A Fully-Parallel Turbo Decoding Algorithm

Robert G. Maunder, *Senior Member, IEEE*

***Abstract*—This paper proposes a novel alternative to the Logarithmic Bahl-Cocke-Jelinek-Raviv (Log-BCJR) algorithm for turbo decoding, yielding significantly improved processing throughput and latency. While the Log-BCJR processes turbo-encoded bits in a serial forwards-backwards manner, the proposed algorithm operates in a fully-parallel manner, processing all bits in both components of the turbo code at the same time. The proposed algorithm is compatible with all turbo codes, including those of the LTE and WiMAX standards. These standardized codes employ odd-even interleavers, facilitating a novel technique for reducing the complexity of the proposed algorithm by 50%. More specifically, odd-even interleavers allow the proposed algorithm to alternate between processing the odd-indexed bits of the first component code at the same time as the even-indexed bits of the second component, and vice-versa. Furthermore, the proposed fully-parallel algorithm is shown to converge to the same error correction performance as the state-of-the-art turbo decoding algorithm. Owing to its significantly increased parallelism, the proposed algorithm facilitates throughputs and latencies that are up to 6.86 times superior to those of the state-of-the art algorithm, when employed for the LTE and WiMAX turbo codes. However, this is achieved at the cost of a moderately increased computational complexity and resource requirement.**

***Index Terms*—Turbo codes, Iterative decoding, Parallel algorithms, Throughput, WiMAX**

I. Introduction

During the past two decades, wireless communication has been revolutionized by channel codes that benefit from iterative decoding algorithms. For example, the Long Term Evolution (LTE) [1] and WiMAX [2] cellular telephony standards employ turbo codes [3], which comprise a concatenation of two convolutional codes. Conventionally, the Logarithmic Bahl-Cocke-Jelinek-Raviv (Log-BCJR) algorithm [4] is employed for the iterative decoding of these convolutional codes. Meanwhile, the WiFi standard for Wireless Local Area Networks (WLANs) [5] has adopted Low Density Parity Check (LDPC) codes [6], which may operate on the basis of the min-sum algorithm [7]. Owing to their strong error correction capability, these sophisticated channel codes have facilitated reliable communication at transmission throughputs that closely approach the capacity of the wireless channel. However, the achievable *transmission* throughput is limited by the *processing* throughput of the iterative decoding algorithm, if realtime operation is required. Furthermore, the iterative decoding algorithm's processing latency imposes a limit upon the end-to-end latency. This is particularly relevant, since multi-gigabit transmission throughputs and ultra-low end-to-end latencies can be expected to be targets for next-generation wireless communication standards [8]. Therefore, there is a demand for iterative decoding algorithms having multi-gigabit processing throughputs and ultra-low processing latencies.

Owing to the inherent parallelism of the min-sum algorithm, it may be operated in a fully-parallel manner, facilitating LDPC decoders having processing throughputs of up to 16.2 Gbit/s [9]. By contrast, the processing throughput of turbo decoders is limited by the inherently serial nature of the Log-BCJR algorithm, which is imposed by the data dependencies of its forward and backward recursions [4]. While a number of techniques have been proposed for increasing the parallelism of the Log-BCJR algorithm, the state-of-the-art LTE turbo decoder [10] achieves a processing throughput of just 2.15 Gbit/s. These techniques include shuffled iterative decoding [11], sub-block parallelism [12], [13], the Radix-4 transform [10] and the Non-Sliding Window (NSW) technique [10]. These techniques allow both recursions of both convolutional codes to be performed simultaneously, as well as allowing the recursions to consider several turbo-encoded bits per time period. However, in each case, the data dependencies of the forward and backward recursions require the turbo-encoded bits of each convolutional code to be processed serially, spread over numerous consecutive time periods. As a result, thousands of time periods are required to complete the iterative decoding process of the state-of-the-art turbo decoder of [10].

This motivates the novel turbo decoder algorithm of this paper, which dispenses with the recursions of the Log-BCJR algorithm and the associated data dependencies, facilitating fully-parallel turbo decoding. More specifically, the proposed fully-parallel turbo decoder algorithm is capable of processing all bits corresponding to both convolutional codes at the same time. The proposed fully-parallel algorithm is compatible with all turbo codes, including those of the LTE and WiMAX standards. These standardized turbo codes employ odd-even interleavers, facilitating a novel technique for reducing the complexity of the proposed algorithm by 50%. More specifically, odd-even interleavers allow the proposed algorithm to alternate between processing the odd-indexed bits of the first component code at the same time as the even-indexed bits of the second component, and vice-versa. This process is repeated iteratively, until a sufficient number of decoding iterations have been performed. Owing to this, the iterative decoding process can be completed using just tens of time periods, which is significantly lower than the number required by the state-of-the-art turbo decoder of [10].

The author is with Electronics and Computer Science, University of Southampton, SO17 1BJ, United Kingdom, e-mail: rm@ecs.soton.ac.uk

The financial support of the EPSRC, Swindon UK under the grants EP/J015520/1 and EP/L010550/1, as well as that of the TSB, Swindon UK under the auspices of grant TS/L009390/1 is gratefully acknowledged. The data for this paper can be found at 10.5258/SOTON/378330.

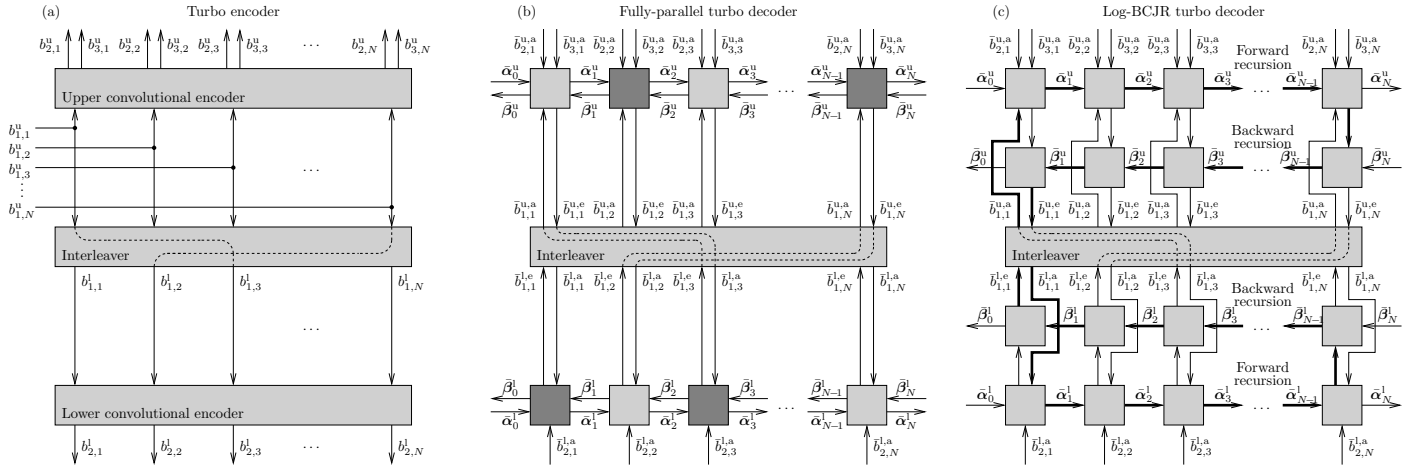

Fig. 1. Schematics of (a) a turbo encoder, (b) the proposed fully-parallel turbo decoder and (c) a Log-BCJR turbo decoder.

Note that a number of fully-parallel turbo decoders have been previously proposed, although these suffer from significant disadvantages that are not manifested in the proposed algorithm. In [14], the min-sum algorithm is employed to perform turbo decoding. However, this approach only works for a very limited set of turbo code designs, which does not include those employed by any standards. A fully-parallel turbo decoder implementation that represents the soft information using analogue currents was proposed in [15], however it only supports very short message lengths N. Similarly, [16] proposes a fully-parallel turbo decoder algorithm that operates on the basis of stochastic bit sequences. However, this algorithm requires significantly more time periods than the Log-BCJR algorithm, therefore having a significantly lower processing throughput.

The rest of this paper is structured as follows. Section II provides background information on turbo encoding and introduces the notation that will be employed throughout this paper. The proposed fully-parallel turbo decoding algorithm is described for generalized turbo codes in Section III, before being applied to the LTE and WiMAX turbo codes, where the above-described 50% reduction in complexity can be afforded. In Section IV, the proposed fully-parallel turbo decoder is compared with the state-of-the-art design at a purely algorithmic level. This is motivated, since very different processing throughputs, latencies, energy consumptions, hardware resource requirements and error correction capabilities may be expected to result for implementations of the proposed algorithm using different hardware platforms, such as Application Specific Integrated Circuit (ASIC), Field Programmable Gate Array (FPGA), Network on Chip (NoC) and General Purpose Graphics Processing Unit (GPGPU) technology, for example. However, all of these platform-dependent hardware characteristics will rely on the common set of fundamental *algorithmic* characteristics that are quantified and thoroughly compared in this paper. More specifically, the proposed fully-parallel algorithm is shown to converge to the same error correction performance as the state-of-the-art turbo decoding algorithm, regardless of which turbo code it is applied for. Owing to its significantly increased parallelism, the proposed algorithm facilitates throughputs and latencies that are up to 6.86 times superior to those of the state-of-the art algorithm, when employed for the LTE and WiMAX turbo codes. However, this is achieved at the cost of a moderately increased computational complexity and resource requirement. Finally, some conclusions are offered in Section V.

II. Turbo encoder

This section provides background information on turbo encoding and introduces the notation that will be employed throughout the remainder of this paper. Section II-A describes a simplified turbo encoder, which facilitates a simplified introduction of the proposed fully-parallel turbo decoder in Section III. Sections II-B and II-C discuss the differences between the simplified turbo encoder of Section II-A and those of LTE and WiMAX, respectively.

A. Simplified turbo encoder

Figure 1(a) depicts a simplified turbo encoder, which does not employ termination or tailbiting. This may be employed to encode a message frame $\mathbf{b}_1^{\mathrm{u}} = [b_{1,k}^{\mathrm{u}}]_{k=1}^{N}$ comprising N number of bits, each having a binary value $b_{1,k}^{\mathrm{u}} \in \{0, 1\}$. This message frame is provided to an upper convolutional encoder, as shown in Figure 1(a). This encoder uses the process described below to generate two N-bit encoded frames, namely a parity frame $\mathbf{b}_2^{\mathrm{u}} = [b_{2,k}^{\mathrm{u}}]_{k=1}^{N}$ and a systematic frame $\mathbf{b}_3^{\mathrm{u}} = [b_{3,k}^{\mathrm{u}}]_{k=1}^{N}$. Meanwhile, the message frame $\mathbf{b}_1^{\mathrm{u}}$ is interleaved, in order to obtain the N-bit interleaved message frame $\mathbf{b}_1^{\mathrm{l}} = [b_{1,k}^{\mathrm{l}}]_{k=1}^{N}$, as shown in Figure 1(a). This is provided to a lower convolutional encoder, which also uses the process described below to generate another N-bit encoded frames, namely a parity frame $\mathbf{b}_2^{\mathrm{l}} = [b_{2,k}^{\mathrm{l}}]_{k=1}^{N}$. Here, the superscripts 'u' and 'l' indicate relevance to the upper and lower convolutional encoders, respectively. However, throughout the remainder of this paper, these superscripts are only used when necessary to explicitly distinguish between the two convolutional encoders and are omitted when the discussion applies equally to both. Note that the turbo encoder represents the N bits of the message frame $\mathbf{b}_1^{\mathrm{u}}$ using three encoded frames, comprising a total of $3N$ bits

and resulting in a turbo coding rate of $R = N/(3N) = 1/3$. Following turbo encoding, the encoded frames may be modulated onto a wireless channel and transmitted to a receiver.

