and eLLRs during the operation of the bit-by-bit RA decoder. We commence in Section III-A by characterizing the average number of LLRs are generated during each TS of each decoding iteration. Then, Section III-B quantifies and compares the computational complexity of the frame-by-frame and bit-by-bit RA decoders in terms of the number of Add, Compare and Select (ACS) operations [9] that they perform in each decoding iteration.

A. Complexity Reduction

As described in Section II-B2, whenever an eLLR $\tilde{z}_k^e$ is provided for the equivalent interleaver $\boldsymbol{\Pi}$, it has the opportunity to update the $(N-1)$ aLLRs having the indices in the k^{th} column of $\boldsymbol{\Pi}$. However, it is sufficient for the equivalent interleaver $\boldsymbol{\Pi}$ to update at most one of these aLLRs, namely that specific $\tilde{z}_{k'}^a$ which will be used first by the Log-MAP URC decoder. This is because the updated $\tilde{z}_{k'}^a$ will lead to better opportunities to update the other $(N-2)$ aLLRs, before they are used by the Log-MAP URC decoder. More specifically, the other $(N-2)$ indices in the k^{th} column of $\boldsymbol{\Pi}$ also appear in the k'^{th} column, owing to the symmetry within the equivalent interleaver. Owing to this, when $\tilde{z}_{k'}^a$ is used by the Log-MAP URC decoder, it will input the corresponding eLLR $\tilde{z}_{k'}^e$ to the equivalent interleaver $\boldsymbol{\Pi}$, granting it the opportunity to update the other $(N-2)$ aLLRs. Note that owing to the symmetry of the equivalent interleaver, this approach will never miss an opportunity to update an aLLR that is used by the Log-MAP URC decoder and hence it does not compromise the error correction capability of the bit-by-bit RA decoder. In the example of Figure 3, the equivalent interleaver $\boldsymbol{\Pi}$ is provided with $\tilde{z}_1^e$ in TS 1, giving it the opportunity to update $\tilde{z}_2^a$ and $\tilde{z}_4^a$. Of these, it is $\tilde{z}_2^a$ that is used first by the Log-MAP URC decoder, in TS 2. In response, the Log-MAP URC decoder provides $\tilde{z}_2^e$ to the equivalent interleaver $\boldsymbol{\Pi}$, giving it a better opportunity to update $\tilde{z}_4^a$, ready to be used by the Log-MAP URC decoder in TS 4. Therefore, even though the opportunity to update $\tilde{z}_4^a$ was not exploited in TS 1, the error correction capability of the bit-by-bit RA decoder is not compromised, since $\tilde{z}_4^a$ is updated in TS 2, ready to be used in TS 4.

Like the equivalent interleaver $\boldsymbol{\Pi}$, the Log-MAP URC decoder has the opportunity to generate an eLLR in each TS of each decoding iteration. However, in some TSs it is not necessary for the Log-MAP URC decoder to generate any eLLRs. In these TSs it is therefore also unnecessary for the equivalent interleaver $\boldsymbol{\Pi}$ to generate any aLLRs, since there is no new extrinsic information to base these on. This situation happens when the Log-MAP URC decoder has the opportunity to generate an eLLR $\tilde{z}_k^e$, but all $(N-1)$ of the indices in the k^{th} column of $\boldsymbol{\Pi}$ are situated behind the index k in the direction of the current recursion. In this case, none of the corresponding $(N-1)$ aLLRs would be used by the Log-MAP URC decoder before the current recursion is completed. Nor would they be used before the eLLR $\tilde{z}_k^e$ is updated again during the opposite recursion, when there is a better opportunity to update these aLLRs. This explains why the Log-MAP URC decoder does not exploit the opportunity to update $\tilde{z}_4^e$ in TS 4 of Figure 3 and why the equivalent interleaver $\boldsymbol{\Pi}$ does not update any aLLRs in this TS, as discussed in Section II-B2. More specifically, this is because the corresponding aLLRs $\tilde{z}_1^a$ and $\tilde{z}_2^a$ have indices that are behind that of $\tilde{z}_4^e$ during the FR.

Based on these observations, we can quantify the average number of LLRs that the equivalent interleaver $\boldsymbol{\Pi}$ and the Log-MAP URC decoder generates during each TS of each decoding iteration. Since the Log-MAP URC decoder and the equivalent interleaver $\boldsymbol{\Pi}$ will each generate either one or zero LLRs during each TS, the average number is given by the fraction of TSs in which one eLLR and one aLLR are generated. During the FR, the k^{th} column of $\boldsymbol{\Pi}$ is associated with the generation of an eLLR and an aLLR, if at least one of its $(N-1)$ elements has a higher value than k. Therefore, the specific fraction of columns in $\boldsymbol{\Pi}$ that are of this type gives the average number of eLLRs and aLLRs that are generated during each TS of the FR. When employing a random interleaver design $\boldsymbol{\pi}$, the expected value for this fraction is given by

$$P_a = \frac{N-1}{N}\left(1 - \frac{1}{LN}\right), \tag{4}$$

as derived in Appendix A. Owing to the symmetry of the situation, the corresponding fraction for the BR will have the same value. Therefore, P_a quantifies the average number of eLLRs that the Log-MAP URC decoder generates during each TS of each decoding iteration, as well as the average number of aLLRs that the equivalent interleaver $\boldsymbol{\Pi}$ generates. Note that we have $P_a \rightarrow \frac{N-1}{N}$ for long messages, where $L \rightarrow \infty$.

