IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 29, 2016, accepted November 19, 2016, date of publication November 29, 2016,
date of current version March 28, 2017.

Digital Object Identifier 10.1109/ACCESS.2016.2633441

A High-Throughput FPGA Architecture for Joint Source and Channel Decoding

MATTHEW F. BREJZA, ROBERT G. MAUNDER, BASHIR M. AL-HASHIMI, AND LAJOS HANZO, (Fellow, IEEE)
Department of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, U.K.

Corresponding author: L. Hanzo (lh@ecs.soton.ac.uk)

This work was supported in part by EPSRC, Swindon, U.K., under Grant EP/J015520/1 and Grant EP/L010550/1, and in part by TSB, Swindon, U.K., under Grant TS/L009390/1. The research data for this paper is available at http://dx.doi.org/10.5258/SOTON/403491

ABSTRACT In the wireless transmission of multimedia information, the achievable *transmission* throughput and latency may be limited by the *processing* throughput and latency associated with source and channel coding. Ultra-high throughput and ultra-low latency processing of source and channel coding are required by the emerging new video transmission applications, such as the first-person remote control of unmanned vehicles. The recently proposed unary error correction (UEC) code facilitates the joint source and channel coding (JSCC) of video information at transmission throughputs that approach the capacity of the wireless channel. In this paper, we propose the first hardware implementation of the UEC code that achieves the high processing throughputs as well as ultra-low processing latencies required. This is achieved by extending the application of the recently proposed fully parallel turbo decoder (FPTD) from pure stand-alone channel coding to JSCC. This paper also proposes several novel improvements to the FPTD, in order to increase its hardware efficiency and supported frame length. We demonstrate the application of these improvements to both the long term evolution turbo code and the UEC code. We synthesize the proposed fully parallel design on a mid-range field programmable gate array, achieving a throughput of 450 Mbps, as well as a factor of 2.4 hardware efficiency improvement over previous implementations of the FPTD.

INDEX TERMS Turbo codes, combined source-channel coding, channel coding, decoding, field programmable gate arrays, throughput, wireless communication.

I. INTRODUCTION

With the increasing application of high and ultra-high definition video, the demand for high throughput wireless communication systems is also increasing. Furthermore, low latency wireless communication is also required by many of these video applications such as the first-person remote control of unmanned vehicles, or mobile access to cloud-computing based video games. A high transmission throughput, and hence a low transmission latency, can be achieved by using near capacity transmission techniques to maximize the bandwidth efficiency. Near-capacity operation may be achieved using Shannon's Separate Source and Channel Coding (SSCC) concept [1]. Here, a near-entropy source code such as the arithmetic code [2] is used to remove redundancy from the source, in order to achieve a high degree of compression. Meanwhile, a separate near-capacity channel code, such as a turbo code [3], is used for introducing specifically selected redundancy to achieve a high degree of error correction. However, Shannon's SSCC concept assumes that infinite computational complexity and latency can be afforded. Indeed, arithmetic codes are only capable of removing all redundancy for the sake of achieving near-entropy compression when they operate on long sequences of source symbols, imposing a latency bottleneck. Meanwhile, the conventional approach to turbo decoding employs the serial Logarithmic Bahl-Cocke-Jelinek-Raviv (Log-BCJR) algorithm [4], imposing a throughput limitation due to its limited grade of parallelism and iterative nature, even when using hardware acceleration. These factors limit the overall throughput and latency, precluding the high-throughput and low-latency applications described above.

This motivates Joint Source and Channel Coding (JSCC) [5], which can offer performance gains compared to conventional SSCC. In contrast to SSCC, a JSCC scheme does not attempt to remove all of the redundancy from the encoded source symbols using sophisticated compression techniques. Instead, a JSCC scheme uses the residual redundancy that remains after compression for the purpose of error correction. As a result, a JSCC scheme can encode frames of any length, while maintaining

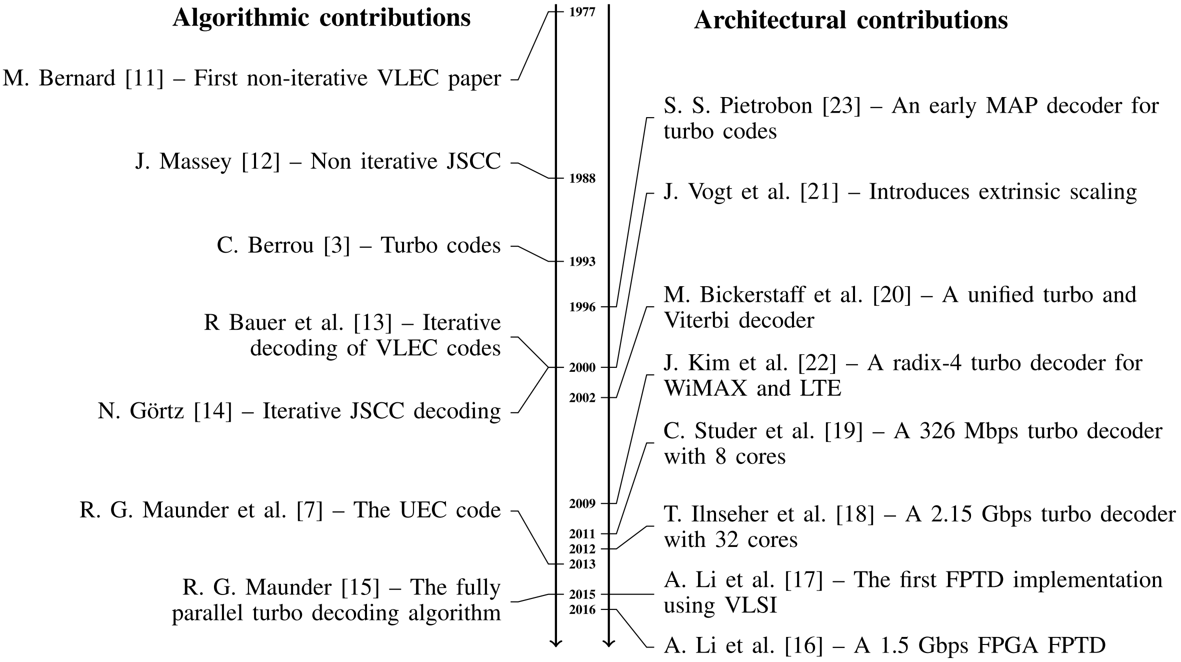

Algorithmic contributions
Architectural contributions
1977
1988
1993
1996
2000
2002
2009
2011
2012
2013
2015
2016
M. Bernard [11] – First non-iterative VLEC paper
J. Massey [12] – Non iterative JSCC
C. Berrou [3] – Turbo codes
R Bauer et al. [13] – Iterative decoding of VLEC codes
N. Görtz [14] – Iterative JSCC decoding
R. G. Maunder et al. [7] – The UEC code
R. G. Maunder [15] – The fully parallel turbo decoding algorithm
S. S. Pietrobon [23] – An early MAP decoder for turbo codes
J. Vogt et al. [21] – Introduces extrinsic scaling
M. Bickerstaff et al. [20] – A unified turbo and Viterbi decoder
J. Kim et al. [22] – A radix-4 turbo decoder for WiMAX and LTE
C. Studer et al. [19] – A 326 Mbps turbo decoder with 8 cores
T. Ilnseher et al. [18] – A 2.15 Gbps turbo decoder with 32 cores
A. Li et al. [17] – The first FPTD implementation using VLSI
A. Li et al. [16] – A 1.5 Gbps FPGA FPTD

FIGURE 1. The key algorithmic and architectural contributions in source and channel coding.

near-capacity operation. Previous JSCC schemes, such as the Variable Length Error-Correction (VLEC) code [6], have a high complexity, owing to the large alphabet from which the source symbols are selected. By contrast, a Unary Error Correction (UEC) code [7]–[10] concatenated with a Unity Rate Convolutional (URC) code, yielding the UEC-URC code, offers near-capacity operation at a low complexity, even for large source alphabets such as those of video sources.

While the UEC-URC coding and its derivatives have received a significant amount of attention at the algorithmic level, a hardware implementation of UEC-URC coding has not been previously considered. In particular, an implementation of UEC-URC coding having a high processing throughput and a low processing latency is required, in order to avoid imposing a bottleneck upon the achievable transmission throughput and latency. This is a particular challenge, since UEC-URC decoding has been previously based on the Log-BCJR algorithm, which suffers from serial data dependencies that are not conducive to high throughput and low latency processing. In order to address the bottleneck imposed by the serial data dependencies of conventional Log-BCJR turbo decoders, our previous work [17] has considered the hardware implementation of turbo decoders having high throughput and low latency. More specifically, our Fully Parallel Turbo Decoder (FPTD) [15] achieves a high processing throughput and a low processing latency by eliminating the serial data dependencies of the conventional Log-BCJR approach, allowing the decoding of all bits to be completed at the same time, in a fully parallel manner, albeit at the cost of requiring a large Application-Specific Integrated Circuit (ASIC) or Field Programmable Gate Array (FPGA) area. These key contributions on JSCC algorithms and turbo decoder architectures are shown on Figure 1.

Our new contribution is that we extend the principle of the FPTD to the implementation of a UEC-URC scheme, which achieves for the first time near-capacity JSCC at a high processing throughput and a low processing latency, hence meeting the requirements described above. Furthermore, we propose several novel techniques for improving the chip-area efficiency of the FPTD approach. This is achieved by improving the fully parallel algorithm to allow a pair of bits to be decoded using the same hardware processing element. This also facilitates improved pipelining for the sake of increasing the clock frequency, as well as for reducing the number of decoding iterations required. Since the UEC-URC scheme comprises both UEC and URC decoding components, our novel hardware processing elements are designed to be capable of processing both of these different decoders, hence facilitating an efficient hardware design.

The remainder of this paper is structured as shown in Figure 2. In Section II, we describe the LTE turbo code and the UEC-URC scheme. Both the LTE turbo decoder and UEC-URC decoder will be considered throughout the paper. By considering the turbo code alongside the UEC-URC scheme, we can compare the architecture proposed on this paper with previous implementations of the turbo code. Section II concludes by describing the fully parallel decoding algorithm introduced in [15]. Following this, Section III proposes a new paired activation order for the blocks of the fully parallel decoding algorithm, using Bit Error Rate (BER) and Symbol Error Ratio (SER) simulations to quantify the number of iterations required to match the

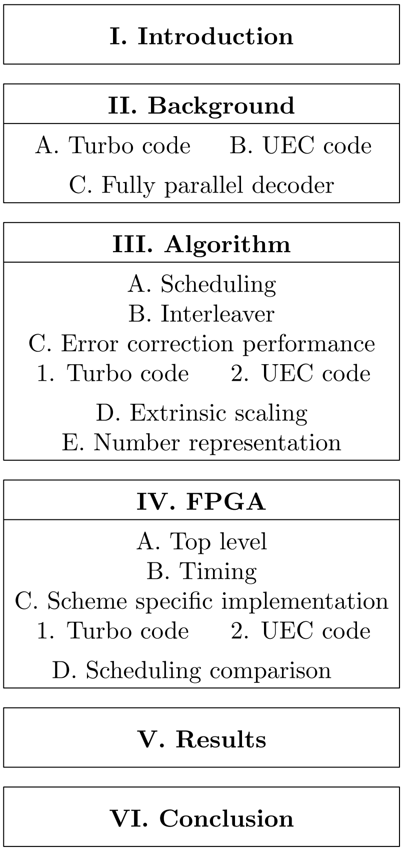


FIGURE 2. The paper outline.

error correction capability of the existing Log-BCJR and FPTD algorithms. Following this, Section IV uses this paired activation order as the basis of an FPGA implementation of the UEC-URC scheme. Section V details the implementation results, where we achieve 450 Mbps on a mid-range FPGA, demonstrating applicability to video applications. We also compare our proposed LTE implementation to previous fully parallel LTE turbo decoder implementations, demonstrating a 2.4-fold hardware efficiency improvement. Finally, Section VI offers our conclusions.

II. BACKGROUND

In this section, we commence by describing the operation of the LTE turbo code of Figure 3 in Section II-A. Following this, Section II-B highlights the operation of the UEC-URC scheme of Figure 5. Finally, Section II-C describes the fully parallel decoding algorithm, which will be applied to both the LTE turbo code and to the UEC-URC scheme.

A. LTE TURBO CODE SCHEME

In this section we describe the operation of the turbo code used in LTE. We commence by describing the encoder in Section II-A1, followed by the decoder in Section II-A2.

1) ENCODER

As shown in Figure 3a, the LTE turbo encoder [24] is comprised of two convolutional encoders [25], namely the upper and lower encoders. The input bit-vector $\mathbf{b}_1^\mathrm{u} = [b_{1,k}^\mathrm{u}]_{k=1}^N$ comprises N bits and typically has equiprobable bit values. This bit-vector is encoded by the upper decoder to give the encoded bit-vector $\mathbf{b}_2^\mathrm{u} = [b_{2,k}^\mathrm{u}]_{k=1}^N$, as well as the systematic bit-vector $\mathbf{b}_3^\mathrm{u} = [b_{3,k}^\mathrm{u}]_{k=1}^N$, which is identical to $\mathbf{b}_1^\mathrm{u}$. Meanwhile, an interleaver π_1 is used to reorder the input bits of $\mathbf{b}_1^\mathrm{u}$ to give the bit-vector $\mathbf{b}_1^\mathrm{l} = [b_{1,k}^\mathrm{l}]_{k=1}^N$, which is encoded by the lower encoder to give the bit-vector $\mathbf{b}_2^\mathrm{l} = [b_{2,k}^\mathrm{l}]_{k=1}^N$. In this way, the LTE turbo code has a coding rate of $R = 1/3$, since each input bit is encoded using three output bits, as shown in figure 3. Note that in the following discussion, the superscripts $^\mathrm{u}$ and $^\mathrm{l}$ will be omitted from the notation when the discussion applies equally to both the upper and lower encoders.

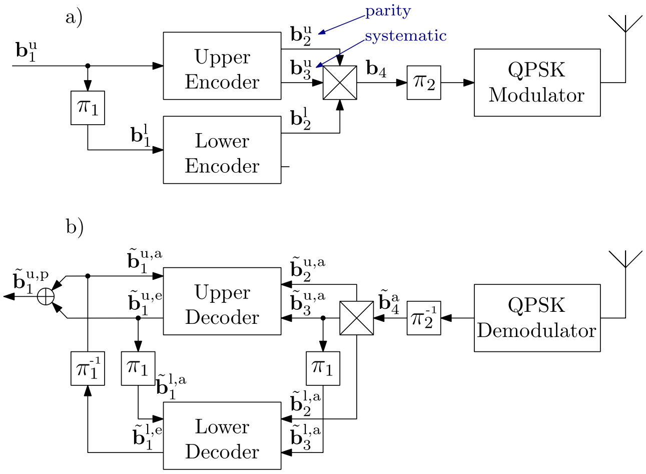

a)
$\mathbf{b}_1^\mathrm{u}$
Upper Encoder
$\mathbf{b}_2^\mathrm{u}$
parity
systematic
$\mathbf{b}_3^\mathrm{u}$
$\mathbf{b}_4$
π_2
QPSK Modulator
π_1
$\mathbf{b}_1^\mathrm{l}$
Lower Encoder
$\mathbf{b}_2^\mathrm{l}$
b)
$\tilde{\mathbf{b}}_1^\mathrm{u,a}$
$\tilde{\mathbf{b}}_1^\mathrm{u,p}$
$\tilde{\mathbf{b}}_1^\mathrm{u,e}$
Upper Decoder
$\tilde{\mathbf{b}}_2^\mathrm{u,a}$
$\tilde{\mathbf{b}}_3^\mathrm{u,a}$
$\tilde{\mathbf{b}}_4^\mathrm{a}$
π_2^{-1}
QPSK Demodulator
π_1^{-1}
π_1
$\tilde{\mathbf{b}}_1^\mathrm{l,a}$
π_1
$\tilde{\mathbf{b}}_2^\mathrm{l,a}$
$\tilde{\mathbf{b}}_1^\mathrm{l,e}$
Lower Decoder
$\tilde{\mathbf{b}}_3^\mathrm{l,a}$

FIGURE 3. The LTE turbo encoder and decoder. The interleaver π_2 is beneficial in the case where QPSK modulation is used for communication over a Rayleigh fading channel.