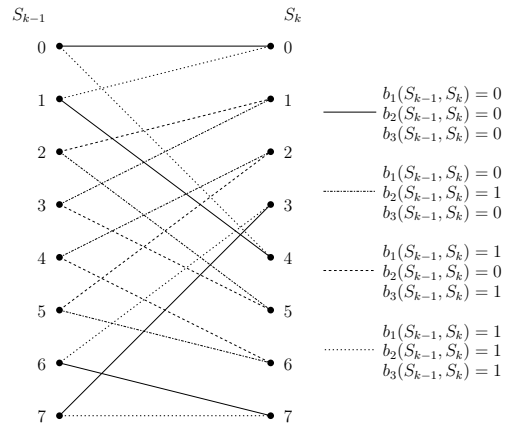

Fig. 2. State transition diagram of the LTE turbo code.

Both convolutional encoders operate in the same manner, on the basis of a state transition diagram like the $M = 8$-state example of Figure 2. The upper convolutional encoder begins from an initial state of $S_0 = 0$ and successively transitions into each subsequent state $S_k \in \{0, 1, 2, \ldots, M-1\}$ by considering the corresponding message bit $b_{1,k}$. Since there are two possible values for the message bit $b_{1,k} \in \{0, 1\}$, there are $K = 2$ possible values for the state S_k that can be reached by transitioning from the previous state S_{k-1}. In Figure 2 for example, a previous state of $S_{k-1} = 0$ implies that the subsequent state is selected from $S_k \in \{0, 4\}$. This example can also be expressed using the notation $c(0, 0) = 1$ and $c(0, 4) = 1$, where $c(S_{k-1}, S_k) = 1$ indicates that it is possible for the convolutional encoder to transition from S_{k-1} into S_k, whereas $c(S_{k-1}, S_k) = 0$ indicates that this transition is impossible. Of the $K = 2$ options, the value for the state S_k is selected such that $b_1(S_{k-1}, S_k) = b_{1,k}$. For example, $S_{k-1} = 0$ and $b_{1,k} = 0$ gives $S_k = 0$, while $S_{k-1} = 0$ and $b_{1,k} = 1$ gives $S_k = 4$ in Figure 2. In turn, binary values are selected for the corresponding bit in the parity frame $\mathbf{b}_2$ and, in the case of the upper convolutional encoder, the systematic frame $\mathbf{b}_3$, according to $b_{2,k} = b_2(S_{k-1}, S_k)$ and $b_{3,k} = b_3(S_{k-1}, S_k)$. In the example of Figure 2, $S_{k-1} = 0$ and $S_k = 0$ gives $b_{2,k} = 0$ and $b_{3,k} = 0$, while $S_{k-1} = 0$ and $S_k = 4$ gives $b_{2,k} = 1$ and $b_{3,k} = 1$.

B. LTE turbo encoder

The LTE turbo encoder [1] employs the state transition diagram of Figure 2, which has $M = 8$ states and $K = 2$ transitions per state. Furthermore, the LTE turbo encoder employs an odd-even interleaver [17] that supports various frame lengths N in the range 40 to 6144 bits. However, in contrast to the simplified turbo encoder of Figure 1(a), the LTE turbo encoder [1] employs twelve additional termination bits to force each convolutional encoder into the final state $S_{N+3} = 0$. More specifically, the upper encoder generates the three message termination bits $b^{\mathrm{u}}_{1,N+1}$, $b^{\mathrm{u}}_{1,N+2}$ and $b^{\mathrm{u}}_{1,N+3}$, as well as the three parity termination bits $b^{\mathrm{u}}_{2,N+1}$, $b^{\mathrm{u}}_{2,N+2}$ and $b^{\mathrm{u}}_{2,N+3}$. The lower convolutional encoder operates in a similar manner, generating corresponding sets of three message termination bits $b^{\mathrm{l}}_{1,N+1}$, $b^{\mathrm{l}}_{1,N+2}$ and $b^{\mathrm{l}}_{1,N+3}$, as well as three parity termination bits $b^{\mathrm{l}}_{2,N+1}$, $b^{\mathrm{l}}_{2,N+2}$ and $b^{\mathrm{l}}_{2,N+3}$. Owing to this, the LTE turbo encoder uses a total of $(3N + 12)$ bits to represent the N bits of the message frame $\mathbf{b}^{\mathrm{u}}_1$, giving a coding rate of $R = N/(3N + 12)$.

C. WiMAX turbo encoder

Like the LTE turbo encoder, the WiMAX turbo encoder [2] employs an odd-even interleaver, supporting various frame lengths N in the range 24 to 2400 bits. However, in contrast to the LTE turbo encoder, the WiMAX turbo encoder is *duo-binary* [2]. More specifically, the upper WiMAX convolutional encoder encodes two N-bit message frames at once $\mathbf{b}^{\mathrm{u}}_1$ and $\mathbf{b}^{\mathrm{u}}_2$. In response, it produces four N-bit encoded frames, namely two parity frames $\mathbf{b}^{\mathrm{u}}_3$ and $\mathbf{b}^{\mathrm{u}}_4$, as well as two systematic frames $\mathbf{b}^{\mathrm{u}}_5$ and $\mathbf{b}^{\mathrm{u}}_6$. Meanwhile, the message frames $\mathbf{b}^{\mathrm{u}}_1$ and $\mathbf{b}^{\mathrm{u}}_2$ are interleaved, in order to obtain the two N-bit interleaved message frames $\mathbf{b}^{\mathrm{l}}_1$ and $\mathbf{b}^{\mathrm{l}}_2$. These are encoded by the lower convolutional encoder, in order to generate two parity frames $\mathbf{b}^{\mathrm{l}}_3$ and $\mathbf{b}^{\mathrm{l}}_4$. Therefore, the WiMAX turbo encoder represents the $2N$ bits of the message frames $\mathbf{b}^{\mathrm{u}}_1$ and $\mathbf{b}^{\mathrm{u}}_2$ using six encoded frames, comprising a total of $6N$ bits and resulting a coding rate of $R = (2N)/(6N) = 1/3$. In the WiMAX turbo encoder, the upper and lower convolutional encoders operate on the basis of a state transition diagram having $K = 4$ transitions from each of $M = 8$ states, in correspondence to the four possible combinations of the two message bits. Rather than employing termination, WiMAX employs tailbiting to ensure that $S_N = S_0$, which may require S_N and S_0 to have non-zero values.

III. The Proposed Fully-Parallel Turbo Decoder

This section describes the operation of the proposed fully-parallel turbo decoding algorithm, which is compatible with all turbo codes. Section III-A considers the generalized applicability of the proposed algorithm, using an example of a parallel turbo decoder that corresponds to the simplified turbo encoder of Section II-A. Following this, Sections II-B and II-C discuss how the proposed fully-parallel turbo decoding algorithm may be applied to the LTE and WiMAX turbo codes, respectively.

A. Simplified turbo decoder

Following their transmission over a wireless channel, the three encoded frames $\mathbf{b}^{\mathrm{u}}_2$, $\mathbf{b}^{\mathrm{u}}_3$ and $\mathbf{b}^{\mathrm{l}}_2$ may be demodulated and provided to the turbo decoder of Figure 1(b). However, owing to the effect of noise in the wireless channel, the demodulator will be uncertain of the bit values in these encoded frames. Therefore, instead of providing frames comprising N hard-valued bits, the demodulator provides three frames each comprising N soft-valued *a priori* Logarithmic Likelihood Ratios (LLRs) $\bar{\mathbf{b}}^{\mathrm{u,a}}_2 = [\bar{b}^{\mathrm{u,a}}_{2,k}]^N_{k=1}$, $\bar{\mathbf{b}}^{\mathrm{u,a}}_3 = [\bar{b}^{\mathrm{u,a}}_{3,k}]^N_{k=1}$ and $\bar{\mathbf{b}}^{\mathrm{l,a}}_2 = [\bar{b}^{\mathrm{l,a}}_{2,k}]^N_{k=1}$. Here, an LLR pertaining to the bit $b_{j,k}$ is defined by

$$\bar{b}_{j,k} = \ln \frac{\Pr(b_{j,k} = 1)}{\Pr(b_{j,k} = 0)}, \tag{1}$$

The proposed fully-parallel turbo decoding algorithm.

$$\bar{\delta}_k(S_{k-1}, S_k) = \left[\sum_{j=1}^{L} \left[b_j(S_{k-1}, S_k) \cdot \bar{b}^{\mathrm{a}}_{j,k}\right]\right] + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k) \tag{2}$$

$$\bar{\alpha}_k(S_k) = \left[\max^*_{\{S_{k-1}|c(S_{k-1},S_k)=1\}} \left[\bar{\delta}_k(S_{k-1}, S_k)\right]\right] - \bar{\beta}_k(S_k) \tag{3}$$

$$\bar{\beta}_{k-1}(S_{k-1}) = \left[\max^*_{\{S_k|c(S_{k-1},S_k)=1\}} \left[\bar{\delta}_k(S_{k-1}, S_k)\right]\right] - \bar{\alpha}_{k-1}(S_{k-1}) \tag{4}$$

$$\bar{b}^{\mathrm{e}}_{j,k} = \left[\max^*_{\{(S_{k-1},S_k)|b_j(S_{k-1},S_k)=1\}} \left[\bar{\delta}_k(S_{k-1}, S_k)\right]\right] - \left[\max^*_{\{(S_{k-1},S_k)|b_j(S_{k-1},S_k)=0\}} \left[\bar{\delta}_k(S_{k-1}, S_k)\right]\right] - \bar{b}^{\mathrm{a}}_{j,k} \tag{5}$$

where the superscripts 'a', 'e' or 'p' may be appended to indicate an *a priori*, extrinsic or *a posteriori* LLR, respectively.