B. Add, Compare and Select Operations

The computational complexity of the frame-by-frame and bit-by-bit RA decoders can be quantified using the number of operations that are performed during each decoding iteration. Both decoders operate solely on the basis of additions, subtractions and $\max^*$ operations [9], where

$$\max{}^*(a,b) = \max(a,b) + \ln(1 + e^{-|a-b|}). \tag{5}$$

The logarithmic term on the right hand side of (5) may be approximated using an 8-entry Look-Up-Table (LUT) [9]. In this case, each $\max^*$ operation can be considered to comprise five ACS operations [9]. More specifically, a first ACS operation may be employed for simultaneously calculating $\max(a,b)$ and $-|a-b|$. Three ACS operations may be employed to logarithmically decompose the LUT and then to select the best approximation for $\ln(1+e^{-|a-b|})$. Finally, a fifth ACS operation may be employed to add this to $\max(a,b)$. Similarly, each addition and each subtraction employed by the frame-by-frame and bit-by-bit RA decoder may be considered to comprise a single ACS operation.

Table I quantifies the computational complexity of the bit-by-bit and frame-by-frame RA decoders per message bit per decoding iteration, when employing the particular combinations of R-rate repetition codes and M-state URC codes, as it will be justified in Section IV. The quantities of Table I were obtained by counting the number of operations performed per message bit per iteration in optimized implementations of the bit-by-bit and frame-by-frame RA decoders. Note that for each

combination of R and M, the bit-by-bit RA decoder requires more ACS operations than the frame-by-frame RA decoder. This is because in the bit-by-bit RA decoder, the equivalent interleaver $\boldsymbol{\Pi}$ and the Log-MAP URC decoder each generate an average of around $(N-1)/N$ LLRs in each TS, where $N = 1/R$. By contrast, in the frame-by-frame RA decoder, both the repeat and the URC decoder each generate an average of only about 1/2 an LLR in each TS. This is because around half of the TSs are dedicated to performing the Log-MAP URC decoder's FR, which does not generate any eLLRs, as exemplified in Figure 2. Despite its higher complexity per message bit per iteration, we will show in Section IV that the bit-by-bit RA decoder has a lower complexity overall, since it requires fewer decoding iterations to converge.

TABLE I
COMPUTATIONAL COMPLEXITY PER MESSAGE BIT PER DECODING ITERATION FOR BIT-BY-BIT AND FRAME-BY-FRAME RA DECODERS, WHEN EMPLOYING R-RATE REPETITION CODES AND M-STATE URC CODES.

Decoder	R	M	$+$ or $-$	$\max^*$	ACS
Bit-by-Bit	1/2	8	102	60	402
	1/3	4	98	48	338
	1/4	2	86	28	226
Frame-by-Frame	1/2	8	100	60	400
	1/3	4	81	42	291
	1/4	2	64	24	184

IV. RESULTS

In this section, the EXIT charts [6] of frame-by-frame and bit-by-bit RA decoders are characterized and compared in Section IV-A. Then, the Bit Error Ratio (BER) performances of these schemes are compared in Section IV-B.

A. EXIT Charts

EXIT charts [6] may be used for characterizing the iterative decoding convergence of iterative decoders, considering the evolution of the Mutual Information (MI) $I(\tilde{\mathbf{z}}^a; \mathbf{z})$ and $I(\tilde{\mathbf{z}}^e; \mathbf{z})$ of the aLLRs and eLLRs, respectively. The EXIT chart of the frame-by-frame RA decoder may be obtained in the conventional manner, by plotting the inverted EXIT function of the repeat decoder together with the EXIT function of the Log-MAP URC decoder, as shown in Figures 4(a)-(e). Here, Figure 4(a) shows that the $R = 1/2$-rate repetition code's EXIT function has a complementary shape to that of the $M = 8$-state frame-by-frame URC code described in Section II-A. This allows the creation of an open EXIT chart tunnel at low E_b/N_0 values, which facilitates iterative decoding convergence to a low BER. Likewise, Figures 4(b) and (d) show that the $R = 1/3$- and $R = 1/4$-rate repetition codes have EXIT functions that complement those of the $M = 4$- and $M = 2$-state frame-by-frame URC codes, respectively. Indeed, our experiments revealed that other combinations of $R = 1/2$, $1/3$, and $1/4$ with $M = 2$, 4 and 8 offered inferior matches between the repetition and URC EXIT functions, requiring higher E_b/N_0 values to create open EXIT chart tunnels. This motivates the selected combinations of R and M, as described in Section II-A. Figures 4(a)-(e) show that the frame-by-frame RA decoder's iterative decoding trajectory offers a perfect match with the corresponding EXIT functions, for all the parametrizations and E_b/N_0 values considered, as may be expected. Note that these trajectories were obtained using a message length of $L = 10^6$, in order to eliminate the variation that occurs from frame-to-frame, when employing shorter message lengths L.