Figure 4 shows the trellis of the LTE convolutional encoders. The trellis characterizes the transition between the $r_\mathrm{LTE} = 8$ possible states of the encoder, based on its input bit-vector $\mathbf{b}_1$. At the beginning of the encoding process, each encoder starts from state $m_0 = 1$. The bits of $\mathbf{b}_1$ are considered in the order of increasing bit index k. Given any particular previous state m_{k-1}, the value of the input bit $b_{1,k}$ will trigger a state-transition to a state m_k selected from one of two potential options, as shown by the transition labels in Figure 4. The transitions between the states m_k corresponding to the successive input bits $b_{1,k}$ form a path $\mathbf{m} = [m_k]_{k=0}^N$ through the trellis. Since each input bit $b_{1,k}$ has equiprobable values, the transitions are also equiprobable. For each transition selected, an output bit $b_{2,k}$ is also identified. These parity bits are concatenated together to form the parity bit-vector $\mathbf{b}_2 = [b_{2,k}]_{k=1}^N$, mentioned above. For example, given the input bit-vector $\mathbf{b}_1 = [1000111011100100]$ comprising $N = 16$ bits, the convolutional encoding selects the $N + 1 = 17$ states $\mathbf{m} = [1, 5, 3, 6, 7, 4, 6, 3, 6, 3, 2, 1, 1, 1, 5, 3, 6]$, which yields the parity bit-vector $\mathbf{b}_2 = [1111100100100111]$.

Note that the LTE turbo encoder also appends three trellis-termination bits to the end of each of the

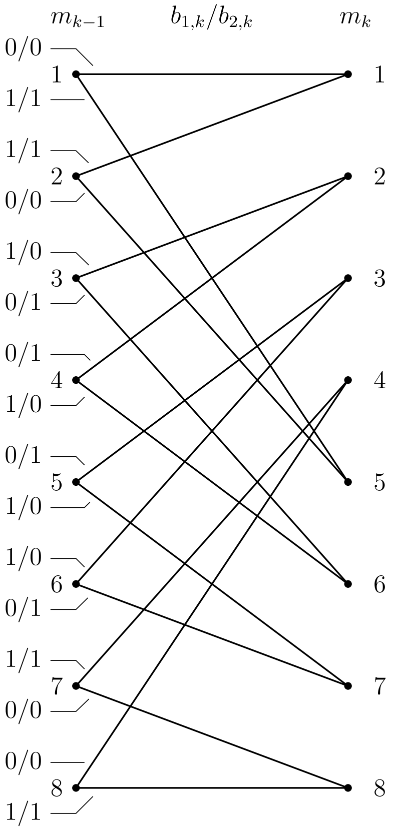


FIGURE 4. Trellis for the LTE turbo code.

bit-vectors $\mathbf{b}_1^{\text{u}}$, $\mathbf{b}_1^{\text{l}}$, $\mathbf{b}_2^{\text{u}}$ and $\mathbf{b}_2^{\text{l}}$, in order to guarantee that the end state of each convolutional encoder is $m_{N+3} = 1$ [24], which avoids an error floor [26]. These 12 termination bits are also output by the turbo encoder, but are not shown in Figure 3 for the sake of simplicity. Following turbo encoding, the resultant output bits are multiplexed and then modulated as well as transmitted over the channel. Since in this work we are assuming QPSK modulation and an uncorrelated narrowband Rayleigh fading channel [27], no channel-interleaving is required, but the interleaver π_2 is beneficial before modulation to nonetheless ensure that neighboring bits are not transmitted as pairs within the QPSK symbols.

2) DECODER

The LTE turbo decoder is shown in Figure 3b, where the upper and lower decoders correspond to the upper and lower encoders of the LTE turbo encoder. Likewise, the demodulator of Figure 3b mirrors the modulator of Figure 3a. While the encoder works on the basis of hard bits, having the values either 0 or 1, the demodulator and decoder uses soft bits called Logarithmic Likelihood Ratios (LLRs) [28], which express the decoder's uncertainty in the bit value owing to the noise in the channel. More specifically, the LLR value is given according to $\tilde{b} = \ln \frac{P(b=1)}{P(b=0)}$, where a high positive valued LLR represents a high confidence that the corresponding bit in the encoder was one-valued, while a negative valued LLR represents greater probability of a zero-valued bit. The use of LLRs allows the two decoders in the receiver to iteratively exchange information about their confidence in the various bit values, yielding improved decoding performance. Following their reception, the LLRs gleaned from the demodulator are de-interleaved by π_2^{-1}, then de-multiplexed, yielding the *a priori* LLR-vectors $\tilde{\mathbf{b}}_2^{\text{u,a}} = [\tilde{b}_{2,k}^{\text{u,a}}]_{k=1}^N$, $\tilde{\mathbf{b}}_2^{\text{l,a}} = [\tilde{b}_{2,k}^{\text{l,a}}]_{k=1}^N$ and $\tilde{\mathbf{b}}_3^{\text{u,a}} = [\tilde{b}_{3,k}^{\text{u,a}}]_{k=1}^N$. The systematic LLR-vector $\tilde{\mathbf{b}}_3^{\text{u,a}}$ is also interleaved through π_1 to yield $\tilde{\mathbf{b}}_3^{\text{l,a}} = [\tilde{b}_{3,k}^{\text{l,a}}]_{k=1}^N$. Furthermore, the lower decoder provides the upper decoder with the *a priori* LLR-vector $\tilde{\mathbf{b}}_1^{\text{u,a}} = [\tilde{b}_{1,k}^{\text{u,a}}]_{k=1}^N$, which is populated with zero-valued LLRs at the start of the decoding process. Likewise, the upper decoder provides the lower decoder with $\tilde{\mathbf{b}}_1^{\text{l,a}} = [\tilde{b}_{1,k}^{\text{l,a}}]_{k=1}^N$, as shown in Figure 3a.

In a conventional turbo decoder, the upper and lower decoders employ the Log-BCJR algorithm [4] for converting the input *a priori* LLR-vectors into the extrinsic output LLR-vectors $\tilde{\mathbf{b}}_1^{\text{u,e}} = [\tilde{b}_{1,k}^{\text{u,e}}]_{k=1}^N$ and $\tilde{\mathbf{b}}_1^{\text{l,e}} = [\tilde{b}_{1,k}^{\text{l,e}}]_{k=1}^N$. Note that the Log-BCJR algorithm can also beneficially exploit the 12 LLRs provided by the demodulator to correspond to the 12 termination bits. In these conventional turbo decoders, the upper and lower decoders are operated alternately, in an iterative manner. More specifically, the upper decoder outputs $\tilde{\mathbf{b}}_1^{\text{u,e}}$, which is interleaved through π_1 to become the *a priori* LLR-vector $\tilde{\mathbf{b}}_1^{\text{l,a}}$, which is input to the lower decoder. Likewise, the lower decoder outputs the extrinsic LLR-vector $\tilde{\mathbf{b}}_1^{\text{l,e}}$, which is de-interleaved through π_1^{-1} and input as the *a priori* LLR-vector $\tilde{\mathbf{b}}_1^{\text{u,a}}$ into the upper decoder. At the same time, the turbo decoder outputs the vector of *a posteriori* LLRs $\tilde{\mathbf{b}}_1^{\text{u,p}} = [\tilde{\mathbf{b}}_{1,k}^{\text{u,p}}]_{k=1}^N$, which is obtained by the addition of $\tilde{\mathbf{b}}_1^{\text{u,a}}$ and $\tilde{\mathbf{b}}_1^{\text{u,e}}$, and so represents the combined knowledge of the bit-vector $\mathbf{b}_1^{\text{u}}$.

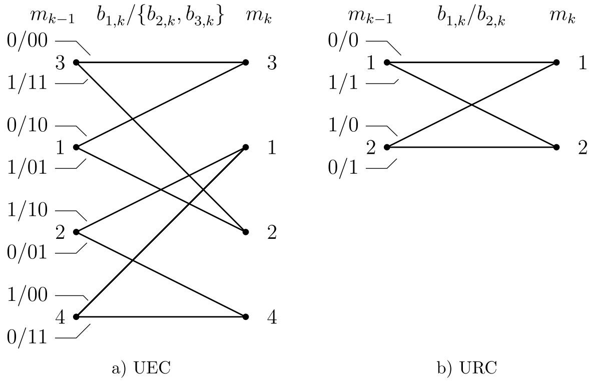
If the channel Signal-to-Noise Ratio (SNR) is sufficiently high, the quality of LLRs may be expected to increase in each successive iteration, as the decoder recovers the original encoded message. The iterations exchanging extrinsic information between the decoders continue until a fixed number of iterations is completed, or until a hard decision based on the *a posteriori* LLR-vectors satisfies the classic Cyclic Redundancy Check (CRC).

B. UEC-URC JSCC SCHEME

In this section, we detail the operation of the UEC-URC JSCC scheme. First, in Section II-B1, we describe how the scheme encodes and transmits a stream of symbols. Following this, Section II-B2 describes the operation of the UEC-URC decoder, which attempts to recover the original symbols. In this section, we adopt notation that is consistent with the turbo scheme of Section II-A.

1) ENCODER

The UEC-URC encoder is shown in Figure 5. This is a JSCC scheme, which operates on the basis of a stream of symbols $\mathbf{x} = [x_i]$, rather than bits. The UEC encoder is well suited to the encoding of symbols that obey Zipf's law [29], such



a) UEC b) URC

FIGURE 5. The UEC-URC scheme.

as the motion vectors and transform coefficients of the H.265 standard video codec [30], which may be modeled by the zeta symbol probability distribution. More specifically, the UEC encodes a stream of source symbols, which are assumed to be a realization of a stream of Independent and Identically Distributed (IID) Random Variables (RVs) $\mathbf{X} = [X_i]$, where each RV is selected from the set $\mathbb{N}_1 = \{1, 2, 3, 4, ..., \infty\}$, according to the zeta probability distribution which is given by [7]

$$P(X_i = x) = P(x) = \frac{x^{-s}}{\sum_{\hat{x}\in\mathbb{N}_1} \hat{x}^{-s}} = \frac{x^{-s}}{\zeta(s)}, \qquad (1)$$

where $\zeta(s) = \sum_{x\in\mathbb{N}_1} x^{-s}$ is the Riemann zeta function. In the zeta distribution, symbols having the value 1 are the most probable, where this probability is given by $p_1 = Pr(X_i = 1)$, and the probability of a symbol value decreases as the symbol value increases. Here, the variable $s > 1$ is related to the parameter p_1 according to $p_1 = \Pr(X_i = 1) = 1/\zeta(s)$. The entropy of the source symbols is given by [7]

$$H_X = \sum_{x\in\mathbb{N}_1} P(x) \log_2 \frac{1}{P(x)} = \frac{\ln(\zeta(s))}{\ln(2)} - \frac{s\zeta'(s)}{\ln(2)\zeta(s)} \qquad (2)$$

where $\zeta'(s) = -\sum_{x\in\mathbb{N}_1} \ln(x)x^{-s}$.

TABLE 1. The unary codewords.

x_i	Unary(x_i)
1	1
2	01
3	001
4	0001
5	00001
6	000001
7	0000001
⋮	⋮

The unary encoder of Figure 5 converts each symbol x_i to a codeword denoted by Unary(x_i), as shown in Table 1. The length of each unary codeword is equal to the value of the corresponding symbol x_i, where the first $(x_i - 1)$ bits in the codeword have the value 0, while the last bit has the value of 1. Since the length of each unary encoded symbol is equal to its value, we can express the average unary codeword length l_{Unary} as [7]

$$l_{\text{Unary}} = \sum_{x\in\mathbb{N}_1} P(x)x = \frac{\zeta(s-1)}{\zeta(s)}. \qquad (3)$$

In contrast to turbo coding, the bit values output by a unary encoder are not equiprobable. A further difference with respect to turbo coding is that for a fixed number of input symbols, unary source coding yields a variable number of bits. This necessitates a mechanism for partitioning the bits output from the unary encoder into fixed length bit-vectors. More specifically, the stream of codewords produced by the unary encoder are concatenated together, then partitioned into a succession of bit-vectors $\mathbf{b}_1^{\text{UEC}} = [b_{1,k}^{\text{UEC}}]_{k=1}^{N}$, having a fixed length of N bits. Owing to the partitioning, some unary codewords may be split between successive bit-vectors, hence a buffer is used for storing these bits so they can be removed from the end of one bit-vector and concatenated onto the start of the next, as shown in Figure 5. For example, the unary encoding of the symbol vector $\mathbf{x} = [1, 2, 1, 1, 4, 1, 3, 1, 2]$ associated with $N = 8$ produces the successive bit-vectors $\mathbf{b}_1^{\text{UEC}} = [1, 0, 1, 1, 1, 0, 0, 0]$ and $\mathbf{b}_1^{\text{UEC}} = [1, 1, 0, 0, 1, 1, 0, 1]$. Note that the fifth element of $\mathbf{x}$ is split between the two bit-vectors of $\mathbf{b}_1^{\text{UEC}}$.

As shown in Figure 5, the bit-vector $\mathbf{b}_1^{\text{UEC}}$ is encoded by the trellis encoder, yielding the bit-vectors $\mathbf{b}_2^{\text{UEC}} = [b_{2,k}^{\text{UEC}}]_{k=1}^{N}$ and $\mathbf{b}_3^{\text{UEC}} = [b_{3,k}^{\text{UEC}}]_{k=1}^{N}$. This encoding is performed according to the $r_{\text{UEC}} = 4$-state trellis of Figure 6a, in a manner similar to the convolutional encoding described in Section II-A1. Figure 6a shows how the trellis transitions from state m_{k-1} to m_k, depending on the input bit $b_{1,k}^{\text{UEC}}$. The transitions are also labeled with the corresponding codeword $\{b_{2,k}^{\text{UEC}}, b_{3,k}^{\text{UEC}}\}$ which is output when each transition is selected. The trellis encoder starts at state $m_0 = 1$ at the beginning of each bit-vector $\mathbf{b}_1^{\text{UEC}}$. The path taken through the trellis encoder upon encoding $\mathbf{b}_1^{\text{UEC}}$ may be represented as $\mathbf{m_k} = [m]_{k=0}^{N}$, comprising $N + 1$ state values. We can model the path $\mathbf{m}$ as a realization of a random vector of RVs $\mathbf{M} = [M_k]_{k=0}^{N}$, where the conditional probability of each state being selected $\Pr(M_k = m|M_{k-1} = m') = P(m|m')$ can be found in [7, eq. (9)]. These conditional transition probabilities $P(m|m')$ can be used to aid the receiver, as it will be described in Section II-B2. In contrast to the turbo code, since the bit values of $\mathbf{b}_1^{\text{UEC}}$ are not equiprobable, the transitions $P(m|m')$ are not equiprobable either.