The demodulator provides these *a priori* LLRs to the fully-parallel turbo decoder's $2N$ algorithmic blocks, which are shown in Figure 1(b) arranged in two rows. More specifically, the *a priori* parity LLR $\bar{b}^{\mathrm{u,a}}_{2,k}$ and the *a priori* systematic LLR $\bar{b}^{\mathrm{u,a}}_{3,k}$ are provided to the k^{th} algorithmic block in the upper row shown in Figure 1(b). Furthermore, the interleaver of Figure 1(b) provides the k^{th} algorithmic block in the upper row with the *a priori* message LLR $\bar{b}^{\mathrm{u,a}}_{1,k}$, as will be detailed below. Meanwhile, the k^{th} algorithmic block in the lower row is correspondingly provided with the *a priori* LLRs $\bar{b}^{\mathrm{l,a}}_{1,k}$ and $\bar{b}^{\mathrm{l,a}}_{2,k}$. Note that the algorithmic blocks in the lower row of Figure 1(b) are not provided with any *a priori* systematic LLRs, hence eliminating the requirement to interleave $\bar{\mathbf{b}}^{\mathrm{u,a}}_{3}$. In addition to the above-mentioned LLRs, the k^{th} algorithmic block in each row is also provided with a vector of *a priori* forward state metrics $\bar{\boldsymbol{\alpha}}_{k-1} = [\bar{\alpha}_{k-1}(S_{k-1})]^{M-1}_{S_{k-1}=0}$ and a vector of *a priori* backward state metrics $\bar{\boldsymbol{\beta}}_k = [\bar{\beta}_k(S_k)]^{M-1}_{S_k=0}$, as will be detailed below. All algorithmic blocks operate in an identical manner, using the equations provided in (2) – (5). Note that these equations are stated in a fully generalized manner, allowing them to be applied to any turbo code, having any state transition diagram and any number L of *a priori* LLRs per algorithmic block.

More specifically, (2) is employed in order to combine the L *a priori* LLRs with the *a priori* state metrics of $\bar{\boldsymbol{\alpha}}_{k-1}$ and $\bar{\boldsymbol{\beta}}_k$. Here, each algorithmic block in the upper row employs $L = 3$ *a priori* LLRs $\bar{b}^{\mathrm{u,a}}_{1,k}$, $\bar{b}^{\mathrm{u,a}}_{2,k}$ and $\bar{b}^{\mathrm{u,a}}_{3,k}$, while $L = 2$ *a priori* LLRs $\bar{b}^{\mathrm{l,a}}_{1,k}$ and $\bar{b}^{\mathrm{l,a}}_{2,k}$ are employed for the lower row. This produces an *a posteriori* metric $\bar{\delta}(S_{k-1}, S_k)$ for each transition in the state transition diagram, namely for each pair of states S_{k-1} and S_k for which $c(S_{k-1}, S_k) = 1$. These *a posteriori* transition metrics are then combined by (3), (4) and (5), in order to produce the vector of extrinsic forward state metrics $\bar{\boldsymbol{\alpha}}_k = [\bar{\alpha}_k(S_k)]^{M-1}_{S_k=0}$, the vector of extrinsic backward state metrics $\bar{\boldsymbol{\beta}}_{k-1} = [\bar{\beta}_{k-1}(S_{k-1})]^{M-1}_{S_{k-1}=0}$ and the extrinsic message LLR $\bar{b}^{\mathrm{e}}_{1,k}$, respectively. These equations employ the Jacobian logarithm, which is defined for two operands as

$$\max^*(\bar{\delta}_1, \bar{\delta}_2) = \max(\bar{\delta}_1, \bar{\delta}_2) + \ln\left(1 + e^{-|\bar{\delta}_1 - \bar{\delta}_2|}\right) \tag{6}$$

and may be extended to more operands by exploiting its associativity property. Alternatively, the exact $\max^*$ operator of (6) may be optionally replaced with the approximation [4]

$$\max^*(\bar{\delta}_1, \bar{\delta}_2) \approx \max(\bar{\delta}_1, \bar{\delta}_2), \tag{7}$$

in order to reduce the complexity of the proposed fully-parallel turbo decoder, at the cost of slightly degrading its error correction performance.

The proposed fully-parallel turbo decoder is operated iteratively, where each of the I iterations comprises the operation of all algorithmic blocks shown in Figure 1(b). The turbo decoder may be considered to be fully-parallel, since each iteration is completed within just $T = 1$ time period, by operating all $2N$ of the algorithmic blocks simultaneously. In general, the extrinsic information produced by each algorithmic block in Figure 1(b) is exchanged with those provided by the connected algorithmic blocks, to be used as *a priori* information in the next decoding iteration. More specifically, the k^{th} algorithmic block in each row provides the vectors of extrinsic state metrics $\bar{\boldsymbol{\alpha}}_k$ and $\bar{\boldsymbol{\beta}}_{k-1}$ for the neighboring algorithmic blocks to employ in the next decoding iteration. Furthermore, the k^{th} algorithmic block in each row passes the extrinsic message LLR $\bar{b}^{\mathrm{e}}_{1,k}$ through the interleaver, to be used as an *a priori* LLR by the connected block in the other row during the next decoding iteration. Meanwhile, this block in the other row provides an extrinsic message LLR which is used as the *a priori* message LLR $\bar{b}^{\mathrm{a}}_{1,k}$ during the next decoding iteration. Note that the interleaver of Figure 1(b) may be hard-wired if only a single interleaver pattern is required or it may adopt a reconfigurable design in order to support any arbitrary interleaver pattern. At the start of the first decoding iteration however, no extrinsic information is available. In this case, the k^{th} algorithmic block in each row employs zero values for $\bar{b}^{\mathrm{a}}_{1,k}$, $\bar{\boldsymbol{\alpha}}_{k-1}$ and $\bar{\boldsymbol{\beta}}_k$. As an exception to this however, the first algorithmic block in the each row employs $\bar{\boldsymbol{\alpha}}_0 = [0, -\infty, -\infty, \ldots, -\infty]$ throughout all decoding iterations, since the convolutional encoders always begin from an initial state of $S_0 = 0$. Similarly, the last algorithmic block from the each row employs $\bar{\boldsymbol{\beta}}_N = [0, 0, 0, \ldots, 0]$ throughout all decoding iterations, since the final state of the the convolutional encoders S_N is not known in advance to the receiver, when termination is not employed. Following

the completion of the final decoding iteration, an *a posteriori* LLR pertaining to the k^{th} message bit $b^{\text{u}}_{1,k}$ may be obtained as $\bar{b}^{\text{u,p}}_{1,k} = \bar{b}^{\text{u,a}}_{1,k} + \bar{b}^{\text{u,e}}_{1,k}$. An estimation of the message bit $b^{\text{u}}_{1,k}$ may then be obtained as the result of the binary test $\bar{b}^{\text{u,p}}_{1,k} > 0$.

B. LTE turbo decoder

As described in Section II-B, the LTE turbo code employs an odd-even interleaver [17]. More explicitly, the LTE interleaver only connects algorithmic blocks from the upper row having an odd index k to blocks from the lower row that also have an odd index k. Similarly, blocks having even indices k in the upper row are only connected to blocks having even indices k in the lower row. Owing to this, the $2N$ algorithmic blocks of Figure 1(b) can be grouped into two sets, where all blocks within a particular set are independent, having no connections to each other. The first set comprises all algorithmic blocks from the upper row having an odd index k, as well as all blocks from the lower row having an even index k, which are depicted with light shading in Figure 1(b). Meanwhile, the second set is complementary to the first, comprising the algorithmic blocks having dark shading in Figure 1(b). In this way, the iterative exchange of extrinsic information between $2N$ algorithmic blocks can be instead thought of as an iterative exchange of extrinsic information between the two sets.

In the general case where the interleaver pattern prevents grouping into sets of independent algorithmic blocks, the approach described in Section III-A is recommended, where all algorithmic blocks are operated in every time period, corresponding to $T = 1$ time period per decoding iteration. However, in the case of an odd-even interleaver, the simultaneous operation of both sets of independent algorithmic blocks is analogous to juggling two balls, which are simultaneously thrown between two hands, but remain independent of each other. In the proposed fully-parallel turbo decoder, this corresponds to two independent iterative decoding processes, which have no influence on each other. Therefore, one of these independent iterative decoding processes can be considered to be redundant and may be discarded. This can be achieved by operating the algorithmic blocks of only one set in each time period, with consecutive time periods alternating between the two sets. With this approach, each decoding iteration can be considered to comprise $T = 2$ consecutive time periods. Although this is double the number required by the $T = 1$ approach described in Section III-A, this $T = 2$ approach requires half as many decoding iterations in order to achieve the same error correction performance. Therefore, the $T = 2$ approach maintains the same processing throughput and latency as the $T = 1$ approach, but achieves a 50% reduction in complexity per message frame. Note that the Log-BCJR turbo decoder cannot exploit an odd-even interleaver to achieve a similar improvement, as will be described in Section IV-A.

As described in Section II-B, the LTE turbo code employs twelve termination bits to force each of its convolutional encoders into the final state $S_{N+3} = 0$. In the receiver, the demodulator provides the corresponding LLRs $\bar{b}^{\text{u,a}}_{1,N+1}$, $\bar{b}^{\text{u,a}}_{1,N+2}$, $\bar{b}^{\text{u,a}}_{1,N+3}$, $\bar{b}^{\text{u,a}}_{2,N+1}$, $\bar{b}^{\text{u,a}}_{2,N+2}$ and $\bar{b}^{\text{u,a}}_{2,N+3}$ to the upper row, while the lower row is provided with $\bar{b}^{\text{l,a}}_{1,N+1}$, $\bar{b}^{\text{l,a}}_{1,N+2}$, $\bar{b}^{\text{l,a}}_{1,N+3}$, $\bar{b}^{\text{l,a}}_{2,N+1}$, $\bar{b}^{\text{l,a}}_{2,N+2}$ and $\bar{b}^{\text{l,a}}_{2,N+3}$. As shown in Figure 3, these LLRs can be provided to three additional algorithmic blocks, which are positioned at the end of each row in the proposed fully-parallel turbo decoder.

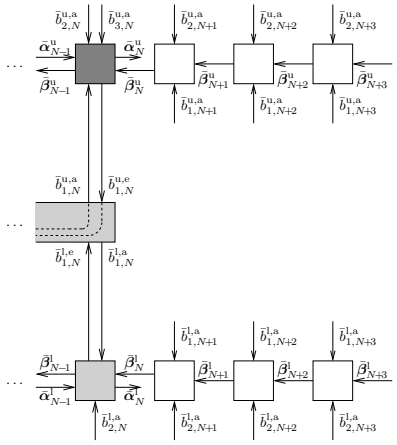

Fig. 3. Schematic of the proposed fully-parallel algorithm, when employing the termination technique of the LTE turbo code.

The three additional algorithmic blocks at the end of each row do not need to be operated iteratively, within the iterative decoding process. Instead, they can be operated just once, before the iterative decoding process begins, using a backwards recursion. More specifically, the algorithmic blocks with the index $k = N + 3$ may employ Equations (8) and (10) in order to process the $L = 2$ LLRs $\bar{b}^{\text{a}}_{1,N+3}$ and $\bar{b}^{\text{a}}_{2,N+3}$. Here, the state metrics $\bar{\boldsymbol{\beta}}_{N+3} = [0, -\infty, -\infty, \ldots, -\infty]$ should be employed since a final state of $S_{N+3} = 0$ is guaranteed. The resultant state metrics $\bar{\boldsymbol{\beta}}_{N+2}$ can then be provided to the algorithmic block having the index $k = N + 2$. In turn, this uses the same process in order to obtain $\bar{\boldsymbol{\beta}}_{N+1}$, which is then provided the block where $k = N + 1$ in order to obtain $\bar{\boldsymbol{\beta}}_N$ in the same way. The resultant values of $\bar{\boldsymbol{\beta}}_N$ may then be employed throughout the iterative decoding process, without any need to operate the three additional algorithmic blocks again. Note that there is no penalty associated with adopting this approach, since Equations (8) and (10) reveal that the values of $\bar{\boldsymbol{\beta}}_N$ are independent of all values that are updated as the iterative decoding process proceeds.