By contrast, the EXIT functions and trajectories of the bit-by-bit RA decoder cannot be obtained in the conventional manner, owing to its unique operation. Instead, the bit-by-bit RA decoder's iterative decoding trajectory is obtained by measuring the MIs $I(\tilde{\mathbf{z}}^a; \mathbf{z})$ and $I(\tilde{\mathbf{z}}^e; \mathbf{z})$ simultaneously, following the completion of the BR at the end of each decoding iteration. While the iterative decoding trajectory of the bit-by-bit RA decoder can be seen to offer a perfect match with the repetition code's EXIT function, it consistently and significantly overshoots that of the frame-by-frame URC code, as shown in Figures 4(a)-(e). This may be explained by the observation that each bit-by-bit decoding iteration can be thought of as being approximately equivalent to two frame-by-frame decoding iterations, as described in Section II-B3. This observation is supported by the so-called 'half-iteration' trajectories, which are also plotted in Figures 4(a)-(e) for the bit-by-bit RA decoder. These half-iteration trajectories were obtained by measuring the MIs $I(\tilde{\mathbf{z}}^a; \mathbf{z})$ and $I(\tilde{\mathbf{z}}^e; \mathbf{z})$ simultaneously, following the completion of not only each BR, but also each FR. Accordingly, it may be observed that each pair of steps in the half-iteration trajectory reaches the same point in the EXIT chart as a single step in the bit-by-bit RA decoder trajectory. However, it may also be observed that this half-iteration trajectory offers a good match with the EXIT functions of both the repetition code and the frame-by-frame URC code. This confirms that each bit-by-bit half iteration is equivalent to a single frame-by-frame iteration and hence that each bit-by-bit iteration is equivalent to two frame-by-frame iterations. Owing to this, Figures 4(a)-(e) show that the proposed bit-by-bit RA decoder is capable of achieving iterative decoding convergence using significantly fewer decoding iterations than that required by the frame-by-frame RA benchmarker, in all the scenarios considered. This reduction in the required number of decoding iterations more than compensates for the bit-by-bit RA decoder's slightly increased complexity per decoding iteration, hence offering a significantly reduced overall complexity, as described in Section III-B.

The EXIT function of the bit-by-bit URC code cannot be plotted in the conventional manner, since this overlooks the memory that is maintained between the operation of the bit-by-bit Log-MAP URC decoder in successive decoding iterations. While the outputs of the conventional frame-by-frame Log-MAP URC decoder depend only on its inputs, those of the bit-by-bit Log-MAP URC decoder additionally depend also on the results calculated during previous decoding iterations and stored in memory. For example, during the bit-by-bit Log-MAP URC decoder's FR, the backward state metrics calculated during the previous decoding iteration are loaded from memory, as described in Section II-B3. Figures 4(a)-(e) provide predictions for the EXIT function of the bit-by-bit RA