Note that the trellis of Figure 6a can be readily extended to more states, which marginally improves the error correction performance, at the cost of imposing extra complexity [7]. Here we have chosen an $r_{\text{UEC}} = 4$-state UEC trellis, since this provides reasonable performance, while allowing an efficient hardware implementation, as we will demonstrate in Section IV. Note that the structure of the UEC trellis is based upon the unary codewords. For each zero-valued unary-encoded bit that is input to the trellis encoder, the transitions

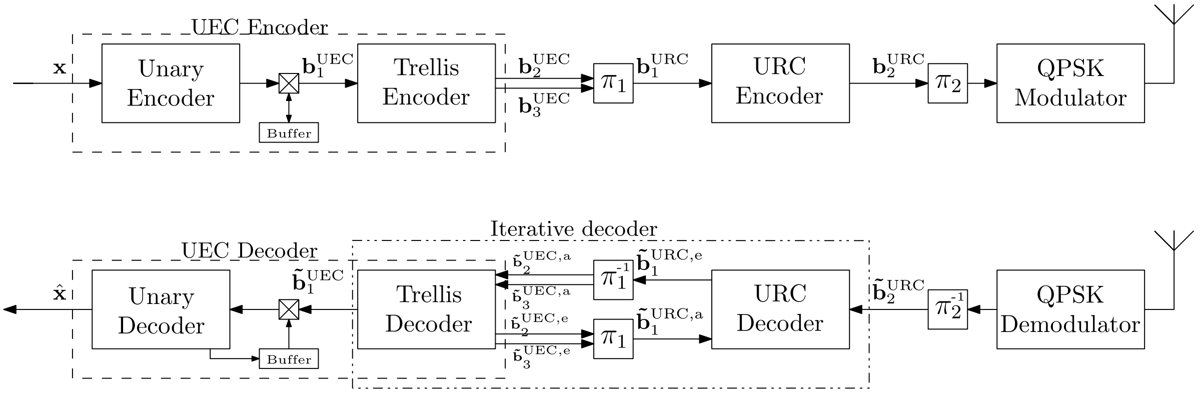

UEC Encoder
$\mathbf{x}$
Unary Encoder
$\mathbf{b}_1^{\text{UEC}}$
Buffer
Trellis Encoder
$\mathbf{b}_2^{\text{UEC}}$
$\mathbf{b}_3^{\text{UEC}}$
π_1
$\mathbf{b}_1^{\text{URC}}$
URC Encoder
$\mathbf{b}_2^{\text{URC}}$
π_2
QPSK Modulator
Iterative decoder
UEC Decoder
$\hat{\mathbf{x}}$
Unary Decoder
$\tilde{\mathbf{b}}_1^{\text{UEC}}$
Buffer
Trellis Decoder
$\tilde{\mathbf{b}}_2^{\text{UEC,a}}$
$\tilde{\mathbf{b}}_3^{\text{UEC,a}}$
$\tilde{\mathbf{b}}_2^{\text{UEC,e}}$
$\tilde{\mathbf{b}}_3^{\text{UEC,e}}$
π_1^{-1}
π_1
$\tilde{\mathbf{b}}_1^{\text{URC,e}}$
$\tilde{\mathbf{b}}_1^{\text{URC,a}}$
URC Decoder
$\tilde{\mathbf{b}}_2^{\text{URC}}$
π_2^{-1}
QPSK Demodulator

FIGURE 6. Trellis for UEC and URC codes. a) UEC. b) URC.

are taken towards the edge of the trellis. When the edge of the trellis is reached, further zeros keep the encoder in the holding state. When the one-valued bit at the end of each unary codeword is input, the encoder traverses to one of the central states. This means that the trellis ensures synchronization between the trellis and the codewords, without requiring an excessive number of states in the trellis. Since the UEC trellis is symmetric and employs complementary codewords in the top and bottom halves, the UEC-encoded output bits $b_{2,k}^{\text{UEC}}$ and $b_{3,k}^{\text{UEC}}$ generated by the trellis encoder are guaranteed to have equiprobable values.

The bits $b_{2,k}^{\text{UEC}}$ and $b_{3,k}^{\text{UEC}}$ output by the trellis encoder are concatenated for forming the vectors $\mathbf{b}_2^{\text{UEC}} = [b_{2,k}^{\text{UEC}}]_{k=1}^{N}$ and $\mathbf{b}_3^{\text{UEC}} = [b_{3,k}^{\text{UEC}}]_{k=1}^{N}$. For example, given the vector comprising $N = 15$ bits $\mathbf{b}_1^{\text{UEC}} = 011001010010111$, the path through the trellis can be expressed as a vector $\mathbf{m} = [1, 3, 2, 1, 3, 3, 2, 4, 1, 3, 3, 2, 4, 1, 2, 1]$ of $N + 1 = 16$-states. This path through the trellis encoder produces the output vectors $\mathbf{b}_2^{\text{UEC}} = [111101001010001]$ and $\mathbf{b}_3^{\text{UEC}} = [010001100011010]$, each comprising $N = 15$ bits.

Following trellis encoding, the bit-vectors $\mathbf{b}_2^{\text{UEC}}$ and $\mathbf{b}_3^{\text{UEC}}$ are concatenated and interleaved through an interleaver π_1, similar to the one used in the turbo code, producing the bit-vector $\mathbf{b}_1^{\text{URC}} = [b_{1,k}^{\text{URC}}]_{k=1}^{2N}$. An $r_{\text{URC}} = 2$-state URC encoder is employed to accumulate the bits of $\mathbf{b}_1^{\text{URC}}$, in order to generate the bit-vector $\mathbf{b}_2^{\text{URC}} = [b_{2,k}^{\text{URC}}]_{k=1}^{2N}$. This accumulation is equivalent to performing encoding using the trellis shown in Figure 6b. This URC code will facilitate iterative decoding exchanging extrinsic information with the UEC trellis code in receiver, for the sake of allowing near-capacity operation, as discussed in Section II-B2. It was shown in [31] that a 2-state URC code has more complementary EXIT characteristics to the UEC trellis encoder, compared to a 4- or 8-state URC encoder. The 2-state URC encoder also has a lower complexity, which will be exploited in Section IV. The URC encoder operates in a similar manner to the convolutional encoder described in Section II-A1. The input bits $b_{1,k}^{\text{URC}}$ are processed in order of increasing index k, where each bit causes the trellis of Figure 6b to transition from the previous state m_{k-1} to the next state m_k, while outputting the associated bit $b_{2,k}^{\text{URC}}$. The path comprising $2N + 1$ states taken by the encoder may be represented as $\mathbf{m} = [m_k]_{k=0}^{2N}$. Since the bits input to the URC encoder have equiprobable values, the bits output from the URC encoder also have equiprobable values. In contrast to the LTE turbo code, no termination is used by the URC trellis. Following URC encoding, the bit-vector $\mathbf{b}_2^{\text{URC}}$ is interleaved by π_2, before being Quadrature Phase Shift Keying (QPSK) modulated and transmitted over the Rayleigh fading channel.

2) DECODER

As shown in Figure 5b, the LLR-vector $\tilde{\mathbf{b}}_2^{\text{a,URC}} = [\tilde{b}_{2,k}^{\text{a,URC}}]_{k=1}^{2N}$ is obtained after QPSK demodulation and de-interleaving by π_2^{-1}. This is entered into the iterative decoder, which is comprised of a URC decoder and a UEC trellis decoder, in correspondence to the URC encoder and UEC trellis encoder in the transmitter. At the start of the decoding process, all other LLR-vectors are populated with zero values. More specifically, the URC decoder is provided with the encoded *a priori* LLR-vector $\tilde{\mathbf{b}}_2^{\text{a,URC}}$ from the demodulator, as well as the uncoded *a priori* LLR-vector $\tilde{\mathbf{b}}_1^{\text{a,URC}} = [\tilde{b}_{1,k}^{\text{a,URC}}]_{k=1}^{2N}$ provided by the trellis decoder. Likewise, the UEC trellis decoder is provided with the *a priori* LLR-vectors $\tilde{\mathbf{b}}_2^{\text{a,UEC}} = [\tilde{b}_{2,k}^{\text{a,UEC}}]_{k=1}^{N}$ and $\tilde{\mathbf{b}}_3^{\text{a,UEC}} = [\tilde{b}_{3,k}^{\text{a,UEC}}]_{k=1}^{N}$ by the URC decoder. In a conventional receiver, the URC decoder may employ the Log-BCJR decoder to transform the *a priori* input LLR-vectors into the extrinsic output LLR-vector $\tilde{\mathbf{b}}_1^{\text{e,URC}} = [\tilde{b}_{1,k}^{\text{e,URC}}]_{k=1}^{2b}$, according to the trellis of Figure 6b. Likewise, the UEC trellis decoder may employ the Log-BCJR algorithm to transform the input *a priori* LLR-vectors into the extrinsic LLR-vectors $\tilde{\mathbf{b}}_2^{\text{e,UEC}} = [\tilde{b}_{2,k}^{\text{e,UEC}}]_{k=1}^{N}$ and $\tilde{\mathbf{b}}_3^{\text{e,UEC}} = [\tilde{b}_{3,k}^{\text{e,UEC}}]_{k=1}^{N}$, according to the trellis of Figure 6a. Note that the trellis decoder's encoded *a priori* LLR vectors $\tilde{\mathbf{b}}_2^{\text{a,UEC}}$ and $\tilde{\mathbf{b}}_3^{\text{a,UEC}}$, as well as the encoded extrinsic LLR vectors $\tilde{\mathbf{b}}_2^{\text{e,UEC}}$ and $\tilde{\mathbf{b}}_3^{\text{e,UEC}}$ jointly comprise two LLRs corresponding to each LLRs in the *a posteriori* LLR-vector $\tilde{\mathbf{b}}_1^{\text{UEC}}$.

In a conventional decoder, the URC decoder and UEC trellis decoder are activated alternately, in a similar manner

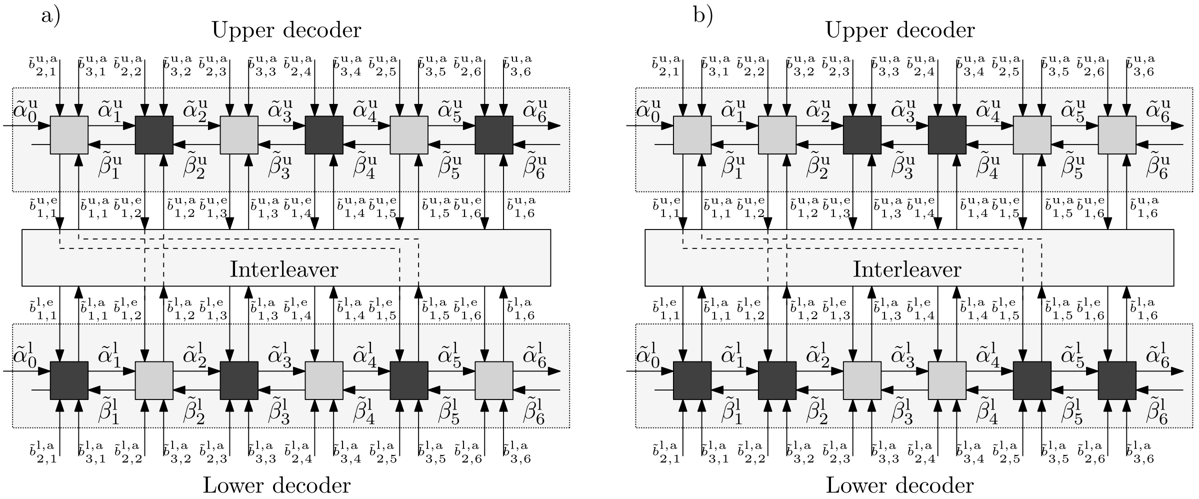


FIGURE 7. Fully parallel iterative decoders for the LTE turbo code. a) State of the art fully parallel iterative decoder (adapted from [15]). b) Novel scheduling proposed in this paper.

to the action of the turbo decoder of Section II-A2. After the activation of one of the two decoders, they exchange their LLR-vectors through the interleaver π_1 and deinterleaver π_1^{-1}. More specifically, the extrinsic LLR-vector $\tilde{\mathbf{b}}_1^{\mathrm{e,URC}}$ gleaned from the URC decoder is passed through π^{-1}, yielding the *a priori* encoded UEC LLR-vectors $\tilde{\mathbf{b}}_2^{\mathrm{a,UEC}}$ and $\tilde{\mathbf{b}}_3^{\mathrm{a,UEC}}$. Furthermore, the extrinsic encoded LLR-vectors $\tilde{\mathbf{b}}_2^{\mathrm{e,UEC}}$ and $\tilde{\mathbf{b}}_3^{\mathrm{e,UEC}}$ generated by the UEC decoder are passed through π, yielding the *a priori* URC input LLR vector $\tilde{\mathbf{b}}_1^{\mathrm{a,URC}}$.

The performance of the UEC decoder can be improved by exploiting the fact that the conditional probabilities associated with different transitions $P(m|m')$ are not equal. The logarithm of these conditional transition probabilities $\ln[P(m|m')]$ may be added to the *a priori* transition probabilities $\tilde{\gamma}$ during the Log-BCJR [7]. The decoder may calculate $P(m|m')$ using only knowledge of the occurrence probability $P(x)$ of the first $r_{\mathrm{UEC}}/2-1$ symbol values x, as well as knowledge of the average unary codeword length l [7]. Note that if this information is unknown at the receiver, the trellis decoder can operate without the transition probabilities $P(m|m')$, at the cost of a reduced error correction performance [9]. In this case, the required information can be estimated heuristically following the decoding of several symbol vectors.