C. WiMAX turbo decoder

Like the LTE turbo code, the WiMAX turbo code employs an odd-even interleaver [17], allowing it to benefit from a 50% reduction in the complexity of the fully-parallel turbo decoder, as described in Section III-B. The algorithm of (2) – (5) supports the duo-binary nature of the WiMAX turbo code. Here, the algorithmic blocks in the upper row consider $L = 6$ *a priori* LLRs, while those in the lower row consider $L = 4$ LLRs. More specifically, the k^{th} algorithmic block in the upper row is provided with six *a priori* LLRs $\bar{b}^{\text{u,a}}_{1,k}$, $\bar{b}^{\text{u,a}}_{2,k}$, $\bar{b}^{\text{u,a}}_{3,k}$, $\bar{b}^{\text{u,a}}_{4,k}$, $\bar{b}^{\text{u,a}}_{5,k}$ and $\bar{b}^{\text{u,a}}_{6,k}$, using these to generate two extrinsic LLRs $\bar{b}^{\text{u,e}}_{1,k}$ and $\bar{b}^{\text{u,e}}_{2,k}$. By contrast, $\bar{b}^{\text{l,a}}_{1,k}$, $\bar{b}^{\text{l,a}}_{2,k}$, $\bar{b}^{\text{l,a}}_{3,k}$ and $\bar{b}^{\text{l,a}}_{4,k}$ are provided to the k^{th} algorithmic block in the lower row, which generates $\bar{b}^{\text{l,e}}_{1,k}$ and $\bar{b}^{\text{l,e}}_{2,k}$ in response. Tailbiting can be achieved

TABLE I
VARIOUS CHARACTERISTICS OF THE PROPOSED FULLY-PARALLEL ALGORITHM, THE LOG-BCJR ALGORITHM AND THE STATE-OF-THE-ART ALGORITHM OF [10], WHEN IMPLEMENTING THE LTE AND WIMAX TURBO DECODERS USING THE APPROXIMATE $\max^*$ OPERATOR OF (7).

Characteristic	Fully-parallel LTE $N \in [40, 6144]$	Fully-parallel WiMAX $N \in [24, 2400]$	Log-BCJR LTE $N \in [40, 6144]$	Log-BCJR WiMAX $N \in [24, 2400]$	State-of-the-art LTE $N \in [2048, 6144]$	State-of-the-art WiMAX $N \in [1440, 2400]$
Time periods per decoding iteration T	2	2	$4N$	$4N$	$N/32$	$N/16$
Time period duration D	8 (7)	9	9	11	3	3
Complexity per decoding iteration C	$155N$	$348N$	$171N$	$436N$	$320N$	$640N$
Decoding iterations required I	48	32	8	8	8	8
Computational resource requirement X	$79N$ $(80N)$	$176N$	87	220	14144	14144
Register resource requirement Y	$20N$ $(21N)$	$24N$	8	8	1792	1792
RAM resource requirement Z	0	0	$28N$	$48N$	$\frac{14N}{3} + 8320$	$\frac{28N}{3} + 8320$
Overall throughput $1/(T \cdot D \cdot I)$	$\frac{1}{768}$ $(\frac{1}{672})$	$\frac{1}{576}$	$\frac{1}{288N}$	$\frac{1}{352N}$	$\frac{4}{3N}$	$\frac{2}{3N}$
Overall latency $T \cdot D \cdot I$	768 (672)	576	$288N$	$352N$	$\frac{3N}{4}$	$\frac{3N}{2}$
Overall complexity $C \cdot I$	$7440N$	$11136N$	$1368N$	$3488N$	$2560N$	$5120N$
Overall resource requirement $9X + 5Y + Z$	$811N$ $(825N)$	$1704N$	$28N + 823$	$48N + 2020$	$\frac{14N}{3} + 144576$	$\frac{28N}{3} + 144576$

by employing $\bar{\boldsymbol{\alpha}}_0 = [0, 0, 0, \ldots, 0]$ and $\bar{\boldsymbol{\beta}}_N = [0, 0, 0, \ldots, 0]$ in the first iteration. In all subsequent iterations, the most-recently obtained values of $\bar{\boldsymbol{\alpha}}_N$ and $\bar{\boldsymbol{\beta}}_0$ can be employed for $\bar{\boldsymbol{\alpha}}_0$ and $\bar{\boldsymbol{\beta}}_N$, respectively.

IV. COMPARISON WITH THE LOG-BCJR TURBO DECODER

This section compares the proposed fully-parallel turbo decoder algorithm with the Log-BCJR turbo decoder algorithm, as well as with the state-of-the-art turbo decoder algorithm of [10], as summarized in Table I. For each of these schemes, Sections IV-A – IV-F quantify the number of time periods required per decoding iteration, the computational complexity per decoding iteration, the time period duration, the computational resource requirements, the memory resource requirements and the number of decoding iterations required to achieve a particular error correction performance, respectively. In Section IV-G, these characteristics are combined in order to quantify the overall throughput, latency, computational complexity and resource requirements of the various algorithms, when employed for both LTE and WiMAX turbo decoding. The comparisons of Table I assume that the turbo code is employed in a proprietary application that only uses a single interleaver pattern, as is typically assumed for fully-parallel LDPC decoders [9]. This allows the interleaver of the proposed fully-parallel turbo decoder algorithm to be hard-wired, which has the advantage of requiring no computational or memory resources and necessitating only a simple system controller. By contrast, the interleavers and controllers of the Log-BCJR and state-of-the-art turbo decoder algorithms require significant computational or memory resources [17], although these resources typically allow many different interleaver patterns to be supported at little or no additional cost. Since the interleavers and controllers of the various algorithms cannot be compared on a like-for-like basis, the comparisons of this section are restricted only to the algorithmic blocks, as depicted in Figure 1. As described in Section V, our future work will consider applications of the turbo code that require it to support multiple interleaver patterns.

A. Operation

Figure 1(c) depicts a simplified Log-BCJR turbo decoder, which corresponds to the simplified turbo encoder of Figure 1(a). Like the fully-parallel turbo decoder of Figure 1(b), the Log-BCJR turbo decoder is operated iteratively, where each of the I iterations comprises the operation of all algorithmic blocks shown in Figure 1(c). However as shown in Table I, $T = 4N$ consecutive time periods are required to complete each decoding iteration, so that the $4N$ algorithmic

blocks can be operated sequentially, in the order indicated by the bold arrows of Figure 1(c). These arrows indicate the data dependencies of the Log-BCJR algorithm, which impose the forward and backward recursions shown in Figure 1(c). Therefore, when implementing the LTE or WiMAX turbo decoders, the number of time periods per iteration required by the Log-BCJR algorithm is $2N$ times higher than the proposed fully-parallel algorithm's $T = 2$ time periods.

During the forward and backward recursions of the Log-BCJR algorithm, the k^{th} pair of algorithmic blocks in the upper and lower rows operate on the basis of Equations (8) – (12) [4]. During the forward recursion, the corresponding k^{th} algorithmic block in the upper row employs (8) to combine the $L = 3$ *a priori* LLRs $\bar{b}^{\text{u,a}}_{1,k}$, $\bar{b}^{\text{u,a}}_{2,k}$ and $\bar{b}^{\text{u,a}}_{3,k}$, while the $L = 2$ *a priori* LLRs $\bar{b}^{\text{l,a}}_{1,k}$ and $\bar{b}^{\text{l,a}}_{2,k}$ are combined for the lower row. This results in an *a priori* metric $\bar{\gamma}(S_{k-1}, S_k)$ for each transition in the state transition diagram. Following this, (9) combines these *a priori* transition metrics with the *a priori* forward state metrics of $\bar{\boldsymbol{\alpha}}_{k-1}$, in order to obtain the extrinsic forward state metrics of $\bar{\boldsymbol{\alpha}}_k$. These extrinsic state metrics are then passed to the $(k+1)^{\text{th}}$ algorithmic block, to be employed as *a priori* state metrics in the next time period. During the backward recursion, the corresponding k^{th} algorithmic block in the upper and lower rows employs (10) to combine the *a priori* metric $\bar{\gamma}(S_{k-1}, S_k)$ of each transition with the *a priori* backward state metrics of $\bar{\boldsymbol{\beta}}_k$. This produces the extrinsic backward state metrics of $\bar{\boldsymbol{\beta}}_{k-1}$, which may be passed to the $(k-1)^{\text{th}}$ algorithmic block, to be employed as *a priori* state metrics in the next time period. Furthermore, the k^{th} algorithmic block in the backward recursion of the upper rows employs (11) to obtain an *a posteriori* metric $\bar{\delta}(S_{k-1}, S_k)$ for each transition in the state transition diagram. Finally, the extrinsic message LLR $\bar{b}^{\text{e}}_{1,k}$ is obtained using (12). As in the proposed fully-parallel turbo decoder of Section III-A, zero-values are employed for the *a priori* message LLRs in the first decoding iteration of the Log-BCJR turbo decoder. In addition to supporting the simplified turbo decoder of Figure 1(c), the Log-BCJR algorithm of (8) – (12) supports the algorithmic blocks of the LTE turbo decoder having $L = 3$ and $L = 2$ *a priori* LLRs, as well as the blocks of the WiMAX turbo code having $L = 6$ and $L = 4$. Depending on whether termination or tailbiting is employed, the values described in Section III for $\bar{\boldsymbol{\alpha}}_0$ and $\bar{\boldsymbol{\beta}}_N$ can be employed in the Log-BCJR turbo decoder. Note that in the LTE turbo decoder, there is a computational complexity, computational resource requirement and memory resource requirement associated with processing the *a priori* LLRs corresponding to the twelve termination bits. However, for the sake of simplicity, these are not considered in the following discussions, since they are common to each algorithm considered and because they are small compared to the overall characteristics of these algorithms.

As may be expected, the proposed fully-parallel turbo decoding algorithm of (2) – (5) is related to that of the Log-BCJR turbo decoder (8) – (12). More explicitly, (2) can be derived by substituting (8) into (11). Using the identity $\max^*(\bar{\delta}_1 - \bar{\delta}_3, \bar{\delta}_2 - \bar{\delta}_3) = \max^*(\bar{\delta}_1, \bar{\delta}_2) - \bar{\delta}_3$, (3) and (4) can be derived by rearranging (11) and substituting it into (9) and (10), respectively.

Note that unlike the proposed fully-parallel turbo decoder, the Log-BCJR turbo decoder cannot exploit an odd-even interleaver to reduce its computational complexity by 50%, as mentioned in Section III-B. This is because the data dependencies within the Log-BCJR algorithm ensures that all LLRs and state metrics depend on those generated during previous iterations, to varying degrees. Owing to this, the Log-BCJR turbo decoder cannot be decomposed into two independent sets of algorithmic blocks. Likewise, the operation of the Log-BCJR turbo decoder cannot be considered to correspond to two independent iterative decoding processes, one of which is redundant. This prevents the Log-BCJR turbo decoder from exploiting an odd-even interleaver in order to achieve a 50% reduction in complexity, using the technique that may be employed by the proposed fully-parallel turbo decoder, as described in Section III-B.