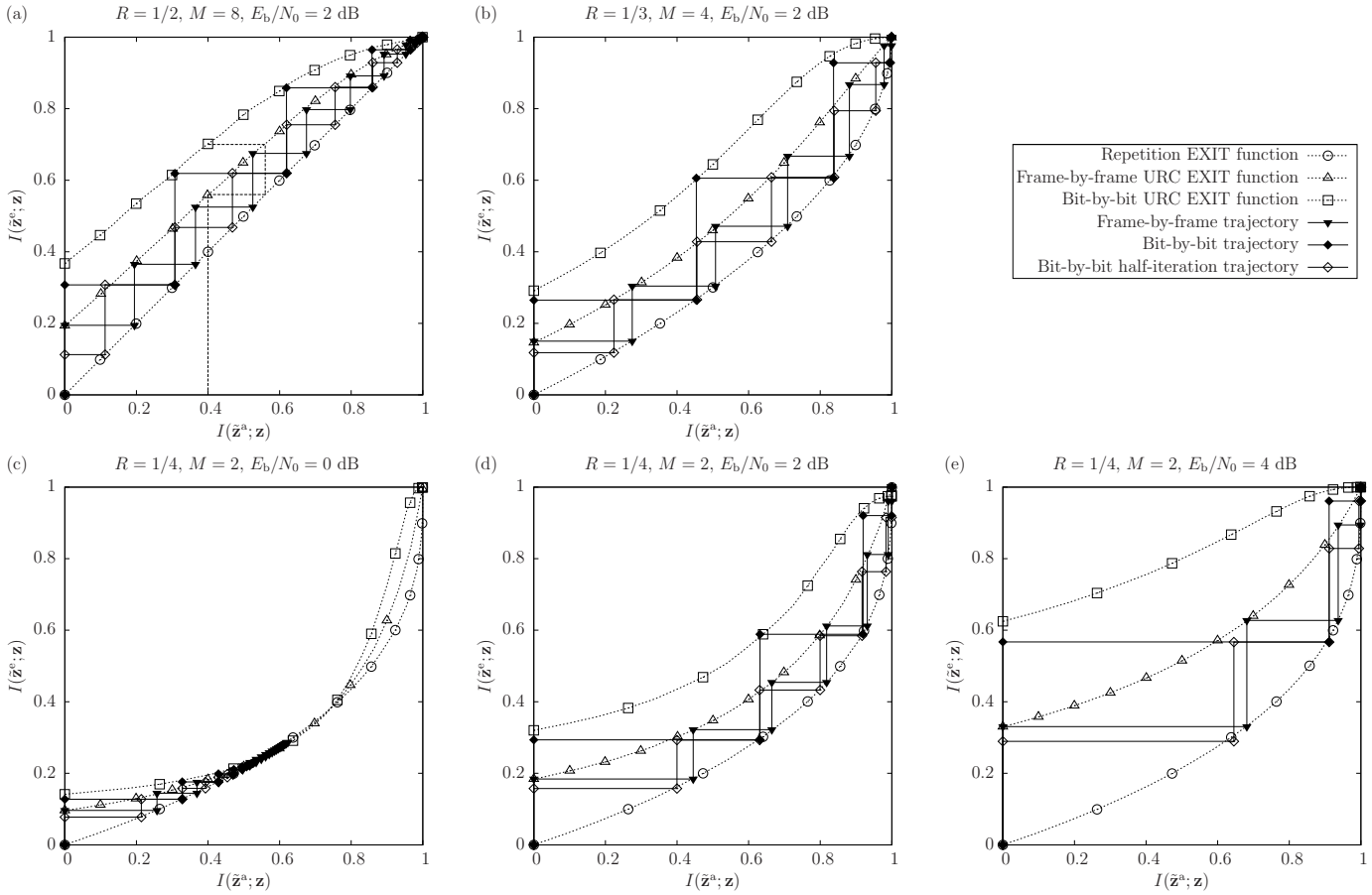


Fig. 4. EXIT charts of frame-by-frame and bit-by-bit RA decoders employing R-rate repetition codes and M-state URC codes, where (a) $R = 1/2$ and $M = 8$, (b) $R = 1/3$ and $M = 4$, as well as (c)-(e) $R = 1/4$ and $M = 2$. EXIT functions and trajectories are obtained for message lengths of $L = 10^6$ bits, when employing BPSK for communication over an AWGN channel having (a), (b) and (d) $E_b/N_0 = 2$ dB, (c) $E_b/N_0 = 0$ dB, as well as (e) $E_b/N_0 = 4$ dB.