Once a sufficient number of decoding iterations have been performed, the trellis decoder can output the vector of N *a posteriori* uncoded LLRs $\tilde{\mathbf{b}}_1^{\mathrm{UEC}} = [\tilde{b}_{1,k}^{\mathrm{UEC}}]_{k=1}^{N}$, which is passed to the unary decoder for recovering the symbol vector $\hat{\mathbf{x}}$. In analogy with the partitioning of the unary-encoded bits at the encoder into fixed length vectors $\mathbf{b}_1^{\mathrm{UEC}}$, there is also a buffer at the receiver for temporarily storing these LLRs, which pertain to symbols that are split between successive symbol-vectors, as described in Section II-B1. In order to assist the unary decoder, the transmitter may send a small amount of optional side information to the receiver. This conveys the number a of logical one-valued bits that are present in the bit-vector $\mathbf{b}_1^{\mathrm{UEC}}$. The unary decoder operates by converting the a highest LLR values in the vector $\mathbf{b}_1^{\mathrm{UEC}}$ to logical one-valued bits, since high LLR values indicate that the corresponding bit is likely to have the value 1, while the rest are converted to zero-valued bits. The unary decoder then converts the resultant-hard decision bit-vector $\tilde{\mathbf{y}}$ into symbols $\tilde{\mathbf{x}}$, according to Table 1. If no side information is invoked, then a hard decision is made of each bit, depending on whether the corresponding LLR is positive or negative.

C. FULLY PARALLEL DECODING ALGORITHM

In this section we will describe the operation of the fully parallel iterative decoder, which was proposed in [15]. In Sections II-A and II-B, we described how the turbo decoder and UEC-URC decoder may employ the Log-BCJR algorithm for alternately processing each component decoder, which iteratively exchange extrinsic information. By contrast, this section will show that the Log-BCJR algorithm can be replaced by the fully parallel iterative decoder, which carries out the tasks of both component decoders simultaneously.

Figure 7 depicts a fully parallel decoder for the LTE turbo code of Section II-A2. The fully parallel decoder is comprised of two component decoders, namely the upper decoder and the lower decoder, which are connected through an interleaver. In the fully parallel decoder, each component decoder is comprised of N decoding blocks, each of which correspond to a different trellis stage. The upper and lower decoder of Figure 7 are provided with the *a priori* LLR-vectors $\tilde{\mathbf{b}}_2^{\mathrm{u,a}}$, $\tilde{\mathbf{b}}_2^{\mathrm{l,a}}$ and $\tilde{\mathbf{b}}_3^{\mathrm{u,a}}$ from the channel. During the iterative decoding process, the upper and lower decoders exchange the extrinsic LLR-vectors $\tilde{\mathbf{b}}_1^{\mathrm{u,e}}$ and $\tilde{\mathbf{b}}_1^{\mathrm{l,e}}$ through the interleaver and deinterleaver, in the same manner as described in Section II-A2. Likewise, Figure 8 shows the fully parallel decoder applied to the UEC-URC decoder of Section II-B2. Here, in contrast to

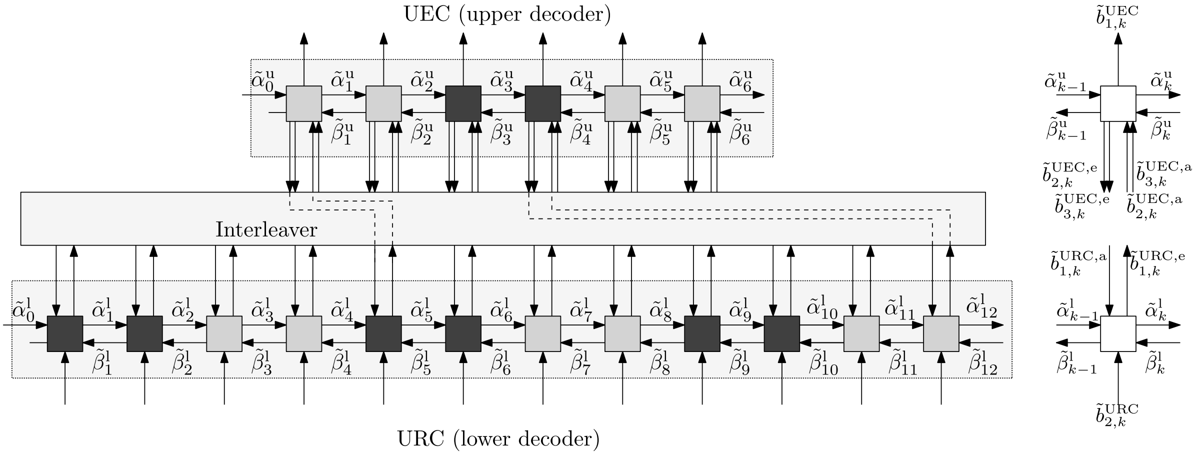

UEC (upper decoder)
$\tilde{\alpha}^{\mathrm{u}}_0$ $\tilde{\alpha}^{\mathrm{u}}_1$ $\tilde{\alpha}^{\mathrm{u}}_2$ $\tilde{\alpha}^{\mathrm{u}}_3$ $\tilde{\alpha}^{\mathrm{u}}_4$ $\tilde{\alpha}^{\mathrm{u}}_5$ $\tilde{\alpha}^{\mathrm{u}}_6$
$\tilde{\beta}^{\mathrm{u}}_1$ $\tilde{\beta}^{\mathrm{u}}_2$ $\tilde{\beta}^{\mathrm{u}}_3$ $\tilde{\beta}^{\mathrm{u}}_4$ $\tilde{\beta}^{\mathrm{u}}_5$ $\tilde{\beta}^{\mathrm{u}}_6$
Interleaver
$\tilde{\alpha}^{\mathrm{l}}_0$ $\tilde{\alpha}^{\mathrm{l}}_1$ $\tilde{\alpha}^{\mathrm{l}}_2$ $\tilde{\alpha}^{\mathrm{l}}_3$ $\tilde{\alpha}^{\mathrm{l}}_4$ $\tilde{\alpha}^{\mathrm{l}}_5$ $\tilde{\alpha}^{\mathrm{l}}_6$ $\tilde{\alpha}^{\mathrm{l}}_7$ $\tilde{\alpha}^{\mathrm{l}}_8$ $\tilde{\alpha}^{\mathrm{l}}_9$ $\tilde{\alpha}^{\mathrm{l}}_{10}$ $\tilde{\alpha}^{\mathrm{l}}_{11}$ $\tilde{\alpha}^{\mathrm{l}}_{12}$
$\tilde{\beta}^{\mathrm{l}}_1$ $\tilde{\beta}^{\mathrm{l}}_2$ $\tilde{\beta}^{\mathrm{l}}_3$ $\tilde{\beta}^{\mathrm{l}}_4$ $\tilde{\beta}^{\mathrm{l}}_5$ $\tilde{\beta}^{\mathrm{l}}_6$ $\tilde{\beta}^{\mathrm{l}}_7$ $\tilde{\beta}^{\mathrm{l}}_8$ $\tilde{\beta}^{\mathrm{l}}_9$ $\tilde{\beta}^{\mathrm{l}}_{10}$ $\tilde{\beta}^{\mathrm{l}}_{11}$ $\tilde{\beta}^{\mathrm{l}}_{12}$
URC (lower decoder)
$\tilde{b}^{\mathrm{UEC}}_{1,k}$
$\tilde{\alpha}^{\mathrm{u}}_{k-1}$ $\tilde{\alpha}^{\mathrm{u}}_k$
$\tilde{\beta}^{\mathrm{u}}_{k-1}$ $\tilde{\beta}^{\mathrm{u}}_k$
$\tilde{b}^{\mathrm{UEC,e}}_{2,k}$ $\tilde{b}^{\mathrm{UEC,a}}_{3,k}$
$\tilde{b}^{\mathrm{UEC,e}}_{3,k}$ $\tilde{b}^{\mathrm{UEC,a}}_{2,k}$
$\tilde{b}^{\mathrm{URC,a}}_{1,k}$ $\tilde{b}^{\mathrm{URC,e}}_{1,k}$
$\tilde{\alpha}^{\mathrm{l}}_{k-1}$ $\tilde{\alpha}^{\mathrm{l}}_k$
$\tilde{\beta}^{\mathrm{l}}_{k-1}$ $\tilde{\beta}^{\mathrm{l}}_k$
$\tilde{b}^{\mathrm{URC}}_{2,k}$

FIGURE 8. Novel scheduling proposed in this paper for the UEC-URC scheme.

the fully parallel turbo decoder of Figure 7, there are twice as many decoder blocks for the URC decoder as for the UEC trellis decoder, since there are twice as many URC trellis stages as there are UEC trellis stages. This is because in the UEC encoder, each input bit $b^{\mathrm{UEC}}_{1,k}$, generates two output bits $b^{\mathrm{UEC}}_{2,k}$ and $b^{\mathrm{UEC}}_{3,k}$, which are encoded by the URC encoder, as shown by Figure 5. A further difference with respect to the turbo decoder is that for the fully parallel UEC-URC decoder, each UEC trellis decoder block is required to output the pair of extrinsic LLRs $\tilde{b}^{\mathrm{UEC,e}}_{2,k}$ and $\tilde{b}^{\mathrm{UEC,e}}_{3,k}$, instead of just one.

Since the fully parallel iterative decoder is comprised of a separate decoding block for each trellis stage, these trellis stages can be processed in parallel, facilitating high throughputs and low latency at the receiver. This is in contrast to the conventional Log-BCJR, where the trellis stages are processed in order, according to forward and backward recursions. The operation of the decoding blocks in Figures 7 and 8 is described in (4)-(8), as shown at the bottom of this page, where the time indicies t and $(t-1)$ are used to show in which time period the various values are used. Note that these equations have been re-arranged from those in [15], so that they match with the hardware schematic that will be used in Section IV. In (4)-(8), the max* operator is defined for two operands as $\max^*(a, b) = \max(a, b) + \ln(1 + e^{-|a-b|})$, but it may be readily extended to more operands by exploiting the associative property.

Like the Log-BCJR algorithm [4], the equations of the fully parallel decoder comprise four sets of metrics. The $\tilde{\gamma}^t_k$ values of (4) represent the *a priori* probabilities of the transitions. These are calculated based on the *a priori* LLRs provided for each decoder block, either by the demodulator or by the interleaver in the previous time period, as well as based on the specific trellis structure $b_k(m_{k-1}, m_k)$ employed by the scheme. The $\tilde{\alpha}^t_k$ and $\tilde{\beta}^t_k$ values of (5) and (6) are forwards and backwards state metrics, respectively. These are provided based on the $\tilde{\alpha}^{t-1}_k$ and $\tilde{\beta}^{t-1}_{k-1}$ values calculated by a neighboring decoding block in the previous time period, as well as the $\tilde{\gamma}^t$ values from the current time period. Each decoder block outputs its $\tilde{\alpha}^t_k$ values to the next decoder block and the $\tilde{\beta}^t_k$ values to the previous decoder block, which are used in the subsequent time period. The $\tilde{\delta}^t_k$ values of (7) represent the *a posteriori* transition probabilities. These are calculated based on the *a priori* $\tilde{\alpha}^{t-1}_k$ and $\tilde{\beta}^{t-1}_{k-1}$ values provided by the neighboring decoding blocks in the previous

$$\tilde{\gamma}^t_k(m_{k-1}, m_k) = \sum_{j=1}^{L} \left[b_j(m_{k-1}, m_k) \cdot \tilde{b}^{\mathrm{a},t-1}_{j,k} \right] + \log[P(m_k|m_{k-1})] \tag{4}$$

$$\tilde{\alpha}^t_k(m_k) = \max^*_{\{s_{k-1}|c(m_{k-1},m_k)=1\}} [\tilde{\gamma}^t_k(m_{k-1}, m_k) + \tilde{\alpha}^{t-1}_{k-1}(m_{k-1})] \tag{5}$$

$$\tilde{\beta}^t_{k-1}(m_{k-1}) = \max^*_{\{s_k|c(m_{k-1},m_k)=1\}} [\tilde{\gamma}^t_k(m_{k-1}, m_k) + \tilde{\beta}^{t-1}_k(m_k)] \tag{6}$$

$$\tilde{\delta}^t_k(m_{k-1}, m_k) = \tilde{\gamma}^t_k(m_{k-1}, m_k) + \tilde{\alpha}^{t-1}_{k-1}(m_{k-1}) + \tilde{\beta}^{t-1}_k(m_k) \tag{7}$$

$$\begin{aligned} \tilde{b}^{\mathrm{e},t}_{j,k} = & \left[\max^*_{\{(s_{k-1},m_k)|b_j(m_{k-1},m_k)=1\}} [\tilde{\delta}^t_k(m_{k-1}, m_k)] \right] - \\ & \times \left[\max^*_{\{(s_{k-1},m_k)|b_j(m_{k-1},m_k)=0\}} [\tilde{\delta}^t_k(m_{k-1}, m_k)] \right] - \tilde{b}^{\mathrm{a},t-1}_{j,k} \end{aligned} \tag{8}$$

time period, as well as the $\tilde{\gamma}_k^t$ values from the current time period. These $\tilde{\delta}_k^t$ values are used to generate the extrinsic outputs $\tilde{b}_{j,k}^{e,t}$ of (8), which may be passed through the interleaver in order to assist the other constituent decoder in the iterative decoding process.

While all of the decoding blocks of both component decoders in Figures 7 and 8 can be operated at the same time, there are also other attractive activation orders. The first example of this is shown in Figure 7a, where two groups of blocks are shown, one with dark shading and one with light shading. This is known as odd-even activation, where each group of shaded blocks are operated in alternate time periods. More specifically, the dark shaded blocks are activated in odd indexed time periods, followed by the lightly shaded blocks in the even indexed time periods. Note that the t and $(t-1)$ notation of Equations (4)-(8) naturally support this activation order, since all inputs provided for a particular block in the previous time period are generated by the blocks having the opposite shading. Note that odd-even activation requires an odd-even interleaver, which connects dark shaded blocks to light shaded ones and vice versa. The LTE interleaver meets this requirement, as we will show in Section III-B. This odd-even scheduling reduces the complexity of the turbo decoder shown in Figure 7a by 50%, without compromising either its throughput or its error correction capability, as described in [15]. Note that the novel schedules shown in Figures 7b and 8 will be introduced in Section III.

III. ALGORITHM ADAPTATIONS

This section details the operation of the proposed enhancements to the fully parallel decoding algorithm of Section II-C. The modifications proposed in this section are motivated by the constraints imposed by the associated hardware implementation. More specifically, to increase the clock frequency of the hardware and hence improve the throughput and latency, more pipelining [32] is required, although this delays the exchange of information through the decoder. A naive approach would be to increase the degree of pipelining without considering the negative impact on the algorithm's error correction performance. By contrast, this section describes a novel scheduling, which considers the effect of pipelining in hardware implementation upon its error correction performance, which will be shown in Section IV to significantly improve the hardware efficiency attained. In this way, these improvements have been achieved by jointly considering the design of the iterative decoding algorithm and hardware. Indeed, we show that these enhancements improve its error correction performance, whilst increasing the achievable throughput of the entire system.