Note that while the simplified Log-BCJR turbo decoder of Figure 1(c) requires $T = 4N$ time periods to complete each decoding iteration, several techniques have been proposed for significantly reducing this. For example, the NSW technique [10] may be employed to decompose the algorithmic blocks of Figure 1(c) into 32 windows, each comprising an equal number of consecutive blocks. Here, each window's recursions are initialized by results provided by the adjacent windows in the previous decoding iteration, eliminating the data dependency between windows in the current iteration and allowing them to be processed simultaneously. Furthermore, within each window, the NSW technique performs the forward and backward recursions simultaneously, only performing Equations (11) and (12) once these recursions have crossed over. Additionally, the Radix-4 transform [10] allows the number of algorithmic blocks employed in Figure 1(c) to be halved, along with the number of time periods required to process them. Here, each algorithmic block corresponds to the merger of two state transition diagrams into one, effectively doubling the number of *a priori* LLRs L considered by each algorithmic block. By combining the NSW technique and the Radix-4 transform, the state-of-the-art LTE turbo decoder [10] can complete each decoding iteration using just $T = N/32$ time periods, provided that the frame length satisfies $N \in [2048, 6114]$, as shown in Table I. Note however that this number is $N/64$ times higher than that of the proposed fully-parallel turbo decoder of Section III-A, which requires only $T = 2$ time periods per decoding iteration. When employing the maximum LTE frame length of $N = 6144$ bits, the number of time periods per decoding iteration required by the state-of-the-art LTE turbo decoder is nearly two orders-of-magnitude above the number required by the proposed fully-parallel algorithm.

As described above, the state-of-the-art LTE turbo decoding algorithm of [10] employs the Radix-4 transform to double the number of *a priori* LLRs considered by each algorithmic block, resulting in $L = 6$ for the blocks in the upper row and $L = 4$ for those in the lower row. Owing to this, the state-of-the-art algorithm can also be employed for WiMAX turbo decoding, since this naturally requires algorithmic blocks that consider $L = 6$ and $L = 4$ *a priori* LLRs, as described

The Log-BCJR turbo decoding algorithm.

$$\bar{\gamma}_k(S_{k-1}, S_k) = \sum_{j=1}^{L} \left[b_j(S_{k-1}, S_k) \cdot \bar{b}^{\mathrm{a}}_{j,k} \right] \tag{8}$$

$$\bar{\alpha}_k(S_k) = \max^*_{\{S_{k-1}|c(S_{k-1},S_k)=1\}} \left[\bar{\gamma}_k(S_{k-1}, S_k) + \bar{\alpha}_{k-1}(S_{k-1}) \right] \tag{9}$$

$$\bar{\beta}_{k-1}(S_{k-1}) = \max^*_{\{S_k|c(S_{k-1},S_k)=1\}} \left[\bar{\gamma}_k(S_{k-1}, S_k) + \bar{\beta}_k(S_k) \right] \tag{10}$$

$$\bar{\delta}_k(S_{k-1}, S_k) = \bar{\gamma}_k(S_{k-1}, S_k) + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k) \tag{11}$$

$$\bar{b}^{\mathrm{e}}_{j,k} = \left[\max^*_{\{(S_{k-1},S_k)|b_j(S_{k-1},S_k)=1\}} \left[\bar{\delta}_k(S_{k-1}, S_k) \right] \right] - \left[\max^*_{\{(S_{k-1},S_k)|b_j(S_{k-1},S_k)=0\}} \left[\bar{\delta}_k(S_{k-1}, S_k) \right] \right] - \bar{b}^{\mathrm{a}}_{j,k} \tag{12}$$

TABLE II
THE NUMBER OF OPERATIONS PER ALGORITHMIC BLOCK. THE NUMBER OF THESE OPERATIONS THAT CONTRIBUTE TO THE CRITICAL PATH OF THE ALGORITHMIC BLOCK IS PROVIDED IN CURLY BRACKETS.

Turbo code	Operation	Fully-parallel					Log-BCJR					
							Forward recursion		Backward recursion			Total
		Eq. (2)	Eq. (3)	Eq. (4)	Eq. (5)	Total	Eq. (8)	Eq. (9)	Eq. (10)	Eq. (11)	Eq. (12)	
LTE	$+$ or $-$	29.5 {3}	8	8	2 {2}	**47.5** {**5**}	1.5	12	12 {1}	28 {2}	2 {2}	**55.5** {**5**}
	$\max^*$	0	8	8	14 {3}	**30** {**3**}	0	8	8 {1}	0	14 {3}	**30** {**4**}
WiMAX	$+$ or $-$	74 {3}	8	8	4 {2}	**94** {**5**}	12	30	30 {1}	62 {2}	4 {2}	**138** {**5**}
	$\max^*$	0	24	24	32 {4}	**80** {**4**}	0	24	24 {2}	0	32 {4}	**80** {**6**}

in Section III-C. Note however that in this application, the turbo decoder does not benefit from halving the number of algorithmic blocks required, as is achieved when applying the Radix-4 transform to an LTE turbo decoder. On the other hand, the WiMAX turbo decoder can benefit from the NSW technique of the state-of-the-art algorithm, provided that $N \in [1440, 2440]$, resulting in $T = N/16$ time periods per decoding iteration, as shown in Table I. This number is $N/32$ times higher than that of the proposed fully-parallel turbo decoder of Section III-A.

B. Computational complexity

The energy that is consumed by a practical hardware implementation of a turbo decoder in order to decode a message frame depends on the computational complexity of the underlying algorithm. The number of additions, subtractions and $\max^*$ operations that are employed within each algorithmic block of the proposed fully-parallel and the Log-BCJR algorithms are quantified in Table II, for both the LTE and WiMAX turbo decoder. A number of techniques have been employed to minimize the number of operations that are listed in Table II. More specifically, the *a priori* metrics $\bar{\gamma}(S_{k-1}, S_k)$ of some particular transitions are equal to each other, allowing them to be computed once and then reused. Furthermore, some *a priori* metrics $\bar{\gamma}(S_{k-1}, S_k)$ are zero-valued and so there is no need to add them into the corresponding $\bar{\delta}(S_{k-1}, S_k)$, $\bar{\alpha}_k(S_k)$ or $\bar{\beta}_{k-1}(S_{k-1})$ calculations. Finally, when computing the extrinsic LLR $\bar{b}^{\mathrm{e}}_{1,k}$ in the WiMAX turbo decoder, the results of some $\max^*$ operations can be reused to compute $\bar{b}^{\mathrm{e}}_{2,k}$. Note that the algorithmic blocks in the upper row of the LTE and WiMAX turbo decoders consider a higher number of *a priori* LLRs L than those of the lower row. Owing to this, the computation of (2) or (8) in the algorithmic blocks of the upper row of the LTE turbo decoder requires one more addition than in the blocks of the lower row. Likewise, two more additions are required for the algorithmic blocks in the upper row of the WiMAX turbo decoder, compared to the blocks in the lower row. Therefore, Table II presents the *average* of the number of operations that are employed by (2) and (8) in the algorithmic blocks of the upper and lower rows, resulting in some non-integer values.

For both the LTE and WiMAX turbo decoders, the proposed fully-parallel turbo decoding algorithm requires fewer additions and subtractions than the Log-BCJR algorithm, as well as an equal number of $\max^*$ operations. When the approximation of (7) is employed, $\max^*$ operations can be considered to have a similar computational complexity to additions and subtractions [18]. Table I quantifies the total number of operations performed per iteration C, among all of the algorithmic blocks in the upper and lower rows. As shown in Table I, the computational complexity per decoding iteration C of the Log-BCJR algorithm is 1.1 and 1.25 times higher than that of the proposed algorithm, when implementing the LTE and WiMAX turbo decoders, respectively.

Note that the state-of-the-art LTE turbo decoder [10] employs the Radix-4 transform, as well as the approximation of (7). When employing the Radix-4 transform, the Log-BCJR LTE turbo decoder has the same complexity per algorithmic block as that presented in Table II for the Log-BCJR WiMAX turbo decoder. However, it should be noted that the Radix-4 transform halves the number of algorithmic blocks that

are required, as discussed in Section IV-A. Furthermore, as will be described in Section IV-E, the state-of-the-art LTE turbo decoder recalculates the *a priori* transition metrics and $5/6$ of the extrinsic state metrics during the forward and backward recursions, in order to reduce the corresponding memory resource requirement. Therefore, the state-of-the-art LTE turbo decoder has a complexity per decoding iteration C that is 2.06 times higher than that of the proposed algorithm, as shown in Table I. When applying the state-of-the-art algorithm's recalculation technique to the WiMAX turbo code, its complexity per decoding iteration C corresponds to 1.84 times higher than that of the proposed algorithm, as shown in Table I.

C. Time period duration

As described in Sections III and IV-A, it may be assumed that the operation of an algorithmic block in the schematics of Figure 1 can be completed within a single time period. However, the amount of time D that is required for each time period depends on the computational requirements of the algorithmic blocks. In a practical hardware implementation of a turbo decoder, each time period may correspond to one clock cycle, in which case the maximum clock frequency and hence the processing throughput is inversely proportional to the duration D of the time periods in the underlying algorithm. More specifically, the required duration D depends on the critical path through the data dependencies that are imposed by the computational requirements of the algorithmic blocks. For example, in the proposed fully-parallel algorithm, Equations (3), (4) and (5) are independent of each other, but they all depend upon (2). As a result, the computation of (2) must be completed first, but then (3), (4) and (5) can be computed in parallel. Of these three equations, it is (5) that requires the most time for computation, since it is a function of more variables than (3) and (4). Therefore, the critical path of the algorithmic blocks in the proposed fully-parallel algorithm depends on the computational requirements of (2) and (5).

Equation (2) is employed to obtain an *a posteriori* metric $\bar{\delta}(S_{k-1}, S_k)$ for each transition in the state transition diagram. However, these can all be calculated in parallel, using an addition of five variables in the case of the algorithmic blocks in the upper row of the LTE turbo decoder, which consider $L = 3$ *a priori* LLRs, for example. By contrast, an addition of just four variables is required in the case of the algorithmic blocks in the lower row, for which $L = 2$. A summation of v number of variables requires $v - 1$ additions, some of which can be performed in parallel. More specifically, the variables can be added together in pairs and then in a second step, the resultant sums can be added together in pairs. This process can continue until only a single sum remains, requiring a total of $\lceil \log_2(v) \rceil$ steps. Accordingly, Equation (2) contributes three additions to the critical path of the algorithmic blocks in the upper row of the proposed fully-parallel LTE turbo decoder, as well as two additions for the blocks in the lower row. The maximum of these two critical path contributions is presented in the corresponding curly brackets of Table II, since it imposes the greatest limitation on the time period duration. A similar analysis can be employed to determine each of the other critical path contributions that are provided in the curly brackets of Table II.