decoder, which attempt to model the effect of this memory. These are motivated by the observation that each bit-by-bit decoding iteration can be thought of as being approximately equivalent to two frame-by-frame decoding iterations. More specifically, each point in the predicted bit-by-bit Log-MAP URC decoder's EXIT function is obtained by modeling two iterations of the frame-by-frame RA decoder. This is exemplified by the dashed line in Figure 4(a), which shows how to obtain the point in the bit-by-bit Log-MAP URC decoder's EXIT function corresponding to $I(\tilde{\mathbf{z}}^a; \mathbf{z}) = 0.4$. The first iteration of the frame-by-frame RA decoder may be modeled by drawing the vertical dashed line from $[I(\tilde{\mathbf{z}}^a; \mathbf{z}), I(\tilde{\mathbf{z}}^e; \mathbf{z})] = [0.4, 0]$ up to the frame-by-frame URC code's EXIT function, which is met at $[I(\tilde{\mathbf{z}}^a; \mathbf{z}), I(\tilde{\mathbf{z}}^e; \mathbf{z})] = [0.4, 0.56]$. The second iteration is modeled by extending this line horizontally to meet the repetition code's EXIT function at $[I(\tilde{\mathbf{z}}^a; \mathbf{z}), I(\tilde{\mathbf{z}}^e; \mathbf{z})] = [0.56, 0.56]$ and then vertically to meet that of the frame-by-frame URC code again at $[I(\tilde{\mathbf{z}}^a; \mathbf{z}), I(\tilde{\mathbf{z}}^e; \mathbf{z})] = [0.56, 0.7]$. This models the operation of the repetition decoder and then the URC decoder again. Finally, the corresponding point in the predicted bit-by-bit URC code's EXIT function is obtained by extending the line horizontally back to $I(\tilde{\mathbf{z}}^a; \mathbf{z}) = 0.4$, obtaining the point in the predicted bit-by-bit URC code's EXIT function at $[I(\tilde{\mathbf{z}}^a; \mathbf{z}), I(\tilde{\mathbf{z}}^e; \mathbf{z})] = [0.4, 0.7]$. This point represents the observation that if the repeat decoder produces aLLRs having the MI $I(\tilde{\mathbf{z}}^a; \mathbf{z}) = 0.4$, then the bit-by-bit URC decoder will produce eLLRs having the MI $I(\tilde{\mathbf{z}}^e; \mathbf{z}) = 0.7$. This process may be repeated to obtain the other points in the predicted bit-by-bit URC code's EXIT function, corresponding to other values of $I(\tilde{\mathbf{z}}^a; \mathbf{z})$. As shown in Figures 4(a)-(e), the approach advocated offers a good, albeit imperfect, match with the bit-by-bit RA decoder's iterative decoding trajectory in all the cases considered. Note that this analysis reveals that bit-by-bit decoding magnifies the EXIT chart tunnel of the RA decoder. More specifically, if the frame-by-frame RA decoder has an open EXIT chart tunnel, then that of the bit-by-bit RA decoder will be wider, expediting the iterative decoding convergence. However, if the frame-by-frame RA decoder has a closed EXIT chart tunnel, then the same is true for the bit-by-bit RA decoder, as exemplified in Figure 4(c). Owing to this, it will be shown in Section IV-B that the bit-by-bit and frame-by-frame RA decoders converge to the same BER performance, although the bit-by-bit decoder requires fewer decoding iterations to achieve this.

B. BER performances

Figure 5 contrasts the BER performance of the frame-by-frame and of the proposed bit-by-bit RA decoders, when employing the parametrizations of Section IV-A, but with the

aid of a more practical message length of $L = 10^3$ bits. During the simulation of the frame-by-frame and bit-by-bit RA decoders, both the BER and the cumulative computational complexity was recorded following each decoding iteration. This facilitated a three-dimensional plot of the BER versus complexity and E_b/N_0. The plots of Figure 5 were obtained by taking slices through this 3-D plot at particular values of computational complexity and interpolating, which facilitates fair comparisons. These particular values of computational complexity were specifically chosen so that they facilitate 3, 6 and 30 iterations of the frame-by-frame RA decoder.

As shown in Figures 5(a)-(c), the proposed bit-by-bit RA decoder offers superior BER performance to that of the frame-by-frame benchmarker, for all the parametrizations and for all the complexity limits considered. In particular, when employing the lowest considered complexity limit, the bit-by-bit RA decoder offers 0.72 dB, 0.83 dB and 0.86 dB gain for the $R = 1/2$-, $1/3$- and $1/4$-rate parametrizations, respectively. Note that this is achieved 'for free', since the comparisons are fair in terms of their overall decoding complexity, as well as in terms of their transmission energy, bandwidth and duration. However, the attainable gains diminish as the complexity limit is increased. This may be expected, because the bit-by-bit RA decoder's EXIT chart tunnel opens at the same E_b/N_0 value as that of the frame-by-frame benchmarker, as discussed in Section IV-A. Owing to this, similar BER performances are achieved, provided the complexity limit is sufficiently high for both schemes to achieve iterative decoding convergence.

V. Conclusions

In this paper, we have proposed a novel bit-by-bit iterative decoding technique, which was shown to expedite the convergence of RA decoders. In a frame-by-frame RA decoder, the repeat and accumulate component decoders are operated iteratively, with one component decoder kept idle, whenever the other is activated. The proposed bit-by-bit RA decoder expedites this process by allowing both component decoders to operate simultaneously, exchanging outputs continuously without buffering. In contrast to previous bit-by-bit iterative decoding techniques, ours reduces the number of decoding iterations required to achieve convergence and improves the associated error correction capability. Furthermore, we have proposed a novel technique for reducing the complexity of our bit-by-bit RA decoder. We have also proposed a novel EXIT chart analysis technique, which demonstrates that each decoding iteration of our bit-by-bit RA decoder is equivalent to two iterations of the conventional frame-by-frame RA decoder. Overall, we have demonstrated that in a range of practical scenarios, the proposed bit-by-bit RA decoder offers gains of up to 0.86 dB, without imposing any additional decoding complexity and without requiring any additional transmission-energy, -bandwidth or -duration.