We commence in Section III-A by justifying the proposed modifications of Equations (4)-(8), as shown at the bottom of this page, in order to improve the scheduling of the fully parallel algorithm and aid its hardware implementation. Following this, Section III-B describes how the properties of the interleaver affect the operation of this modified fully parallel decoding algorithm. Section III-C character-

if $t \mod 4 = 0$, $k \in 1, 5, 9, \ldots$ (upper), $k \in 3, 7, 11, \ldots$ (lower) or
if $t \mod 4 = 2$, $k \in 3, 7, 11, \ldots$ (upper), $k \in 1, 5, 9, \ldots$ (lower)

$$\tilde{\gamma}_k^t(m_{k-1}, m_k) = \sum_{j=1}^{L}\left[b_j(m_{k-1}, m_k)\cdot \tilde{b}_{j,k}^{a,t-1}\right] + \log[P(m_k|m_{k-1})] \tag{9}$$

$$\tilde{\alpha}_k^t(m_k) = \max{}^* \{s_{k-1}|c(m_{k-1}, m_k) = 1\}[\tilde{\gamma}_k^t(m_{k-1}, m_k) + \tilde{\alpha}_{k-1}^{t-1}(m_{k-1})] \tag{10}$$

$$\tilde{\beta}_k^t(m_k) = \max_{\{s_{k+1}|c(m_k, m_{k+1})=1\}}{}^* [\tilde{\gamma}_{k+1}^t(m_k, m_{k+1}) + \tilde{\beta}_{k+1}^{t-1}(m_{k+1})] \tag{11}$$

$$\tilde{\delta}_k^t(m_{k-1}, m_k) = \tilde{\gamma}_k^t(m_{k-1}, m_k) + \tilde{\alpha}_{k-1}^{t-1}(m_{k-1}) + \tilde{\beta}_k^{t-4}(m_k) \tag{12}$$

$$\tilde{b}_{j,k}^{e,t} = \left[\max_{\{(s_{k-1},m_k)|b_j(m_{k-1},m_k)=1\}}{}^* [\tilde{\delta}_k^t(m_{k-1}, m_k)]\right] - \left[\max_{\{(s_{k-1},m_k)|b_j(m_{k-1},m_k)=0\}}{}^* [\tilde{\delta}_k^t(m_{k-1}, m_k)]\right] - \tilde{b}_{j,k}^{a,t-1} \tag{13}$$

if $t \mod 4 = 1$, $k \in 2, 6, 10, \ldots$ (upper), $k \in 4, 8, 12, \ldots$ (lower) or
if $t \mod 4 = 3$, $k \in 4, 8, 12, \ldots$ (upper), $k \in 2, 6, 10, \ldots$ (lower)

$$\tilde{\gamma}_k^t(m_{k-1}, m_k) = \sum_{j=1}^{L}\left[b_j(m_{k-1}, m_k)\cdot \tilde{b}_{j,k}^{a,t-1}\right] + \log[P(m_k|m_{k-1})] \tag{14}$$

$$\tilde{\alpha}_k^t(m_k) = \max_{\{s_{k-1}|c(m_{k-1}, m_k)=1\}}{}^* [\tilde{\gamma}_k^t(m_{k-1}, m_k) + \tilde{\alpha}_{k-1}^{t-1}(m_{k-1})] \tag{15}$$

$$\tilde{\beta}_{k-2}^t(m_{k-2}) = \max_{\{s_{k-1}|c(m_{k-2}, m_{k-1})=1\}}{}^* [\tilde{\gamma}_{k-1}^t(m_{k-2}, m_{k-1}) + \tilde{\beta}_{k-1}^{t-1}(m_{k-1})] \tag{16}$$

$$\tilde{\delta}_k^t(m_{k-1}, m_k) = \tilde{\gamma}_k^t(m_{k-1}, m_k) + \tilde{\alpha}_{k-1}^{t-1}(m_{k-1}) + \tilde{\beta}_k^{t-2}(m_k) \tag{17}$$

$$\tilde{b}_{j,k}^{e,t} = \left[\max_{\{(s_{k-1},m_k)|b_j(m_{k-1},m_k)=1\}}{}^* [\tilde{\delta}_k^t(m_{k-1}, m_k)]\right] - \left[\max_{\{(s_{k-1},m_k)|b_j(m_{k-1},m_k)=0\}}{}^* [\tilde{\delta}_k^t(m_{k-1}, m_k)]\right] - \tilde{b}_{j,k}^{a,t-1} \tag{18}$$

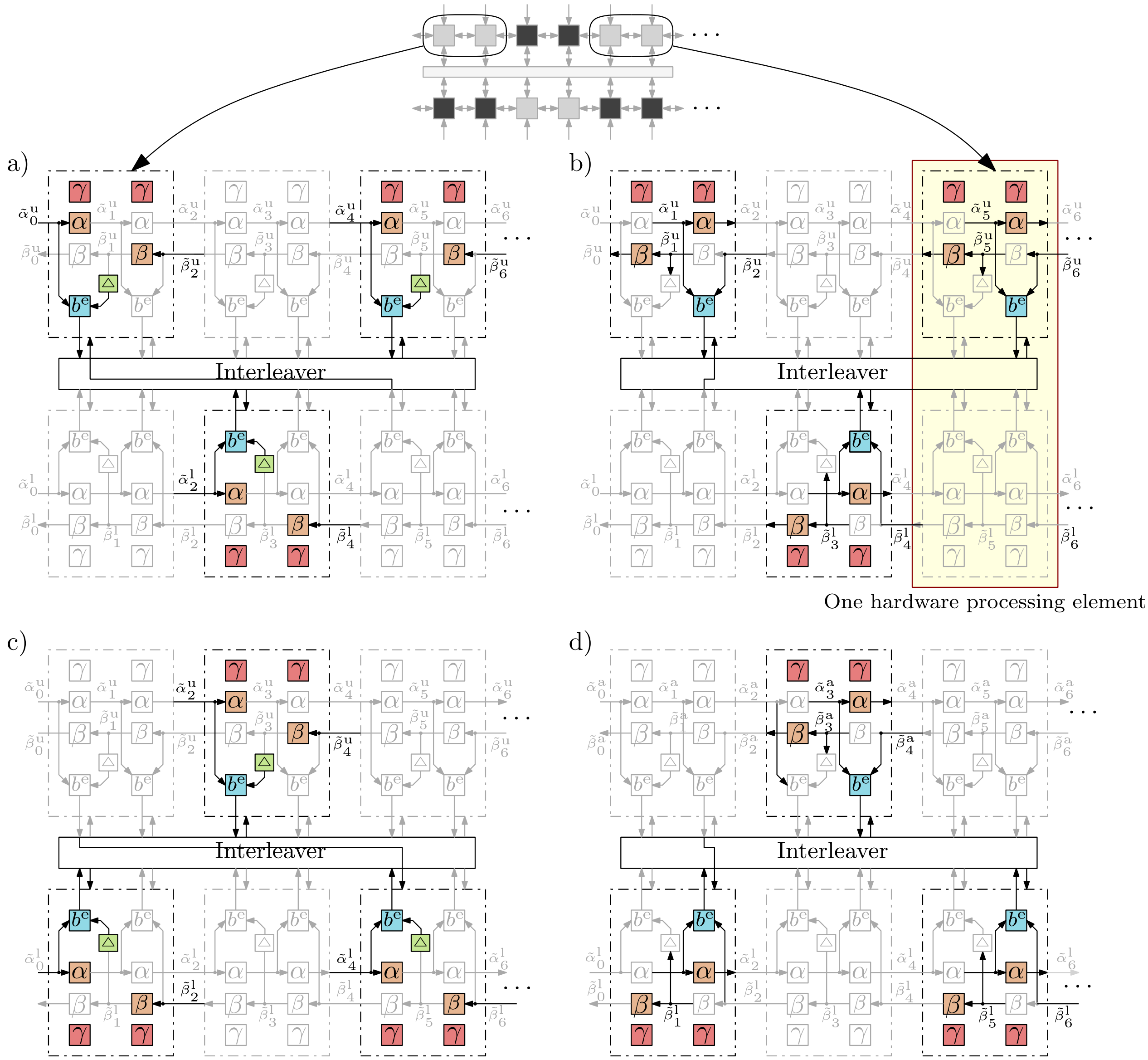

a)
b)
c)
d)
γ
α
β
b^e
Interleaver
$\tilde{\alpha}^u_0$ $\tilde{\alpha}^u_1$ $\tilde{\alpha}^u_2$ $\tilde{\alpha}^u_3$ $\tilde{\alpha}^u_4$ $\tilde{\alpha}^u_5$ $\tilde{\alpha}^u_6$
$\tilde{\beta}^u_0$ $\tilde{\beta}^u_1$ $\tilde{\beta}^u_2$ $\tilde{\beta}^u_3$ $\tilde{\beta}^u_4$ $\tilde{\beta}^u_5$ $\tilde{\beta}^u_6$
$\tilde{\alpha}^l_0$ $\tilde{\alpha}^l_2$ $\tilde{\alpha}^l_4$ $\tilde{\alpha}^l_6$
$\tilde{\beta}^l_0$ $\tilde{\beta}^l_1$ $\tilde{\beta}^l_2$ $\tilde{\beta}^l_3$ $\tilde{\beta}^l_4$ $\tilde{\beta}^l_5$ $\tilde{\beta}^l_6$
$\tilde{\alpha}^a_0$ $\tilde{\alpha}^a_1$ $\tilde{\alpha}^a_2$ $\tilde{\alpha}^a_3$ $\tilde{\alpha}^a_4$ $\tilde{\alpha}^a_5$ $\tilde{\alpha}^a_6$
$\tilde{\beta}^a_0$ $\tilde{\beta}^a_1$ $\tilde{\beta}^a_2$ $\tilde{\beta}^a_3$ $\tilde{\beta}^a_4$ $\tilde{\beta}^a_5$ $\tilde{\beta}^a_6$
One hardware processing element
. . .

FIGURE 9. Proposed enhanced fully parallel decoding algorithm. The shaded portion shows the components that are implemented by each hardware decoder of Figure 16. Subfigures a, b, c and d show the four successive steps of each decoding iteration.

izes the error correction performance of the proposed fully parallel algorithm, by comparing the number of decoding iterations required to those of the conventional Log-BCJR algorithm and the fully parallel decoding algorithm of [7]. Section III-D discusses the impact of adopting the reduced complexity approach of the Max-Log-BCJR algorithm [21] upon the error correction performance, showing that this can be mitigated by using extrinsic scaling [33]. Finally, Section III-E discusses the bit-widths required for the fixed point two's-complement numbers used inside the decoder.

A. SCHEDULING

In this section we will detail our novel approach to scheduling the operations of the fully parallel decoding algorithm, where the Equations (4)-(8) are reordered and transformed into (9)-(18), in order to facilitate its improved hardware implementation and to improve its error correction performance. Figure 9 graphically represents Equations (9)-(18), detailing which of the different operations of each algorithmic block are performed in which of the four successive time periods that constitute each complete decoding iteration. More specifically, Figure 9 shows the four time periods of the proposed scheduling, when applied to the LTE turbo code, while Figure 10 illustrates one of the four time periods in the schedule for the UEC-URC scheme. Note that since the UEC-URC scheme comprises twice as many URC trellis stages as that of the UEC trellis stages, the former are arranged into two rows of equal length in Figure 10.

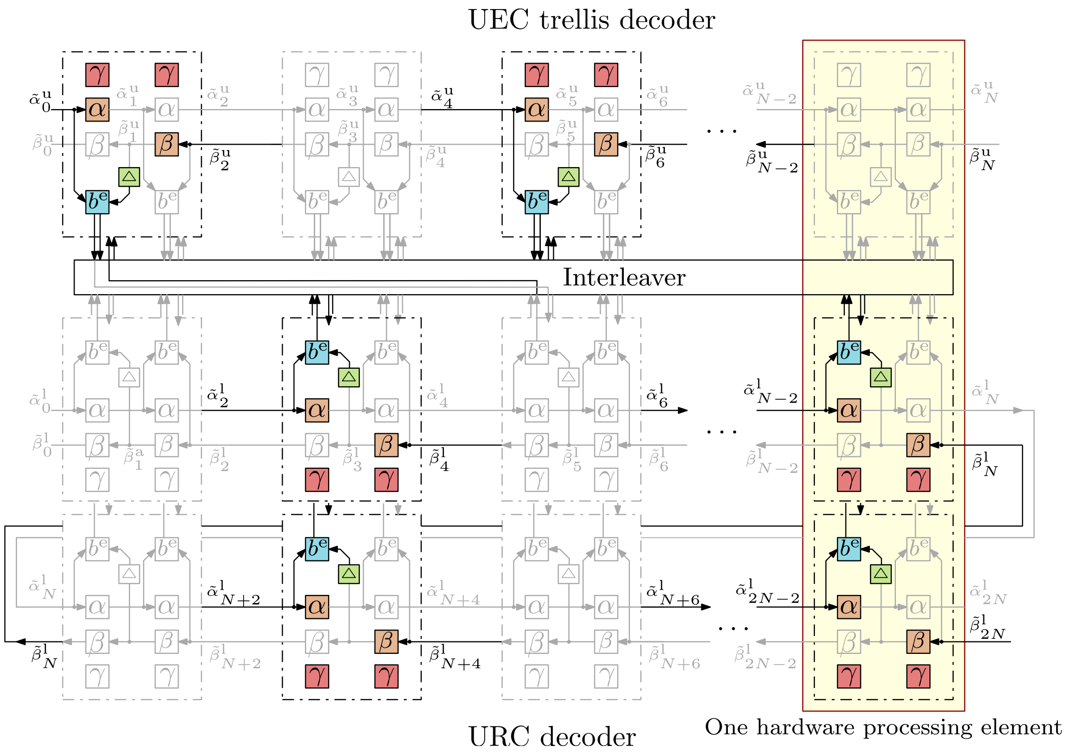


FIGURE 10. Proposed enhanced fully parallel decoding algorithm for the UEC-URC scheme. Here, only the first of four steps is shown, in analogy to that shown in Figure 9a.

Equations (4)-(8) of the fully parallel decoding algorithm of [15] have been transformed into (9)-(18) of the proposed modification by altering the order of operation. When applying an odd-even interleaver, each decoding iteration of the fully parallel decoding algorithm of [15] requires two time periods t, in which each algorithmic block is operated once, as described in Section II-C. By contrast, Equations (9)-(18) result in each decoding iteration of the proposed modification requiring four time periods t, during which each algorithmic block is also operated once. Note that the changes relative to Equations (4)-(8) have been highlighted in (9)-(18). Some of the equations appear unchanged, but are shown in (9)-(18) because the time slot in which they are activated has changed.

In Figures 9 and 10, the dashed lines group the processing, which is required for each *pair* of algorithmic blocks. Specifically, the pairs of algorithmic blocks, which are being processed in the current time period t are surrounded by black dashed lines, while the pairs of algorithmic blocks that are not being processed are surrounded by grayed dashed lines. Within the pairs of algorithmic blocks, individual operations are also shown, which are active when shaded, while the inactive ones are printed in gray. More specifically, the orange blocks marked α and β implement Equations (10), (15) and (11), (16), respectively. The blue blocks marked b_e represent (12), (17) and (13), (18), while the red blocks marked γ correspond to Equations (9) and (14). The green Δ block represents a memory element, which is used to store values that are not used immediately. These colors are consistently used throughout the remainder of this paper to show the mapping between the algorithm and hardware implementation.

The proposed scheduling of this section preserves the odd-even activation order that was initially proposed in [15]. However, in the proposed algorithm, the odd-even scheduling applies to pairs of algorithmic blocks rather than to individual blocks. As will be described in Section III-B, the LTE interleaver supports this scheduling, while the UEC-URC interleaver can be readily designed to support this scheduling.