As shown in Table II, the critical path of the Log-BCJR algorithm is longer than that of the proposed fully-parallel algorithm, requiring time periods having a longer duration D and resulting in slower operation. When the approximation of (7) is employed, $\max^*$ operations can be considered to make similar contributions to the critical path as additions and subtractions. As shown in Table I, the critical path and hence the required time period duration D of the Log-BCJR algorithm is therefore 1.13 and 1.22 times higher than that of the proposed algorithm, when implementing the LTE and WiMAX turbo decoders, respectively.

Note however that the state-of-the-art LTE turbo decoder [10] employs the Radix-4 transform, as well as the approximation of (7). When employing the Radix-4 transform, the Log-BCJR LTE turbo decoder has the same critical path as that presented in Table II for the Log-BCJR WiMAX turbo decoder. However, the state-of-the-art LTE turbo decoder employs pipelining [10] to spread the computation of Equations (8) – (12) over several consecutive time periods. This reduces the critical path to that of Equation (10) alone, namely one addition and two $\max^*$ operations. By contrast, the proposed fully-parallel algorithm has a critical path comprising five additions and three $\max^*$ operations, as shown in Table I. Note however that the contribution of one addition can be eliminated from this total by employing a technique similar to pipelining. More specifically, the sum of the *a priori* parity LLRs $\bar{b}_2^{\mathrm{u,a}}$ and the *a priori* systematic LLRs $\bar{b}_3^{\mathrm{u,a}}$ may be computed before iterative decoding commences. As will be described in Section IV-E, the result may be stored and used throughout the iterative decoding process by the algorithmic blocks in the upper row of the proposed fully-parallel LTE turbo decoder. This reduces the critical path contribution of Equation (2) in the upper row to two additions, which is equal to that of the lower row. This reduces the critical path of the proposed fully-parallel LTE turbo decoder to 7 operations, as shown in brackets in Table I. Therefore, the critical path and time period duration D of the state-of-the-art LTE turbo decoder can be considered to be 0.43 times that of the proposed algorithm. Similarly, when applying the state-of-the-art algorithm to WiMAX turbo decoding, the result is the same critical path of one addition and two $\max^*$ operations. As shown in Table I, this critical path is 0.33 times that of the proposed algorithm, which requires five additions and four $\max^*$ operations.

D. Computational resource requirement

In a practical hardware implementation of a turbo decoder, the chip area or hardware resource requirement depends on the computational resource requirement X of the underlying algorithm. In the proposed fully-parallel turbo decoder algorithm of Figure 1(b), the algorithmic blocks each have an average computational resource requirement that is quantified by the totals provided in Table II. When an odd-even interleaver is not employed, every algorithmic block is operated in every time

period, resulting in a total computational resource requirement X of $2N$ algorithmic blocks. However, when employing an odd-even interleaver like that of those of the LTE and WiMAX turbo decoders, the computational resource requirement X is reduced to just N algorithmic blocks. This is because algorithmic blocks in the upper row of Figure 1(b) are not operated simultaneously with the corresponding blocks in the lower row in this case, as described in Section III-B. This allows computational resources to be shared between each pairing of blocks from the upper and lower rows. Note however that in this case, the computational resource requirements become slightly higher than the averages provided in Table II, since they must accommodate the higher number L of *a priori* LLRs that are considered by the blocks in the upper row. Furthermore, a set of N adders is required for computing the *a posteriori* LLR $\bar{b}_{1,k}^{\mathrm{u,p}} = \bar{b}_{1,k}^{\mathrm{u,a}} + \bar{b}_{1,k}^{\mathrm{u,e}}$ for each algorithmic block in the upper row following the completion of the iterative decoding process. When the approximation of (7) is employed, $\max^*$ operations can be considered to have a similar computational resource requirement to additions and subtractions [18]. Based upon these observations, Table I quantifies the total computational resource requirement X of the proposed fully-parallel turbo decoder when employed for LTE and WiMAX turbo decoding. Note that if the pipelining technique of Section IV-C is employed for reducing the time period duration D of the LTE turbo decoder, then a set of N adders is required in order to accommodate the computation of $\bar{b}_{2}^{\mathrm{u,a}} + \bar{b}_{3}^{\mathrm{u,a}}$ for each algorithmic block in the upper row, as shown in brackets in Table I.

Similarly, the Log-BCJR turbo decoder algorithm of Figure 1(c), the algorithmic blocks each have an average computational resource requirement that is quantified by the totals provided in Table II. However, in contrast to the proposed fully-parallel turbo decoder, the Log-BCJR turbo decoder algorithm has a computational resource requirement of just one of the algorithmic blocks used during the forward recursion and one used during the backward recursion. This is because only one of the algorithmic blocks shown in Figure 1(c) is operated in each time period, allowing the same computational resources to be reused for each algorithmic block. Note however that the computational resource requirements are slightly higher than the average values provided in Table II, since they must accommodate the higher values of L that are used for the blocks in the upper row. Furthermore, one more adder is required for computing the *a posteriori* LLRs following the completion of the iterative decoding process, although this adder can be shared between the different algorithmic blocks. Based upon these observations, Table I quantifies the total computational resource requirement X of the Log-BCJR turbo decoder algorithm.

As in the Log-BCJR turbo decoder algorithm, computational resources are reused during the forward and backward recursions of the state-of-the-art LTE turbo decoder algorithm of [10]. However, this algorithm requires 64 sets of computational resources since it employs the NSW technique to perform a forward and backward recursion simultaneously, in each of 32 windows of algorithmic blocks. Furthermore, each of these 64 sets requires computational resources for simultaneously recalculating the *a priori* transition metrics and the extrinsic state metrics, in order to reduce the corresponding memory resource requirement, as mentioned in Section IV-B. Overall, each of the 64 sets requires computational resources for performing each of (8) – (12). Since it employs the Radix-4 transform, the state-of-the-art LTE turbo decoder has computational resources that are quantified by the corresponding values presented in Table II for the Log-BCJR WiMAX turbo decoder, as described in Section IV-B. This allows the state-of-the-art LTE turbo decoder to also be applied for the WiMAX turbo decoder, without any change to the computational resource requirements. Based upon these observations, Table I quantifies the total computational resource requirement X of the state-of-the-art turbo decoder algorithm, when considering the higher values of L used in the upper row and the computation of the *a posteriori* LLRs, as described above. As shown in Table I, the computational resource requirement X of the proposed fully-parallel turbo decoder algorithm can be orders of magnitude higher than those of the Log-BCJR and state-of-the-art turbo decoder algorithms, depending on the frame length N. However, it will be shown in Section IV-G that the comparison is significantly more favorable, when additionally considering the memory resource requirement and processing throughput of the algorithms.

E. Memory resource requirement

In addition to the computational resource requirement X of Section IV-D, the chip area or hardware resource requirements of a practical turbo decoder hardware implementation also depend on the underlying algorithm's memory resource requirement, which comprises two parts. Firstly, the register resource requirement Y quantifies the amount of memory that is arranged into registers, which store values that can be accessed all at once, as often as in every time period. By contrast, the Random Access Memory (RAM) resource requirement Z quantifies the amount of storage that is arranged into RAM, which stores different values in different addresses that are accessed in different time periods.

In the proposed fully-parallel turbo decoder algorithm of Figure 1(b), memory resources are required for storing the forward state metrics, backward state metrics and extrinsic LLRs of Equations (3), (4) and (5), respectively. These outputs are produced whenever an algorithmic block is operated and must be stored into the next time period, where they are consumed by the connected blocks. However, when employing an odd-even interleaver like that of those of the LTE and WiMAX turbo decoders, the algorithmic blocks in the upper row of Figure 1(b) are not operated simultaneously with the corresponding blocks in the lower row, as described in Section III-B. This allows the corresponding memory resources to be shared between each pairing of blocks from the upper and lower rows. In the case of the LTE turbo decoder, memory resources are therefore required for storing N extrinsic LLRs, $MN = 8N$ forward state metrics and $MN = 8N$ backward state metrics. Memory resources are also required for storing $3N$ of the *a priori* LLRs provided by the demodulator, so that they can be used throughout the iterative decoding process.

In the case of the WiMAX turbo decoder, memory resources are required for storing $2N$ extrinsic LLRs, $MN = 8N$ forward state metrics, $MN = 8N$ backward state metrics and $6N$ *a priori* LLRs. Owing to the fully-parallel operation of the algorithm of Figure 1(b), all of the above-mentioned memory resources are accessed in every time period, implying a register resource requirement Y, as quantified in Table I. Note that if the pipelining technique of Section IV-C is employed for reducing the time period duration D of the LTE turbo decoder, then additional registers are required in order to store $\bar{b}_2^{\mathrm{u,a}} + \bar{b}_3^{\mathrm{u,a}}$ for each algorithmic block in the upper row, as shown in brackets in Table I.

Like the proposed fully-parallel turbo decoder algorithm, the Log-BCJR turbo decoder algorithm of Figure 1(c) requires memory for storing the *a priori* LLRs provided by the demodulator. More specifically, memory is required for storing $3N$ *a priori* LLRs in the case of the LTE turbo decoder and $6N$ *a priori* LLRs in the case of the WiMAX turbo decoder, as described above. Similarly to when an odd-even interleaver is employed in the proposed fully-parallel turbo decoder algorithm, the algorithmic blocks in the upper rows of the Log-BCJR turbo decoder algorithm are not operated simultaneously with the blocks in the lower rows. Owing to this, memory resources are only required for storing N extrinsic LLRs in the case of the LTE turbo decoder and $2N$ extrinsic LLRs in the case of the WiMAX turbo decoder, since these resources can be shared between each pairing of blocks from the upper and lower rows, as described above. Furthermore, this allows the upper and lower rows to share memory resources that are used for storing intermediate values generated during the forward recursion. More specifically, memory resources are required for storing the MKN *a priori* transition metrics of (8) and the MN forward state metrics of (9), so that they can be used during the backward recursion by (10) and (11), respectively. Since N time periods are used to successively activate the blocks within each forward or backward recursion, the above-mentioned memory resources correspond to N addresses within a RAM resource requirement Z, as quantified in Table I. Furthermore, this allows the memory reuse technique to be extended further, during the backwards recursions. More specifically, the M state metrics output by (10) in a particular time period only need to be stored for use as an input to (10) in the next time period. This allows same set of M memory resources to be reused in each successive time period of the backward recursion, corresponding to a register resource requirement Y, as shown in Table I.