Appendix A
Derivation of (4)

We are interested in the situation where at least one of the $(N-1)$ elements in the k^{th} column of the equivalent interleaver $\mathbf{\Pi}$ has a higher value than k. When the interleaver $\boldsymbol{\pi}$ adopts random designs according to the uniform interleaver distribution [1], the fraction of columns that are of the type described above will vary from design to design. The expected value for the fraction of columns that are of the type described above across all random designs is given by

$$P_a = \frac{1}{LN}\sum_{k=1}^{LN} \Pr\left(\bigcup_{n=1}^{N-1} \Pi_{n,k} > k\right), \tag{6}$$

where $\Pr\left(\bigcup_{n=1}^{N-1} \Pi_{n,k} > k\right)$ is the probability that the k^{th} column of $\mathbf{\Pi}$ is of the type described above. This fraction may be obtained as one minus the fraction of columns that are not of the type described above, according to

$$P_a = 1 - \frac{1}{LN}\sum_{k=1}^{LN} \Pr\left(\bigcap_{n=1}^{N-1} \Pi_{n,k} < k\right), \tag{7}$$

where $\Pr\left(\bigcap_{n=1}^{N-1} \Pi_{n,k} < k\right)$ is the probability that the k^{th} column of $\mathbf{\Pi}$ is not of the type described above. This probability is given by

$$\Pr\left(\bigcap_{n=1}^{N-1} \Pi_{n,k} < k\right) \tag{8}$$

$$= \prod_{n=1}^{N-1} \Pr\left(\Pi_{n,k} < k \,\middle|\, \bigcap_{n'=1}^{n-1} \Pi_{n',k} < k\right) \tag{9}$$

For cases where $k \geq N$, we have

$$\Pr\left(\bigcap_{n=1}^{N-1} \Pi_{n,k} < k\right) \tag{10}$$

$$= \frac{k-1}{LN} \times \frac{k-2}{LN-1} \times \ldots \times \frac{k-(N-1)}{LN-(N-1)+1} \tag{11}$$

$$= \frac{\binom{k-1}{N-1}}{\binom{LN}{N-1}}. \tag{12}$$

Meanwhile, for cases where $k < N$, we have $\Pr\left(\bigcap_{n=1}^{N-1} \Pi_{n,k} < k\right) = 0$. Substituting these results into (7) yields

$$P_a = 1 - \frac{1}{LN}\frac{\sum_{k=N}^{LN}\binom{k-1}{N-1}}{\binom{LN}{N-1}}. \tag{13}$$

Substituting $m = k-1$ into the index of the summation in (13) gives

$$P_a = 1 - \frac{1}{LN}\frac{\sum_{m=N-1}^{LN-1}\binom{m}{N-1}}{\binom{LN}{N-1}}. \tag{14}$$

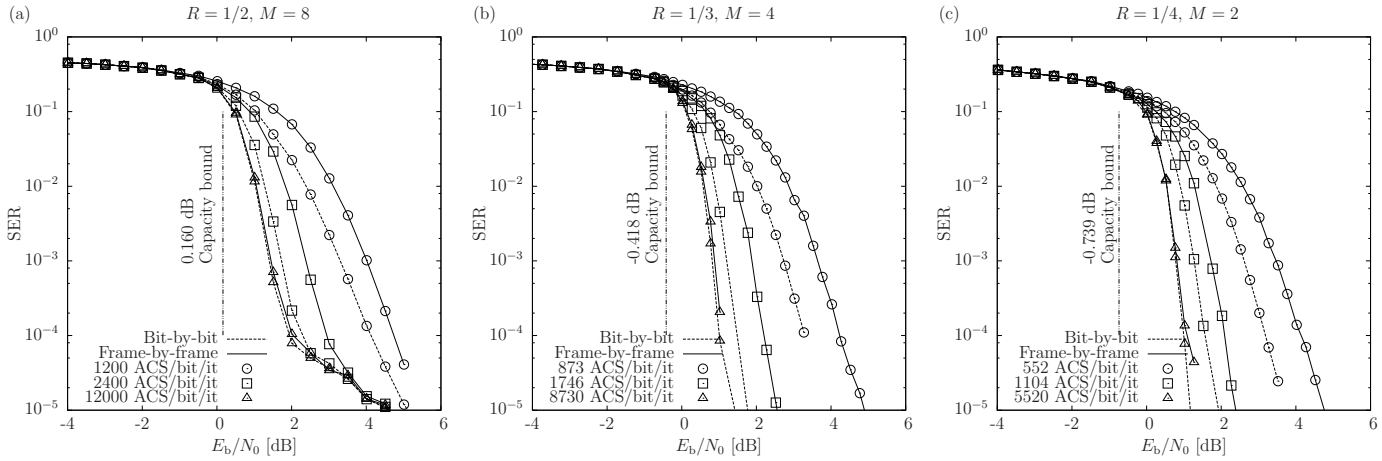


Fig. 5. BER plots of frame-by-frame and bit-by-bit RA decoders employing R-rate repetition codes and M-state URC codes, where (a) $R = 1/2$ and $M = 8$, (b) $R = 1/3$ and $M = 4$, as well as (c) $R = 1/4$ and $M = 2$. These plots are obtained for message lengths of $L = 10^3$ bits, when employing BPSK for communication over an AWGN channel having various E_b/N_0 values. In each case, plots are provided for three iterative decoding complexity limits, which are chosen to facilitate 3, 6 and 30 iterations of the frame-by-frame RA decoder.