The shaded region of Figures 9 and 10 shows the specific portion of the algorithm that is decoded by one hardware processing element, as will be discussed in Section IV. More specifically, Figure 9 illustrates the processing performed by three hardware processing elements for 12 algorithmic blocks of Figure 7, in four time steps.

Note that steps (a) and (b) of Figure 9 correspond to the light shaded pairs of blocks shown in Figures 7b and 8. These are processed in the first two of the four time periods. Meanwhile, steps (c) and (d) show how the dark shaded pairs of blocks of Figures 7b and 8 are processed in the remaining two time periods. The novel scheduling of this work allows information to pass through the two decoders at a higher rate than in the conventional fully parallel decoding algorithm. More specifically, after one iteration of the proposed algorithm, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ state metrics have propagated from $\tilde{\boldsymbol{\alpha}}_k$ to $\tilde{\boldsymbol{\alpha}}_{k+4}$ and $\tilde{\boldsymbol{\beta}}_{k+4}$ to $\tilde{\boldsymbol{\beta}}_k$, respectively. By contrast, for the odd-even fully parallel decoding algorithm of [15] requires only two iterations for state metrics to propagate this distance.

The $\tilde{b}_k^{\mathrm{e},t}$ calculation in steps (a) and (c) uses the $\tilde{\boldsymbol{\alpha}}_{k-1}^{t-1}$ values, which have been calculated in the previous time period, but require the $\tilde{\boldsymbol{\beta}}_k^{t-4}$ values that were calculated in the previous iteration. This is because the $\tilde{b}_k^{\mathrm{e},t}$ and $\tilde{\boldsymbol{\beta}}_k^t$ calculations are performed in the same time period, and so the result of the $\tilde{\boldsymbol{\beta}}_k^t$ calculation is not ready in time for use in the $\tilde{b}_k^{\mathrm{e},t}$ calculation. Owing to this, the memory elements Δ are loaded with the output of the $\tilde{\boldsymbol{\beta}}$ calculation at the start of step (b). These $\tilde{\boldsymbol{\beta}}$ values are then stored until the start of step (a) in the next iteration, where they are used as inputs to the $\tilde{b}_k^{\mathrm{e},t}$ calculation. Note that the $\tilde{b}_k^{\mathrm{e},t}$ calculation of steps (b) and (d) do not suffer from this issue, since the $\tilde{b}_k^{\mathrm{e},t}$ calculation requires $\tilde{\boldsymbol{\beta}}_k^{t-2}$, which is also used in the previous time period.

It is worth noting that the paired operation proposed here is different to the radix-4 technique [18], [34], which is often used with conventional Log-BCJR decoders. Radix-4 decoders traverse two trellis stages in the same time period, by combining two trellis stages together and processing the combined trellis as one operation. By contrast, the paired operation proposed here requires two time periods for processing two trellis stages, but with the overlapping of their processing.

B. INTERLEAVER

As previously discussed in Section II-C, the fully parallel turbo decoder of Figure 7a benefits from odd-even operation, which enables a 50% reduction in computational complexity. More specifically, the blocks of different shading are operated alternately in successive time periods, so that outputs generated in one time period by blocks of one shading are consumed in the next time period by blocks of the other shading. In order to facilitate this, the interleaver of Figure 7a should be designed such that the lightly shaded blocks are only connected to the darkly shaded blocks. More specifically, the interleaver π_1 only connects blocks in the upper row having even indices to blocks in the lower row having even indices $\pi(i)$. Likewise, blocks having odd indices are only connected to blocks in the other row having odd indices. We may express this property as $\mod(\pi(i), 2) = \mod(i, 2)$, where $\mod(\cdot)$ is the modulo operator. This property is held by all of the 188 LTE interleavers, which have different frame lengths $N \in \{40, 48, \ldots, 6144\}$.

In the proposed algorithm of Section III-A, the interleaver is still required to connect all lightly shared blocks to darkly shaded blocks, in order to maintain correct odd-even operation. Since the blocks of Figures 7b and 8 are arranged as pairs of the same color, a different interleaver property is required relative to the algorithm of Figure 7a. More specifically, the first block in each pair of a specific shading must be interleaved to the first block of another pair of the other shading. Likewise, the second block in each pair of one shading must be interleaved to the second block of another pair of the other shading. This can be seen in Figure 7b, where the extrinsic LLR $\tilde{b}_{1,1}^{\mathrm{e}}$, which is output from the first block of a lightly shaded pair, is interleaved to $\tilde{b}_{1,5}^{\mathrm{a}}$ and input to the first block of a darkly shaded pair. This maximizes the number of time periods available for extrinsic LLRs to be generated, interleaved and be used as *a priori* LLRs in the connected algorithmic block. We may express the interleaver property required by the proposed scheduling as $\mod(\pi(i), 4) = \mod(i, 4)$, which we refer to as the *mod4 type A* property. This expression is also demonstrated by Figure 11, where the solid lines show the *mod4 type A* property, which will be exploited by the implementation of Section IV. Approximately half of the 188 LTE interleavers for different frame lengths $N \in \{40, 48, \ldots, 6144\}$ meet the *mod4 type A* property. The other half have a similar property, which we refer to as the *mod4 type B* property and which is shown by the dashed lines of Figure 11. When a *mod4 type B* interleaver is used, the odd-even scheduling property shown in Figure 7b is not entirely satisfied, since half of the extrinsic LLR connections through the interleaver will connect to blocks of the same shading. While a scheme using this interleaver can still be decoded using the scheduling of Figure 9 and the architecture of Section IV, there is a slight performance disadvantage, which will be explored in Section III-C. This performance degradation is imposed by the extrinsic LLRs that suffer from an additional delay before they are used as *a priori* LLRs.

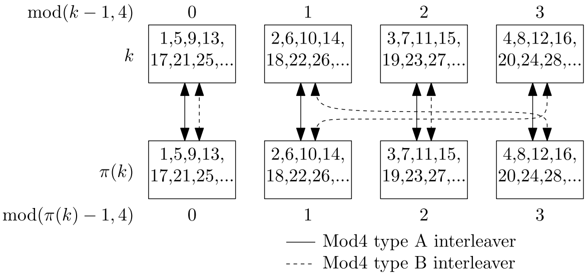


FIGURE 11. The mod4 property of the LTE interleaver, which is of either type A or type B for all 188 designs for the different frame lengths $N \in \{40, 48, \ldots, 6144\}$.

For the UEC-URC scheme of Figure 5, a mod4 interleaver is also required. Since we are free to design an interleaver for this proprietary scheme, a mod4 type A interleaver may be designed for all supported frame lengths. To maintain the mod4 properties of the interleaver, each extrinsic output $\tilde{b}_{2,k}^{\mathrm{UEC},e}$ and $\tilde{b}_{3,k}^{\mathrm{UEC,e}}$ from each UEC algorithmic block of a particular shading must be interleaved to a URC algorithmic block of the other shading. Likewise, the deinterleaver must route the extrinsic LLR $\tilde{b}_{1,k}^{\mathrm{URC,e}}$ produced by a URC algorithmic block of a particular shading to become either the *a priori* LLR $\tilde{b}_{2,k}^{\mathrm{UEC,a}}$ or $\tilde{b}_{3,k}^{\mathrm{UEC,a}}$ of a UEC algorithmic block having the opposite shading, using the inverse interleaving pattern.

C. ERROR CORRECTION PERFORMANCE

Figure 12 shows the BER performance of the LTE turbo code of Figure 3 using the proposed decoding schedule of Figure 9, when using QPSK modulation for communication over an uncorrelated narrowband Rayleigh fading chan-

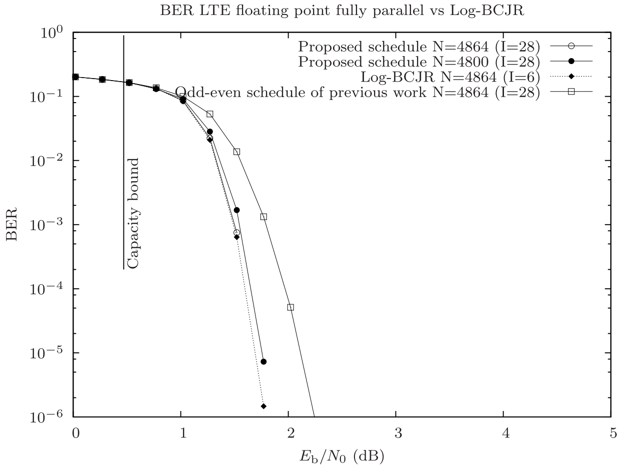


FIGURE 12. BER performance of the LTE turbo scheme of Figure 3, when employing various decoding algorithms and QPSK modulation for communication over an uncorrelated narrowband Rayleigh fading channel.

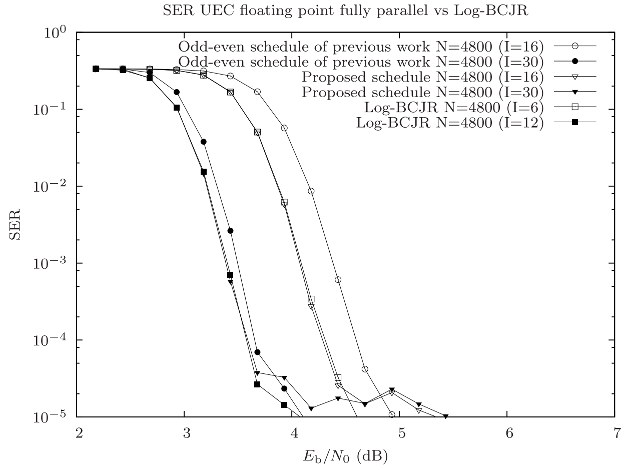


FIGURE 13. SER performance of the UEC-URC scheme of Figure 5, when employing various decoding algorithms and QPSK modulation for communication over an uncorrelated narrowband Rayleigh fading channel.

nel. Here, we compare the proposed scheduling with the Log-BCJR algorithm and the odd-even scheduling of [15], for both types of interleavers characterized in Figure 11. More specifically, the $N = 4864$-bit LTE interleaver is of *mod4 type A*, while the $N = 4800$-bit LTE interleaver is of *mod4 type B*. Figure 12 plots the BER performance for the Log-BCJR algorithm when employing 6 decoding iterations. Figure 12 shows that the same BER performance may be achieved using the proposed turbo decoding schedule of Figure 9 and a *mod 4 type A* interleaver, when performing 28 decoding iterations. However, when employing 28 iterations for a *mod4 type B* interleaver, the proposed schedule of Figure 9 can be seen to impose a modest 0.07 dB performance degradation at a BER of 10^{-5}. Likewise, when performing 28 decoding iterations, the odd-even scheduling of [15] suffers a more significant 0.3 dB performance degradation at a BER of 10^{-5}, compared to the proposed scheduling. Indeed, this schedule requires 42 decoding iterations to achieve the BER performance offered by the proposed schedule. Note that since each decoding iteration constitutes one activation of each algorithmic block, the comparison of the fully parallel schemes is fair in terms of decoding complexity.

Figure 13 plots the SER performance of the UEC-URC scheme of Figure 5 using the proposed decoding schedule of Figure 10, when employing QPSK modulation for communication over an uncorrelated narrowband Rayleigh fading channel. Here, the generated symbols $\mathbf{x}$ obey a zeta distribution, having $p_1 = 0.8$, while the unary-encoded bits are partitioned into frames $\mathbf{b}_1^{\mathrm{UEC}}$ comprising $N = 4800$ bits, which corresponds to an average of 3134 symbols per frame. These schemes use random interleavers that obey the *mod 4 type A* constraint of Section III-B. Figure 13 also compares the SER performance achieved, when employing the Log-BCJR algorithm and the fully parallel algorithm using the odd-even scheduling of [15]. The number of decoding iterations was chosen to match the performance achieved by the Log-BCJR after 6 and 12 iterations, which requires 16 and 30 iterations of the proposed schedule, respectively. Note that for the UEC-URC scheme, the proposed schedule requires only 16 decoding iterations to match the performance of the Log-BCJR after 6 iterations, which is significantly lower than the 28 required for the LTE turbo decoder. Similarly to the LTE scheme, our proposed scheduling improves the UEC-URC scheme by about 0.3 dB compared to the odd-even schedule after 16 iterations, as well as offering a marginal improvement after 30 iterations, since further iterations give marginal performance gains.

D. EXTRINSIC SCALING

As described in Section II-C, the algorithmic blocks of Figures 7 and 8 employ the max* operator, where $\max^*(a, b) = \max(a, b) + \ln(1 + e^{-|a-b|})$. However, in order to reduce the associated computational complexity, both the natural logarithm and the exponential operations are often omitted by using the approximation $\max^*(a, b) \approx \max(a, b)$. This approximation typically imposes an error correction performance penalty of about 1 dB depending on the scheme, although some of this loss can be mitigated by using extrinsic scaling [33], [35]. This method multiplies the iteratively exchanged extrinsic LLRs by a constant value, which represents the decoder's reduced confidence in the values of the bits, due to the employment of sub-optimal decoding. The optimal value for this scaling value is typically in the range 0.6 to 0.8, depending on the channel SNR [36]. Despite this, typically a fixed scaling value is used for the sake of simplifying the implementation. More specifically, typically a value of 0.75 is chosen, since it can be implemented in a simple manner, when using the two's-complement fixed point number representation. In this case, a multiplication by 0.75 can be approximated by adding the extrinsic

LLR right-shifted once, to itself but right-shifted twice. This yields the floor(·) of the multiplication by 0.75, owing to the limited precision of the fixed point numbers [17].

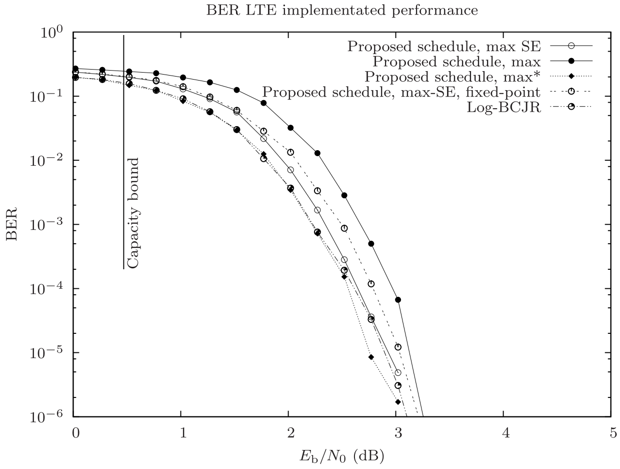


FIGURE 14. BER results for the LTE turbo scheme of Figure 3, using different extrinsic scaling and numerical representation techniques. Here, an uncorrelated narrowband Rayleigh fading channel and QPSK modulation is used. All schemes use a frame length of $N = 440$ bits, as well as 28 decoding iterations.