The state-of-the-art turbo decoder algorithm of [10] has the same RAM resource requirement as the Log-BCJR algorithm for storing *a priori* and extrinsic LLRs. However, this RAM must be arranged into a small number of addresses, since the NSW technique and the Radix-4 transform use only a small number of consecutive time periods to access all LLRs in each set. For example, 48 addresses are accessed during 48 consecutive time periods in the case of the $N = 6144$ LTE turbo code [10]. Note that the Radix-4 transform halves the number of extrinsic state metrics that are generated in the LTE turbo code. Furthermore, the state-of-the-art algorithm uses a re-computation technique [10] to further reduce the RAM resource requirement for both the LTE and WiMAX turbo codes. Rather than storing the *a priori* transition metrics during the forward recursion, so that they can be reused during the backward recursion and vice versa, the re-computation technique simply recalculates these metrics when they are needed for a second time. In addition to this, the state-of-the-art LTE turbo decoder stores only $1/6$ of the extrinsic state metrics during the forward recursion and recalculates the other $5/6$ of these metrics during the backward recursion, and vice versa. However, the associated RAM must be arranged into an even smaller number of addresses, since it is only accessed in $1/6$ of the time periods. For example, eight addresses are required in the case of the $N = 6144$ LTE turbo code, since the extrinsic state metrics are only accessed in $1/6$ of the above-mentioned 48 time periods. Furthermore, these recalculations require a small amount of additional memory resource to complement each of the 64 sets of computational resources described in Section IV-D. More specifically, each set requires RAM resources having five addresses, which is used for storing the five groups of M recalculated extrinsic forward or backward state metrics. Additionally, RAM resources having six addresses are required for storing the six groups of the 15 possible non-zero values that can be adopted by the *a priori* transition metrics. Furthermore, register resources are required for storing 28 intermediate values that are produced by the pipelining of the computations performed within each set of computational resources, as described in Section IV-C. Table I quantifies the total register and RAM resource requirements Y and Z of the state-of-the-art turbo decoder.

F. Error correction performance

The Bit Error Ratio (BER) of the proposed fully-parallel turbo decoding algorithm is compared with that of the Log-BCJR algorithm in Figures 4 and 5. In each case, BPSK modulation is employed for transmission over an uncorrelated narrowband Rayleigh fading channel having a range of Signal to Noise Ratio (SNR) per bit E_b/N_0, where $E_b/N_0\ [\mathrm{dB}] = \mathrm{SNR}\ [\mathrm{dB}] - 10\log_{10}(R)$ in this case. In Figure 4, the algorithms are compared for the case of LTE turbo decoding using the exact $\max^*$ operator of (6), for frame lengths of $N \in \{48, 480, 4800\}$ and for various numbers of decoding iterations I. Figure 4 shows that regardless of the frame length N, the proposed fully-parallel algorithm can converge to the same error correction performance as the Log-BCJR algorithm. However, the proposed fully-parallel algorithm can be seen to converge relatively slowly, requiring significantly more decoding iterations I than the Log-BCJR algorithm. This is because information is not propagated from each algorithmic block to all others using forward and backward recursions in the proposed algorithm, which instead relies upon a sufficient number of decoding iterations to propagate information. Note the requirement for an increased number of decoding iterations is not unexpected, since LDPC decoders employing a parallel scheduling are known to require more decoding iterations than those employing a serial scheduling [19]. Figure 4 suggests that the number of decoding iterations I required by the Log-BCJR algorithm to achieve a particular BER is consistently

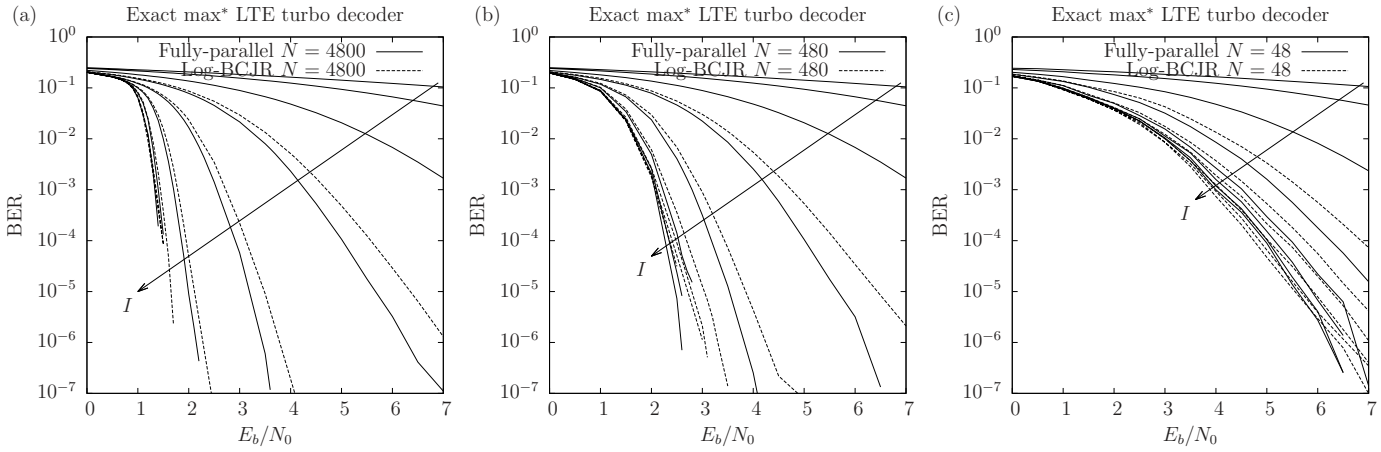

Fig. 4. The error correction performance of the LTE turbo decoder when using the exact $\max^*$ operator of (6) to decode frames comprising (a) $N = 4800$, (b) $N = 480$ and (c) $N = 48$ bits. Here, BPSK modulation is employed for transmission over an uncorrelated narrowband Rayleigh fading channel. Plots are provided for the case where $I \in \{1, 2, 4, 8, 16, 32, 64, 128\}$ decoding iterations are performed using the proposed fully-parallel algorithm, as well as $I \in \{1, 2, 4, 8, 16\}$ decoding iterations using the conventional Log-BCJR algorithm. MATLAB code for producing these plots is available at http://dx.doi.org/10.5258/SOTON/378330.

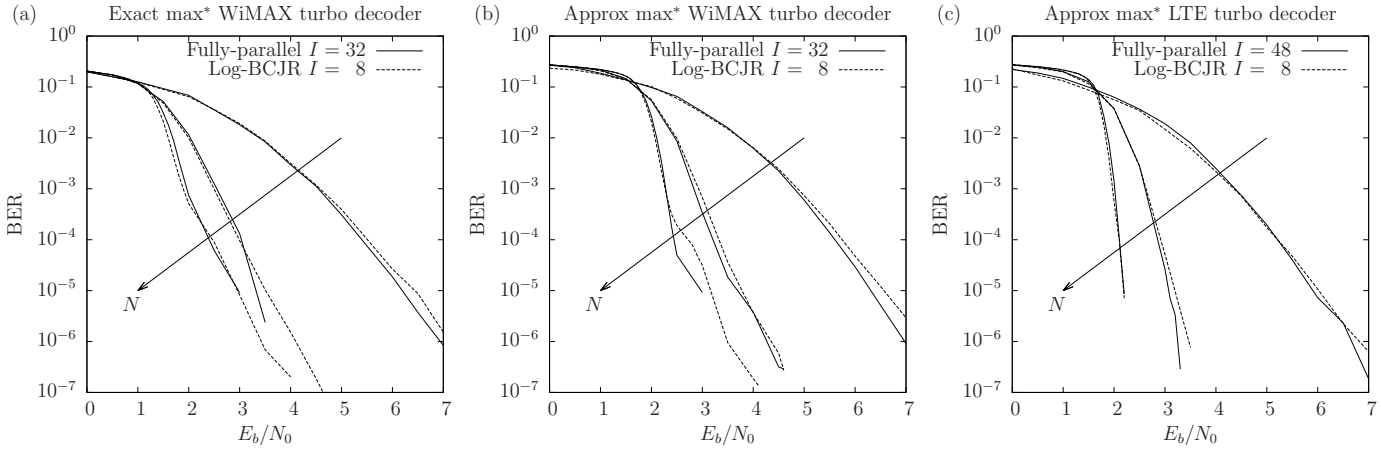

Fig. 5. The error correction performance of (a) the WiMAX turbo decoder when using the exact $\max^*$ operator of (6), (b) the WiMAX turbo decoder when using the approximate $\max^*$ operator of (7) and (c) the LTE turbo decoder when using the approximate $\max^*$ operator. Here, BPSK modulation is employed for transmission over an uncorrelated narrowband Rayleigh fading channel. Plots are provided for the case where $I = 32$ or $I = 48$ decoding iterations are performed using the proposed fully-parallel algorithm, as well as $I = 8$ decoding iterations using the conventional Log-BCJR algorithm. Frame lengths of $N \in \{48, 480, 4800\}$ are adopted for the LTE turbo code, while $N \in \{24, 240, 2400\}$ for the WiMAX turbo code. MATLAB code for producing these plots is available at http://dx.doi.org/10.5258/SOTON/378330.

around $1/7$ times that of the proposed algorithm, for the case of LTE turbo decoding using the exact $\max^*$ operator of (6). However, when employing the approximate $\max^*$ operator of (7), this number changes to $1/6$ times that of the proposed algorithm, as shown in Figure 5(c) and Table I. More specifically, Figure 5(c) shows that regardless of the frame length $N \in \{48, 480, 4800\}$, the Log-BCJR algorithm employing $I = 8$ decoding iterations achieves the same BER as the proposed fully-parallel algorithm employing $I = 48$ iterations. In the case of the WiMAX turbo code, Figures 5(a) and (b) reveal that the number of decoding iterations I required by the Log-BCJR algorithm is $1/4$ times that of the proposed algorithm, regardless of the frame length N and whether the exact or the approximate $\max^*$ operator is employed. Note that the error correction performance of the state-of-the-art algorithm of [10] is slightly degraded by its employment of the NSW technique, although this degradation can be considered to be insignificant. Therefore as shown in Table I, the number of decoding iterations I required by the state-of-the-art algorithm can also be considered to be $1/6$ and $1/4$ times that of the proposed algorithm, for the LTE and WiMAX turbo codes, respectively.

G. Overall characteristics

The latency $D \times T \times I$ of a turbo decoder is given by the product of the time period duration D, the number of time periods per decoding iteration T and the required number of decoding iterations I. Meanwhile, the processing throughput is inversely proportional to the latency $D \times T \times I$. For both LTE and WiMAX turbo decoding, Table I quantifies the latency and throughput of the proposed fully-parallel algorithm, the Log-BCJR algorithm and the state-of-the-art algorithm of [10]. In the case of an LTE turbo code employing the longest supported frame length of $N = 6144$ bits, the latency and

throughput of the proposed fully-parallel algorithm are more than three orders-of-magnitude superior to those of the Log-BCJR algorithm. Furthermore, when compared with the state-of-the-art algorithm of [10], the proposed fully-parallel algorithm has a latency and throughput that is 6.86 times superior. Note however that the advantage offered by the proposed fully-parallel algorithm is mitigated if the frame length N is reduced. In the case of the shortest frame length $N = 2048$ that is supported by the considered parametrization of the state-of-the-art algorithm's NSW technique, the superiority of the proposed fully-parallel algorithm is reduced to 2.29 times. When applying the state-of-the-art algorithm to the WiMAX turbo decoding of frames having lengths in the range $N \in [1440, 2400]$, the superiority of the proposed fully-parallel algorithm ranges from 3.75 times, up to 6.25 times. Compared to the Log-BCJR algorithm for WiMAX turbo decoding, the proposed fully-parallel algorithm is more than three orders-of-magnitude superior, when employing the maximum frame length of $N = 2400$.