Applying the identity $\sum_{m=k}^{n} \binom{m}{k} = \binom{n+1}{k+1}$ to (14) gives

$$P_a = 1 - \frac{1}{LN} \frac{\binom{LN}{N}}{\binom{LN}{N-1}}. \tag{15}$$

Finally, applying the identity $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ to (15) and simplifying it yields (4).

REFERENCES

[1] D. Divsalar, H. Jin, and R. J. McEliece, "Coding theorems for 'turbo-like' codes," in *Proc. Allerton Conf. on Communications, Control and Computing*, (Urbana, IL, USA), pp. 201–210, Sept. 1998.

[2] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "Soft-input soft-output modules for the construction and distributed iterative decoding of code networks," *Euro. Trans. Telecommun.*, vol. 9, pp. 155–172, Mar. 1998.

[3] M. F. U. Butt, S. X. Ng, and L. Hanzo, "Self-concatenated code design and its application in power-efficient cooperative communications," *IEEE Commun. Surveys Tutorials*, vol. 14, pp. 858–883, Sept. 2012.

[4] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," *Telecommunications and Data Acquisition Progress Report*, vol. 122, pp. 56–65, Apr. 1995.

[5] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (Corresp.)," *IEEE Trans. Inform. Theory*, vol. 20, pp. 284–287, Mar. 1974.

[6] R. G. Maunder and L. Hanzo, "Near-capacity irregular variable length coding and irregular unity rate coding," *IEEE Trans. Wireless Commun.*, vol. 8, pp. 5500–5507, Nov. 2009.

[7] A. Roumy, S. Guemghar, G. Caire, S. Verdu, "Design methods for irregular repeat-accumulate codes," *IEEE Trans. Inform. Theory*, vol. 50, pp. 1711–1727, August 2004.

[8] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 429–445, Mar. 1996.

[9] L. Li, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A low-complexity turbo decoder architecture for energy-efficient wireless sensor networks," *IEEE Trans. VLSI Syst.*, vol. 21, pp. 14–22, Jan. 2013.

[10] K. Chung, "Irregular repeat quaternary-accumulate (IRqA) codes and two-dimensional BCJR decoding," *IET Commun.*, vol. 6, pp. 1284–1290, July 2012.

[11] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proc. IEEE Int. Conf. on Communications*, vol. 2, (Seattle, WA, USA), pp. 1009–1013, June 1995.

[12] S. Yoon and Y. Bar-Ness, "A parallel MAP algorithm for low latency turbo decoding," *IEEE Commun. Lett.*, vol. 6, pp. 288–290, July 2002.

[13] T. Ilnseher, F. Kienle, C. Weis, and N. Wehn, "A 2.15GBit/s turbo code decoder for LTE Advanced base station applications," in *Proc. Int. Symp. on Turbo Codes and Iterative Information Processing*, (Gothenburg, Sweden), pp. 21–25, Aug. 2012.

[14] C. Argon and S. W. McLaughlin, "A parallel decoder for low latency decoding of turbo product codes," *IEEE Commun. Lett.*, vol. 6, pp. 70–72, Feb. 2002.

[15] Y.-C. Lu and E.-H. Lu, "A parallel decoder design for low latency turbo decoding," in *Proc. Conf. Innovative Computing Information Control*, (Kumamoto, Japan), p. 386, Sept. 2007.

[16] Y.-C. Lu, T. C. Chen, and E.-H. Lu, "Low-latency turbo decoder design by concurrent decoding of component codes," in *Proc. IEEE Int. Conf. Innovative Computing Information Control*, (Dalian, China), p. 533, June 2008.

[17] M. Taskaldiran, R. C. S. Morling, and I. Kale, "Increasing the speed of parallel decoding of turbo codes," in *Proc. Ph.D. Research in Microelectronics Electronics*, no. 1, (Cork, Ireland), pp. 304–307, July 2009.

[18] S. ten Brink, "Convergence of iterative decoding," *Electron. Lett.*, vol. 35, pp. 806–808, May 1999.

[19] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeat-accumulate codes," in *Proc. Int. Symp. Turbo Codes Related Topics*, (Brest, France), Sept. 2000.

[20] A. Abbasfar, D. Divsalar, and K. Yao, "Accumulate-repeat-accumulate codes," *IEEE Trans. Commun.*, vol. 55, pp. 692–702, Apr. 2007.