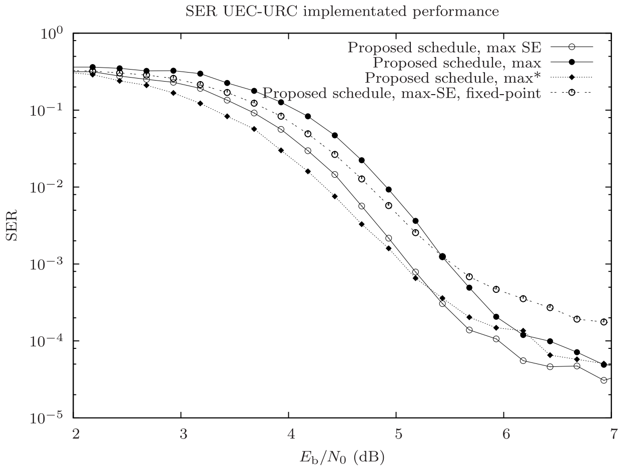


FIGURE 15. SER results for UEC-URC scheme of Figure 5, using different extrinsic scaling and numerical representation techniques. An uncorrelated narrowband Rayleigh fading channel and QPSK modulation is used. All schemes use partitioning to guarantee $N = 440$ bits in each frame, as well as 20 decoding iterations.

Figures 14 and 15 show the resultant BER and SER performance of the LTE turbo code scheme and UEC-URC schemes, respectively. Here, we use frame lengths of $N = 440$ bits, which is representative of the frame lengths presented in Section V. Figures 14 and 15 compare the error correction performance obtained using the ideal max* operator, the max operator with an extrinsic LLR scaling factor of 0.75 (max-SE) and the max operator without scaling. For the LTE scheme, the extrinsic scaling reduces the performance gap between the ideal max* and the max operations to 0.2 dB, while the non-scaled max operator suffers from a 0.5 dB loss. The UEC-URC scheme has similar performance gaps, where the extrinsic scaling reduces the loss to 0.4 dB, while a loss of 0.8 dB is imposed without extrinsic scaling.

E. NUMBER REPRESENTATION

In this section we discuss the specific number representations used in the proposed decoders. While floating point numbers have been assumed throughout the simulations discussed in the previous sections, fixed point twos-complement number representations are preferred in hardware implementations, since this dramatically reduces the complexity. In particular, this section discusses a method conceived for reducing the dynamic range of the $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\boldsymbol{\beta}}$ state metrics. We also quantify the fixed-point bit widths that are required, in order to approach the upper-bound performance of a floating point decoder.

When using two's-complement fixed-point numbers, the LLRs provided by the demodulator may be represented by fixed-point numbers having a bit width of w_d, the extrinsic LLRs may use a bit width of w_e, while the state metric $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\boldsymbol{\beta}}$ values may be conveyed between adjacent algorithmic blocks using w_m bits. However, as shown in Equations (5) and (6), the values of the state metrics $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\boldsymbol{\beta}}$ tend to grow without bound in successive iterations of the proposed decoding algorithm, owing to the accumulation of values that are typically positive, due to the action of the max operator. If however the values of the state metrics $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\boldsymbol{\beta}}$ become excessively large, they may cause twos-complement overflow, where a small positive integer added to large positive integer erroneously results in a large negative integer. These errors can severely degrade the operation of the decoder, resulting in a very poor error correction performance. Since the state metrics tend to grow without bound, this overflow problem is inevitable unless an excessively high bit width w_m is employed or unless a technique is used for reducing the dynamic range of the $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\boldsymbol{\beta}}$ values. In the proposed algorithm, we reduce the dynamic range of the state metrics and maintain a modest bit width w_m by using the clipping technique [17]. This technique relies on the observation that the absolute value of $\tilde{\boldsymbol{\alpha}}$ or $\tilde{\boldsymbol{\beta}}$ is not important, but rather it is the difference between the $\tilde{\boldsymbol{\alpha}}$ or $\tilde{\boldsymbol{\beta}}$ values produced for each trellis stage that conveys the relative probability of each state. Note that the number r of $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\boldsymbol{\beta}}$ values produced for each trellis stage is given by $r_{\text{LTE}} = 8$ for the LTE code, $r_{\text{UEC}} = 4$ for the UEC trellis code and $r_{\text{URC}} = 2$ for the URC code of the UEC-URC scheme. An implication of this observation is that adding or subtracting the same value to all state metrics in a set of r number of $\tilde{\boldsymbol{\alpha}}$ or $\tilde{\boldsymbol{\beta}}$ values makes no difference as to the decoding algorithm's operation.

In the clipping technique, each processing element avoids overflow by using more than w_m bits for its internal calculations, but the state metrics $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\boldsymbol{\beta}}$ are clipped to the bit widths of w_m. More specifically, the clipping method subtracts the $\tilde{\alpha}$ and $\tilde{\beta}$ value for the first trellis state from the rest of the values for each trellis stage, according to $\tilde{\alpha}_k(m_k) = \tilde{\alpha}'_k(m_k) - \tilde{\alpha}'_k(1)$ and $\tilde{\beta}_k(m_k) = \tilde{\beta}'_k(m_k) - \tilde{\beta}'_k(1)$. This guarantees an output where $\alpha_k(1) = 0$ and $\beta_k(1) = 0$, and where the remaining state metrics have lower values than they would otherwise. As a further step to ensure that the α and β values do not overflow, each α and β value is clipped so that it does not exceed the bit width w_m of the fixed-point number representation. Note that since clipping guarantees that we have $\tilde{\alpha}_k(1) = 0$ and $\tilde{\beta}_k(1) = 0$, it has the additional advantage that these values can be readily removed from subsequent calculations.

Figures 14 and 15 characterize the impact of using the fixed-point number representation on the BER and SER performance of the LTE scheme and UEC-URC scheme, respectively. Each figure compares the idealized floating point performance against the performance obtained when fixed-point numbers having particular bit-widths are employed.

More specifically, in the case of the LTE turbo decoder employing clipping, we recommend the use of a bit width of $w_m = 6$ for the state metrics, $w_e = 6$ bits for the extrinsic LLRs, and $w_d = 6$ bits for the LLRs provided by the demodulator. Meanwhile, in the case of the UEC-URC decoder employing clipping, we recommend the use of $w_m = 7$ bits for the state metrics, $w_e = 6$ bits for the extrinsic output, and $w_d = 6$ bits for the demodulator input. Note that wider bit widths are required for the state metrics of the UEC-URC scheme owing to the extended dynamic range that is caused by the non-equiprobable transitions and states in the UEC trellis, as described in Section II-B1. As shown in Figures 14 and 15, the bit widths recommended above offer a similar error correction performance to the floating point algorithm using the max approximation and extrinsic scaling. Note that if shorter bit widths are chosen, the decoder can exhibit a high error floor or a turbo cliff at a higher SNR.

IV. FPGA IMPLEMENTATION

In this section, we detail the FPGA implementation of the algorithm described in Section III. By designing the algorithm and architecture jointly, we achieve an efficient exploitation of the decoder hardware, as well as a powerful error correction performance. Furthermore, the proposed FPGA implementation is designed for facilitating the decoding of longer frame lengths than that which can be achieved with the aid of the previous design of [17] within the limited amount of hardware resources on an FPGA. This is achieved by sharing hardware processing elements between pairs of trellis stages, which also supports efficient pipelining and an increased clock frequency.

We commence in Section IV-A by detailing the operation of each hardware processing element in the decoder. We discuss a generic hardware processing element, which could be applied to either the LTE or UEC-URC schemes of Figures 3 and 5 respectively. Following this, Section IV-B describes the timing of the hardware processing elements, as well as how the algorithmic schedule of Section III-A is implemented on the hardware. In Section IV-C we discuss the specific modifications required by the UEC-URC scheme of Figure 5 compared to the LTE scheme of Figure 3, detailing each of the computation units within each hardware processing element. Finally, Section IV-D compares our jointly designed algorithm and hardware FPGA implementation with the previous implementations of the fully parallel decoder.

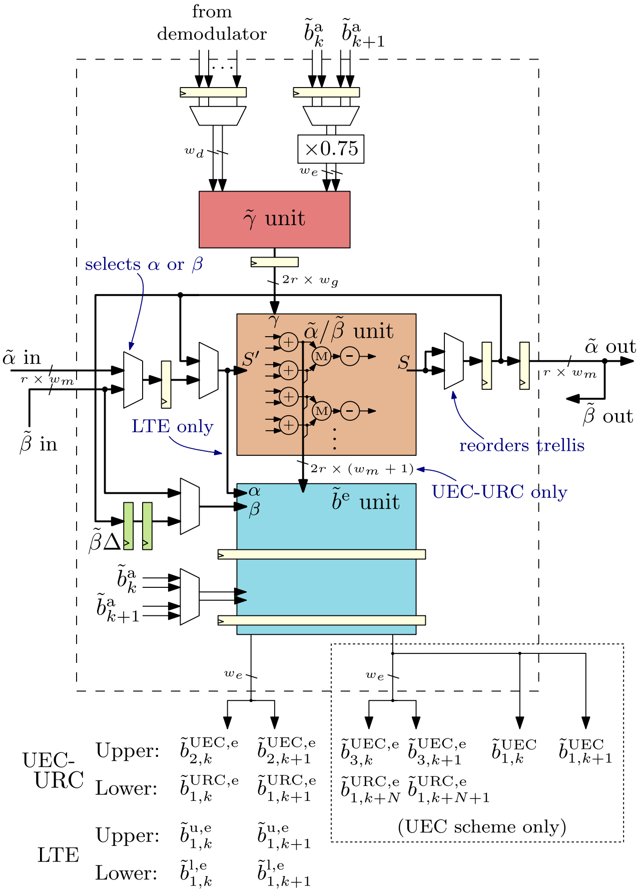

from demodulator
$\tilde{b}^{\mathrm{a}}_k$
$\tilde{b}^{\mathrm{a}}_{k+1}$
w_d
×0.75
w_e
$\tilde{\gamma}$ unit
selects α or β
$2r \times w_g$
γ
$\tilde{\alpha}/\tilde{\beta}$ unit
S'
S
M
M
$\tilde{\alpha}$ in
$r \times w_m$
$\tilde{\beta}$ in
LTE only
$\tilde{\alpha}$ out
$r \times w_m$
$\tilde{\beta}$ out
reorders trellis
$2r \times (w_m + 1)$
UEC-URC only
α
β
$\tilde{b}^{\mathrm{e}}$ unit
$\beta\Delta$
$\tilde{b}^{\mathrm{a}}_k$
$\tilde{b}^{\mathrm{a}}_{k+1}$
w_e
w_e
UEC-URC
Upper:
$\tilde{b}^{\mathrm{UEC,e}}_{2,k}$
$\tilde{b}^{\mathrm{UEC,e}}_{2,k+1}$
$\tilde{b}^{\mathrm{UEC,e}}_{3,k}$
$\tilde{b}^{\mathrm{UEC,e}}_{3,k+1}$
$\tilde{b}^{\mathrm{UEC}}_{1,k}$
$\tilde{b}^{\mathrm{UEC}}_{1,k+1}$
Lower:
$\tilde{b}^{\mathrm{URC,e}}_{1,k}$
$\tilde{b}^{\mathrm{URC,e}}_{1,k+1}$
$\tilde{b}^{\mathrm{URC,e}}_{1,k+N}$
$\tilde{b}^{\mathrm{URC,e}}_{1,k+N+1}$
(UEC scheme only)
LTE
Upper:
$\tilde{b}^{\mathrm{u,e}}_{1,k}$
$\tilde{b}^{\mathrm{u,e}}_{1,k+1}$
Lower:
$\tilde{b}^{\mathrm{l,e}}_{1,k}$
$\tilde{b}^{\mathrm{l,e}}_{1,k+1}$

FIGURE 16. Schematic of a hardware processing element. Here, $k \in \{1, 3, 5, \ldots, N-1\}$ for each of the $N/2$ hardware processing elements. The color shading of the blocks matches with the colors of previous figures.

A. DECODING BLOCK TOP LEVEL

As described in Section III-A, the $2N$ algorithmic blocks of the proposed LTE decoder are implemented using $N/2$ hardware processing elements, as indicated by the shaded area of Figure 9b Likewise, the $3N$ algorithmic blocks of the proposed UEC-URC decoder are implemented using $N/2$ hardware processing elements, as indicated by the shaded area of Figure 10. The schematic of each of the $N/2$ hardware processing block is shown in Figure 16. Each hardware processing element of Figure 16 is used for implementing the algorithmic blocks in both the top and bottom rows of the scheme. More specifically, when the hardware processing elements implement the scheduling of Figure 9, half of the hardware processing elements process the top decoder, while half the processing elements process the bottom decoder during steps (a) and (b). For steps (c) and (d), each hardware processing element switches to carrying out the other decoder's actions.

In this work, each hardware processing block is comprised of an $\tilde{\alpha}/\tilde{\beta}$ unit, a $\tilde{b}^{\mathrm{e}}$ unit, a $\tilde{\gamma}$ unit, as well as other multiplexers and registers. More specifically, in the $\tilde{\alpha}/\tilde{\beta}$ unit, we use a single piece of hardware to undertake the $\tilde{\alpha}$ and $\tilde{\beta}$ calculations of (10), (11), (15), (16) and Figure 9. This is in contrast to [17], where separate hardware was used for the $\tilde{\alpha}$ and $\tilde{\beta}$ calculations. This also allows a more heavily pipelined design, which increases the clock frequency, as will be detailed in Section IV-B. Furthermore, the $\tilde{\gamma}$ unit of Figure 16 is used to calculate equation (9) and (14), while the $\tilde{b}^{\mathrm{e}}$ unit is used to calculate the final extrinsic output of equation (13) and (18). This $\tilde{b}^{\mathrm{e}}$ unit is pipelined, enabling the hardware processing element to achieve very similar path lengths for the $\tilde{\alpha}/\tilde{\beta}$ unit and the two $\tilde{b}^{\mathrm{e}}$ unit stages, facilitating high clock frequencies and hence high throughputs and low latencies. The LLRs output from each $\tilde{b}^{\mathrm{e}}$ unit must be interleaved and input to the appropriate hardware processing element. In this work we employ a hardwired interleaver pattern, although our future work will develop a more flexible interleaver that will enable the fully parallel turbo decoder to support different frame lengths and interleaver patterns at runtime. Since each hardware processing element can undertake decoding for both the upper and lower decoder, the interleaver also employs multiplexers for selecting between the hardwired interleaver connections of the upper and lower decoder.

Each hardware processing element is connected to its two neighbors, as well as to the interleaver. More specifically, the '$\tilde{\alpha}$ *in*' and '$\tilde{\beta}$ *out*' signals of Figure 16 connect to the neighboring hardware processing element that processes trellis stages with lower indexes k. Meanwhile, the '$\tilde{\alpha}$ *out*' and '$\tilde{\beta}$ *in*' signals connect to the neighboring hardware processing element that processes trellis stages with higher indexes k. These connections correspond to the $\tilde{\alpha}$ and $\tilde{\beta}$ connections between neighboring pairs of algorithmic blocks in Figures 9b and 10. Since each hardware processing element undertakes decoding for the upper and lower decoder, these signals alternate between conveying the $\tilde{\alpha}^{\mathrm{u}}$ and $\tilde{\alpha}^{\mathrm{l}}$ values, or the $\tilde{\beta}^{\mathrm{u}}$ and $\tilde{\beta}^{\mathrm{l}}$ values in successive half iterations. In the case of the UEC-URC scheme, when a hardware processing element is undertaking the decoding of two URC trellis stages, the signals '$\tilde{\alpha}$ *in*' and '$\tilde{\alpha}$ *out*' are comprised of $\{\tilde{\boldsymbol{\alpha}}_{k-1}, \tilde{\boldsymbol{\alpha}}_{k+N-1}\}$ and $\{\tilde{\boldsymbol{\alpha}}_{k+1}, \tilde{\boldsymbol{\alpha}}_{k+N+1}\}$, respectively. Meanwhile, the signals '$\tilde{\beta}$ *in*' and '$\tilde{\beta}$ *out*' are respectively comprised of $\{\tilde{\boldsymbol{\beta}}_{k+1}, \tilde{\boldsymbol{\beta}}_{k+N+1}\}$ and $\{\tilde{\boldsymbol{\beta}}_{k-1}, \tilde{\boldsymbol{\beta}}_{k+N-1}\}$, in correspondence to the notation used within the URC decoder of Figure 10.