The state-of-the-art LTE turbo decoder ASIC of [10] achieves a processing throughput of 2.15 Gbit/s and a latency of 2.85 μs, when decoding frames comprising $N = 6144$ bits. This is achieved using a clock frequency of 450 MHz, which corresponds to a time period duration of 2.22 ns. The results of Table I suggest that a fully-parallel turbo decoder ASIC could achieve a processing throughput of 14.7 Gbit/s and a latency of 0.42 μs, using a clock frequency of 194 MHz. Furthermore, it may be assumed that the state-of-the-art turbo decoder ASIC of [10] could maintain a processing throughput of 2.15 Gbit/s when applied for WiMAX decoding. If so, then this suggests that the a fully-parallel turbo decoder ASIC could achieve a processing throughput of 13.4 Gbit/s and a latency of 0.36 μs, when decoding frames having a length of $N = 2400$ bits. Note that these multi-gigabit throughputs are comparable to those that are offered by fully-parallel LDPC decoders [9].

While the proposed fully-parallel algorithm offers significant improvements to processing throughput and latency, this is achieved at the cost of requiring an increased computational complexity. The overall computational complexity $C \times I$ is given as the product of the computational complexity per decoding iteration C and the required number of decoding iterations I. For both LTE and WiMAX turbo decoding, Table I quantifies the overall computational complexity of the proposed fully-parallel algorithm, the Log-BCJR algorithm and the state-of-the-art algorithm of [10]. As shown in Table I, the computational complexity of the proposed fully-parallel algorithm can be more than five times higher than that of the Log-BCJR algorithm. Compared to the state-of-the-art algorithm of [10] however, the proposed fully-parallel algorithm has a computational complexity that is 2.90 and 2.18 times higher in the case of the LTE and WiMAX turbo codes, respectively.

Furthermore, the proposed fully-parallel algorithm has an increased overall resource requirement, which is given by combining the computational-, register- and RAM-resource requirements, according to $9X + 5Y + Z$. This is justified, since adders and registers can be implemented using nine and five NAND gates per bit, respectively. Meanwhile, RAM comprising a sufficiently high number of addresses can be implemented using similar hardware resources per bit as a NAND gate [20]. Indeed, the overall resource requirements of the proposed fully-parallel turbo decoder algorithm are significantly greater than those of the Log-BCJR and state-of-the-art algorithms, as shown in Table I. However, in order to achieve the same processing throughput as the proposed algorithm, several instances of the Log-BCJR and state-of-the-art algorithms could be operated in parallel. Note however that this approach assumes that several frames are available for simultaneous decoding and this also proportionally increases the associated resource requirements. When normalized by the processing throughput in this way, the overall resource requirements of the proposed fully-parallel turbo decoder algorithm are 80 and 42 times lower than those of the Log-BCJR algorithm, in the case of the $N = 6144$ LTE turbo code and the $N = 2400$ WiMAX turbo code, respectively. However, the normalized resource requirements of the proposed algorithm are 4.27 times higher than that of the state-of-the-art algorithm in the case of the $N = 6144$ LTE turbo code and 3.92 times higher in the case of the $N = 2400$ WiMAX turbo code. Note however that this comparison may be regarded as rather pessimistic, owing to a number of implementational issues that cannot be accurately captured by the algorithmic analysis of Table I. Firstly, the RAM resources employed in the state-of-the-art algorithm have only between 5 and 48 addresses. Since the overhead associated with the input and output hardware resources of the RAM is shared over so few addresses, the associated hardware resources per bit are significantly higher than those of a NAND gate. Owing to its employment of centralized RAM, the state-of-the-art turbo decoder implementation of [10] requires both a significantly larger controller and cache-memories than would be required to implement the proposed fully-parallel turbo decoder algorithm. In addition to this, the employment of the Radix-4 technique in the state-of-the-art algorithm requires it to employ a high number of bits to represent each state metric and LLR [10]. Finally, while operating several instances of the state-of-the-art turbo decoder in parallel is capable of improving its processing throughput, this does not improve its processing latency. So, while the overall computational complexity and normalized resource requirements of the proposed algorithm are higher than those of the state-of-the-art algorithm of [10], its key benefit is a processing latency which is up to 6.86 times superior. This trade-off is attractive in applications such as low-latency capital market trading, in which enormous profits are earned by financial institutions having the lowest latency wireless communication links to stock exchanges in different cities [21].

V. Conclusions

This paper has proposed a novel turbo decoding algorithm, which eliminates the data dependencies of the state-of-the-art algorithm and facilitates fully-parallel operation. The proposed fully-parallel algorithm is compatible with all turbo codes, including those of the LTE and WiMAX standards. These standardized turbo codes employ odd-even interleavers, facilitating a novel technique for reducing the complexity of

the proposed algorithm by 50%. More specifically, odd-even interleavers allow the proposed algorithm to alternate between processing the odd-indexed bits of the first component code at the same time as the even-indexed bits of the second component, and vice-versa. Furthermore, the proposed fully-parallel algorithm was shown to converge to the same error correction performance as the state-of-the-art turbo decoding algorithm, regardless of which turbo code it is applied for. Owing to its significantly increased parallelism, the proposed algorithm facilitates throughputs and latencies that are up to 6.86 times superior to those of the state-of-the art algorithm, when employed for the LTE and WiMAX turbo codes. Using these applications as examples, it may be expected that the proposed algorithm facilitates ASIC processing throughputs of up to 14.7 Gbit/s, as well as latencies as small as 0.42 μs, satisfying the requirements of next-generation wireless communication standards for the first time. However, this is achieved at the cost of a computational complexity and normalized resource requirement that are several times higher than those of the state-of-the-art algorithm, although these comparisons may be considered to be pessimistic, as described in Section IV-G.

Our future work will consider the design of fully-parallel turbo decoders that support multiple interleaver patterns. A fully-reconfigurable interleaver could be implemented using an arbitrary-size Beneš network [22]. For the $N = 6144$ LTE turbo code, the Beneš network would comprise $S(N) = 73728$ switches, where $S(N)$ is defined in [22]. In order to facilitate interleaving and deinterleaving, these switches would have to be bi-directional. Therefore, each switch would comprise four 2-to-1 multiplexers, which corresponds to 12 NAND gates per bit, as well as a single inverter that is shared by all bits. Ignoring the negligible contribution of the inverters, this would therefore increase the overall resource requirement from $(825 \cdot 6144)$ to $(825 \cdot 6144 + 12 \cdot 73728)$, which represents a 17.5% increase. However, our future work will exploit the regular structure of the LTE interleaver in order to minimize this multiplexing resource requirement. Furthermore, the Beneš network will be designed to allow different groups of algorithmic blocks to simultaneously decode different frames having any combination of different lengths and different interleaver patterns. Provided that there is a sufficient number of unassigned algorithmic blocks, the decoding of each frame can begin as soon as it is received, without interrupting the decoding of any frames already in progress. However, this will require a relatively complicated controller, which may be expected to have a resource requirement exceeding that of the above-mentioned switches, perhaps resulting in a 40% increase to the overall resource requirement, rather than a 17.5% increase. In an alternative approach to a Beneš network, we will allow the extrinsic state metrics to bypass some of the algorithmic blocks, at the cost of only a negligible multiplexing resource requirement. This will allow a hard-wired interleaver to be used for decoding shorter frames having particular compatible interleaver patterns. We will design a family of compatible interleaver patterns having similar lengths and error correction capabilities as the LTE interleaver. As a further extension to this work, we will enable full compatibility with the LTE interleaver by combining this bypass technique with a small Beneš network.

Our future work will also consider the practical ASIC, FPGA, NoC and GPGPU implementation of the proposed fully-parallel algorithm, in order to determine the processing throughputs and latencies that may be achieved in practice. Furthermore, this study will reveal how the proposed algorithm's increased computational complexity and resource requirements translate into energy consumption and chip area or hardware requirements. By jointly designing the proposed algorithm with its practical implementation, it may be expected that further innovations can be found which close the gap to the energy consumption and normalized chip area of the state-of-the-art design of [10], while maintaining the high processing throughput and low latency mentioned above.

REFERENCES

[1] *ETSI TS 136 212 LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and Channel Coding*, V12.0.0 ed., 2013.

[2] *IEEE 802.16-2012 Standard for Local and Metropolitan Area Networks - Part 16: Air Interface for Broadband Wireless Access Systems*, 2012.

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

[4] 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, June 1995, pp. 1009–1013.

[5] *IEEE 802.11n-2009 Standard for Information Technology - Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY)*, 2009.

[6] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 32, no. 18, pp. 457–458, Aug. 1996.

[7] M. Fossorier, "Reduced complexity decoding of low-density parity check codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, no. 5, pp. 673–680, May 1999.

[8] "5G Radio Access," Ericsson White Paper, Tech. Rep., June 2013.

[9] V. A. Chandrasetty and S. M. Aziz, "FPGA implementation of a LDPC decoder using a reduced complexity message passing algorithm," *Journal of Networks*, vol. 6, no. 1, pp. 36–45, Jan. 2011.

[10] 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, Aug. 2012, pp. 21–25.

[11] J. Zhang and M. P. C. Fossorier, "Shuffled iterative decoding," *IEEE Trans. Commun.*, vol. 53, no. 2, pp. 209–213, Feb. 2005.

[12] O. Muller, A. Baghdadi, and M. Jezequel, "Exploring parallel processing levels for convolutional turbo decoding," in *Int. Conf. Inform. Commun. Tech.*, Damascus, Syria, Apr. 2006, pp. 2353–2358.

[13] O. Muller, A. Baghdadi, and M. Jézéquel, "From parallelism levels to a multi-ASIP architecture for turbo decoding," *IEEE Trans. VLSI Syst.*, vol. 17, no. 1, pp. 92–102, Jan. 2009.

[14] L. Fanucci, P. Ciao, and G. Colavolpe, "VLSI design of a fully-parallel high-throughput decoder for turbo gallager codes," *IEICE Trans. Fundamentals*, vol. E89-A, no. 7, pp. 1976–1986, July 2006.

[15] D. Vogrig, A. Gerosa, A. Neviani, A. Graell I Amat, G. Montorsi, and S. Benedetto, "A 0.35-μm CMOS analog turbo decoder for the 40-bit rate 1/3 UMTS channel code," *IEEE J. Solid-State Circuits*, vol. 40, no. 3, pp. 753–762, 2005.

[16] Q. T. Dong, M. Arzel, C. J. Jego, and W. J. Gross, "Stochastic decoding of turbo codes." *IEEE Trans. Signal Processing*, vol. 58, no. 12, pp. 6421–6425, Dec. 2010.

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

[18] 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, no. 1, pp. 14–22, Jan. 2013. [Online]. Available: http://eprints.soton.ac.uk/271820/

[19] P. Radosavljevic, A. de Baynast, and J. R. Cavallaro, "Optimized message passing schedules for LDPC decoding," in *Asilomar Conf. Signals Systems and Computers*, no. 1, Pacific Grove, CA, USA, Oct. 2005, pp. 591–595.

[20] ARM, "65nm Fast Cache Instances User Guide," Apr. 2009, ARM PUG 0026E.

[21] D. Schneider, "The microsecond market," *IEEE Spectrum*, vol. 49, no. 6, pp. 66–81, June 2012.

[22] C. Chang and R. Melhem, "Arbitrary Size Benes Networks," *Parallel Processing Letters*, vol. 07, no. 03, pp. 279–284, 1997.

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.