[21] G. Wang, I. Land, and A. Grant, "Design of irregular repeat accumulate codes for finite decoder iterations," *IEEE Trans. Commun.*, vol. 62, pp. 3092–3103, Sept. 2014.

[22] H.-A. Loeliger, "An introduction to factor graphs," *IEEE Signal Processing Magazine*, vol. 21, pp. 28–41, Jan. 2004.

[23] X. Wu, Y. Xue, and H. Xiang, "On concatenated zigzag Codes and their decoding schemes," *IEEE Commun. Lett.*, vol. 8, pp. 54–56, Jan. 2004.

[24] J. Li, K. R. Narayanan, and C. N. Georghiades, "Product accumulate codes: A class of codes with near-capacity performance and low decoding complexity," *IEEE Trans. Inform. Theory*, vol. 50, pp. 31–46, Jan. 2004.

[25] K. Yen, N. Veselinovic, and T. Matsumoto, "Space-time weighted nonbinary repeat-accumulate codes in frequency-selective MIMO channels," in *Proc. IEEE Veh. Technol. Conf.*, vol. 5, (Stockholm, Sweden), pp. 3152–3156, May 2005.

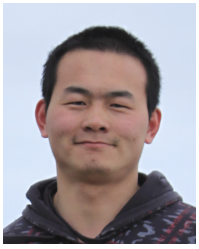
[26] A. Wolf, J. Ertel, and A. Finger, "Performance of serial concatenated codes under iterative decoding and different update modes," in *Proc. IEEE Workshop Signal Processing Advances Wireless Commun.*, (Cannes, France), July 2006.

[27] T. Bhatt and V. Stolpman, "Structured interleaves and decoder architectures for zigzag codes," in *Proc. Asilomar Conf. Signals Systems Computers*, (Pacific Grove, CA, USA), pp. 99–104, Oct. 2006.

[28] M.-C. Chiu, "Bandwidth-efficient modulation codes based on nonbinary

irregular repeat-accumulate codes," *IEEE Trans. Inform. Theory*, vol. 56, pp. 152–167, Jan. 2010.

Jinhui Chen Jinhui Chen received her BEng degree in Telecommunication Engineering (first class honors) in Xi'an Jiao Tong Liverpool University, Suzhou, China, in 2012. She was awarded a MSc degree in Wireless Communications (distinction) in the University of Southampton, Southampton, UK, in 2013. She is currently working in Panawell & Partners, LLC, Beijing, China. Her interests include intellectual property, channel coding, and antennas.



Wenbo Zhang received the M.E. degree in Information and Communication Engineering from the University of Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2011. He is currently working toward the Ph.D. degree with the Communications Research Group, Electronics and Computer Science, University of Southampton, Southampton, UK. His current research interests include joint source/channel coding and variable length coding.

Robert G. Maunder has studied with Electronics and Computer Science, University of Southampton, UK, since October 2000. He was awarded a first class honors BEng in Electronic Engineering in July 2003, as well as a PhD in Wireless Communications in December 2007. He became a lecturer in 2007 and an Associated Professor in 2013. Rob's research interests include joint source/channel coding, iterative decoding, irregular coding and modulation techniques. For further information on this research, please refer to http://users.ecs.soton.ac.uk/rm.

Lajos Hanzo (http://www-mobile.ecs.soton.ac.uk) FREng, FIEEE, FIET, Fellow of EURASIP, DSc received his degree in electronics in 1976 and his doctorate in 1983. In 2009 he was awarded the honorary doctorate "Doctor Honoris Causa" by the Technical University of Budapest. During his 35-year career in telecommunications he has held various research and academic posts in Hungary, Germany and the UK. Since 1986 he has been with the School of Electronics and Computer Science, University of Southampton, UK, where he holds the chair in telecommunications. He has successfully supervised 80 PhD students, co-authored 20 John Wiley/IEEE Press books on mobile radio communications totalling in excess of 10 000 pages, coauthored 1300+ research entries at IEEE Xplore, acted both as TPC and General Chair of IEEE conferences, presented keynote lectures and has been awarded a number of distinctions. Currently he is directing a 100-strong academic research team, working on a range of research projects in the field of wireless multimedia communications sponsored by industry, the Engineering and Physical Sciences Research Council (EPSRC) UK, the European IST Programme and the Mobile Virtual Centre of Excellence (VCE), UK. He is an enthusiastic supporter of industrial and academic liaison and he offers a range of industrial courses. He is also a Governor of the IEEE VTS. During 2008 - 2012 he was the Editor-in-Chief of the IEEE Press and a Chaired Professor also at Tsinghua University, Beijing. His research is funded by the European Research Council's Senior Research Fellow Grant. For further information on research in progress and associated publications please refer to http://www-mobile.ecs.soton.ac.uk