As shown in Equations (5) and (8), the $(\tilde{\alpha} + \tilde{\gamma})$ term is common to both the $\tilde{\alpha}$ calculation of (5) and the $\tilde{b}^{\mathrm{e}}$ calculation of (8). Owing to this, the adders that perform this operation can be efficiently shared between the $\tilde{\alpha}/\tilde{\beta}$ unit and the $\tilde{b}^{\mathrm{e}}$ unit, as shown in Figure 16 and as it will be detailed in Section IV-C.

Since the same hardware performs both the $\tilde{\alpha}$ and $\tilde{\beta}$ calculations in different clock cycles, multiplexers are required for selecting between the inputs $\tilde{\alpha}_n$ and $\tilde{\beta}_n$, as shown in Figure 16. Furthermore, a feedback path is employed across the $\tilde{\alpha}/\tilde{\beta}$ unit for allowing it to calculate two successive $\tilde{\boldsymbol{\alpha}}$ or $\tilde{\boldsymbol{\beta}}$ values in two successive clock cycles. More specifically, this feedback path allows the $\tilde{\alpha}/\tilde{\beta}$ unit to calculate $\tilde{\boldsymbol{\alpha}}_{k+1}$ and $\tilde{\boldsymbol{\beta}}_{k-1}$ based on the results of $\tilde{\boldsymbol{\alpha}}_k$ and $\tilde{\boldsymbol{\beta}}_k$, respectively. At the output of the $\tilde{\alpha}/\tilde{\beta}$ unit, a multiplexer reorders the output state metrics. In the case of the UEC-URC scheme, this is required since the UEC and URC trellises differ from each other. In the case of the LTE turbo scheme, this reordering is required, since the trellis connections for calculating the $\tilde{\boldsymbol{\alpha}}$ values are different from the trellis connections for calculating the $\tilde{\boldsymbol{\beta}}$ values. This reordering allows the $\tilde{\alpha}/\tilde{\beta}$ unit to have fixed connections for both the $\tilde{\alpha}$ and $\tilde{\beta}$ calculations.

Two register stages placed in series are used to implement the $\tilde{\beta}\Delta$ memory of Figure 9. Two registers are required, since the memory must hold the $\tilde{\boldsymbol{\beta}}$ values for both the upper and lower decoder during each algorithmic iteration. Another multiplexer is employed at the input of the $\tilde{b}^{\mathrm{e}}$ unit in order to select which $\tilde{\boldsymbol{\beta}}$ values it is provided with. For steps (b) and (d) of Figure 9, this multiplexer selects the $\tilde{\beta}_{k+1}$ values provided by the neighboring processing element. Meanwhile, in steps (a) and (c) of Figure 9, the $\tilde{\beta}_k$ values are selected from the $\tilde{\beta}\Delta$ memory. Finally, in the case of the URC-UEC decoder, another multiplexer is used to provide the appropriate *a priori* LLRs into the second pipeline stage of the $\tilde{b}^{\mathrm{e}}$ unit, as required by Equation (8). More specifically, when the hardware processing element is undertaking URC decoding, the multiplexer selects either $\tilde{b}^{\mathrm{URC,a}}_{1,k}$ and $\tilde{b}^{\mathrm{URC,a}}_{1,k+N}$ or $\tilde{b}^{\mathrm{URC,a}}_{1,k+1}$ and $\tilde{b}^{\mathrm{URC,a}}_{1,k+N+1}$. Meanwhile, when the hardware processing element is undertaking UEC decoding, the multiplexer selects either $\tilde{b}^{\mathrm{UEC,a}}_{2,k}$ and $\tilde{b}^{\mathrm{URC,a}}_{3,k}$ or $\tilde{b}^{\mathrm{URC,a}}_{2,k+1}$ and $\tilde{b}^{\mathrm{URC,a}}_{3,k+1}$.

B. SCHEDULING

In this section, we propose a novel pipelining technique, in which the hardware processing elements use two clock cycles for processing each of the time periods (a)-(d) of Figure 9. Owing to this, each decoding iteration requires 8 clock cycles in the proposed decoder.

Figure 17 characterizes the timing of the various operations performed by the proposed decoder, illustrating the operation of two hardware processing elements, which we refer to as A and B. More specifically, Figure 17 shows when each hardware processing element performs each of the different operations of Figure 9. Note that the flow of data is the same in both Figures 9 and 17, but Figure 17 shows how the hardware implements this data flow on a clock cycle by clock cycle basis, allowing the pipelining techniques to be shown. Note that while Figure 17 adopts the notation of the LTE scheme, the operation of the UEC-URC scheme is identical.

Figure 17 shows how the hardware processing elements alternate between performing processing for the upper and lower decoders in successive half-iterations. More specifically, $N/4$ hardware processing elements, including blocks A and B, use four clock cycles for performing steps (a) and (b) of Figure 9, where each hardware processing

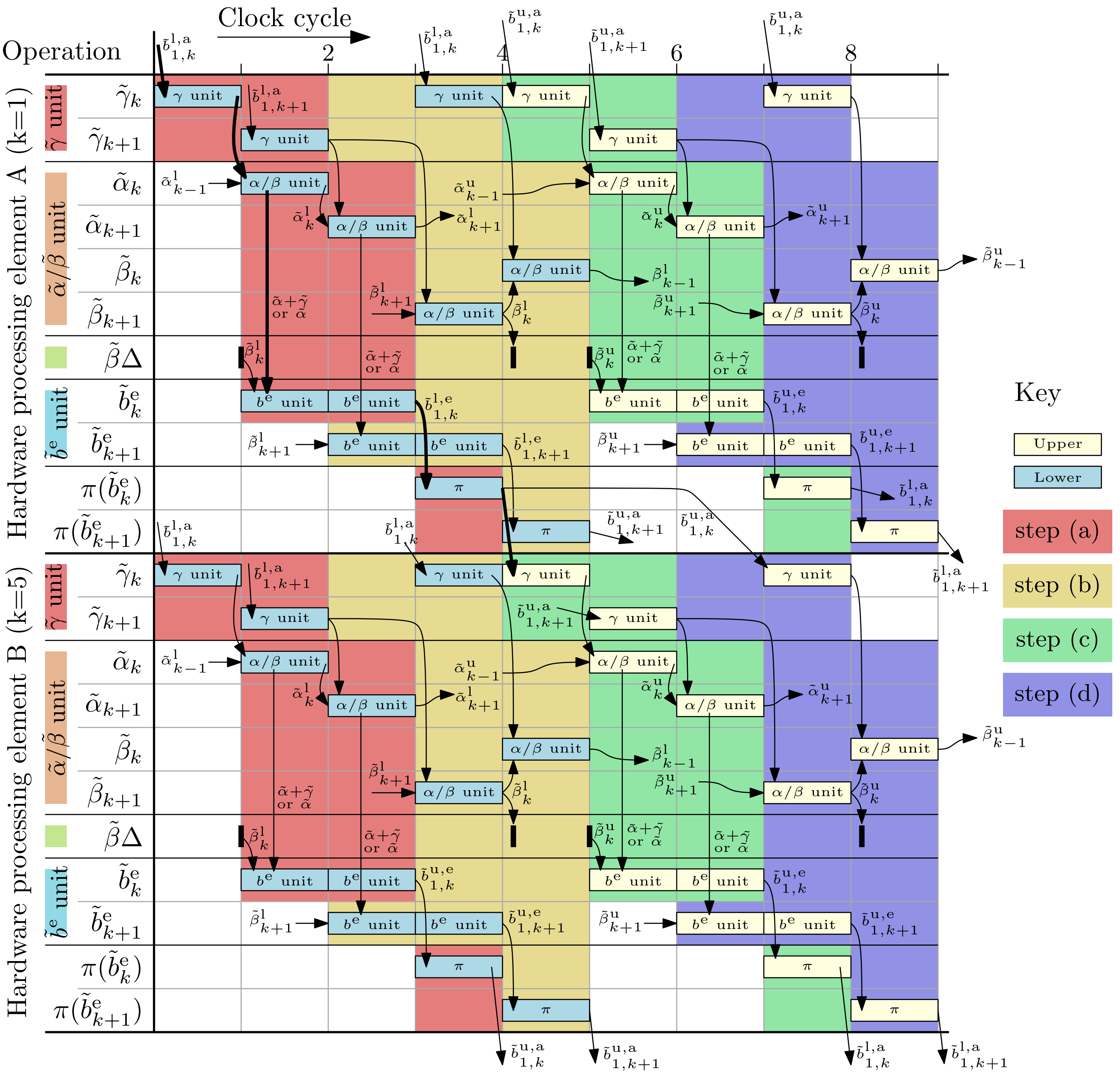


FIGURE 17. Timing diagram for two decoder blocks during one iteration of the LTE decoding algorithm. These decoding blocks perform the operations associated with different bit indexes of the upper or lower decoder in the same time periods. Owing to this, the output $\tilde{b}^e_{1,k}$ provided by hardware processing element A is shown being interleaved to the input $\tilde{b}^a_{1,k}$ of hardware processing element B. In the example of Figure 9, hardware processing element A processes the bits having the indices $k \in \{1, 2\}$, while hardware processing element B processes bits $k \in \{5, 6\}$. For each decoder, the diagram shows when each of the different tasks are undertaken, as well as the transfer of data between the tasks according to the dependencies between them. The background shading identifies which step of Figure 9 each operation corresponds to.

element undertakes the processing tasks for two trellis stages. Following this, these hardware processing elements use four clock cycles to perform steps (c) and (d) of Figure 9, where each hardware processing element undertakes the processing for two trellis stages of the lower decoder. At the same time, the other $N/4$ hardware processing elements each perform steps (a) and (b) of Figure 9 for two trellis stages of the lower decoder, then perform steps (c) and (d) of Figure 9 for two trellis stages of the upper decoder.

As shown in Figure 7b, the upper decoder's extrinsic LLRs are interleaved to become the lower decoder's *a priori* LLR inputs. Accordingly, Figure 17 provides an example of how the extrinsic LLRs gleaned from processing element A may be passed to processing element B through the interleaver. More specifically, Figure 17 shows that an extrinsic LLR arriving from the $k = 1^{\text{st}}$ block in the upper decoder is interleaved and entered into the $k = 5^{\text{th}}$ block of the lower decoder. This example illustrates the critical path in the flow of information through the different operations, where the 4-stage pipeline transversing through the decoder is shown for one bit by the bold arrows. This example also shows that the *mod4* interleaver property of Figure 11 is key to facilitating the 4-stage pipelining technique proposed in this section. More specifically, if the extrinsic LLR produced by one algorithmic block in group 1 of Figure 11 was interleaved to an algorithmic block in group 0 of Figure 11, then the *a priori* LLRs output from the interleaver would arrive one clock cycle too late to be used. This can be seen in Figure 17, where the extrinsic LLR $\tilde{b}^{\mathrm{l,e}}_{1,k+1}$ output from hardware processing element A is not interleaved to $\tilde{b}^{\mathrm{u,a}}_{1,k+1}$ in time to be used at the start of step (c) by hardware processing element B, as may be required by an interleaver that does not have the *mod4* property. Instead of being consumed immediately, these *a priori* LLRs would need to be stored in an additional memory, until the next opportunity to use them arose. In addition to this additional hardware requirement, this delay would degrade the decoders error correction capability. By contrast, the *mod4 type A* interleaver ensures that these *a priori* LLRs do not need storing, since they are consumed immediately.

C. SCHEME-SPECIFIC IMPLEMENTATION

This section describes the specific features of the FPGA implementation that are used when implementing the LTE turbo decoder scheme of Figure 3, as well as the UEC-URC decoder scheme of Figure 5. In particular, we detail the components of each hardware processing block that behave differently for the pair of schemes considered.

In contrast to the LTE turbo code of Figure 3, Figure 6 shows that the UEC and URC decoders employ two different trellises, both of which must be implemented using the same hardware in order to maintain a high hardware efficiency. As described in Section II-B2, the UEC decoder comprises N trellis stages, where each trellis stage has $r_{\mathrm{UEC}} = 4$ states. During the processing of each trellis stage, two extrinsic LLRs $\tilde{b}^{\mathrm{e,UEC}}_{2,j}$ are generated. By contrast, the URC decoder comprises $2N$ trellis stages, where each trellis stage has $r_{\mathrm{URC}} = 2$ states. Here, each trellis stage generates only one extrinsic LLR $\tilde{b}^{\mathrm{e,URC}}_{1,k}$. In order to maintain a high hardware efficiency, each hardware processing unit is designed for processing one UEC trellis stage at a time, or process two URC trellis stages at a time. Since each UEC trellis stage corresponds to the same number of states, transitions and extrinsic LLRs as two URC trellis stages, both the UEC and URC trellises can be efficiency processed by the same hardware, as will be described in the following sections.

1) $\tilde{\gamma}$ UNIT

Figure 18 illustrates the $\tilde{\gamma}$-calculation unit both for the LTE and for the UEC-URC implementation, which produces the *a priori* transition probabilities $\tilde{\gamma}$ according to (9) or (14). In the case of the UEC-URC scheme, Figure 18b employs the four multiplexers with gray shading on the $\tilde{\gamma}_{1b}$, $\tilde{\gamma}_{2a}$, $\tilde{\gamma}_{3b}$ and $\tilde{\gamma}_{4a}$ outputs, in order to switch the pairs of $\tilde{\gamma}$ values that are output depending on whether the $\tilde{\alpha}/\tilde{\beta}$ unit is currently decoding $\tilde{\alpha}$ values or $\tilde{\beta}$ values. For example, in the URC trellis of Figure 6b, the $\alpha_k(1)$ calculation requires $\tilde{\gamma}(1,1)$ and $\tilde{\gamma}(2,1)$, while the $\beta_k(1)$ calculation requires $\tilde{\gamma}(1,1)$ and $\tilde{\gamma}(1,2)$, necessitating the $\tilde{\gamma}$ unit to swap some of the $\tilde{\gamma}$ values. Note that the corresponding multiplexers are not required for the LTE turbo decoder scheme, since the different connections for the $\tilde{\boldsymbol{\alpha}}$ values and $\tilde{\boldsymbol{\beta}}$ values can be handled by the reordering scheme of Figure 16.

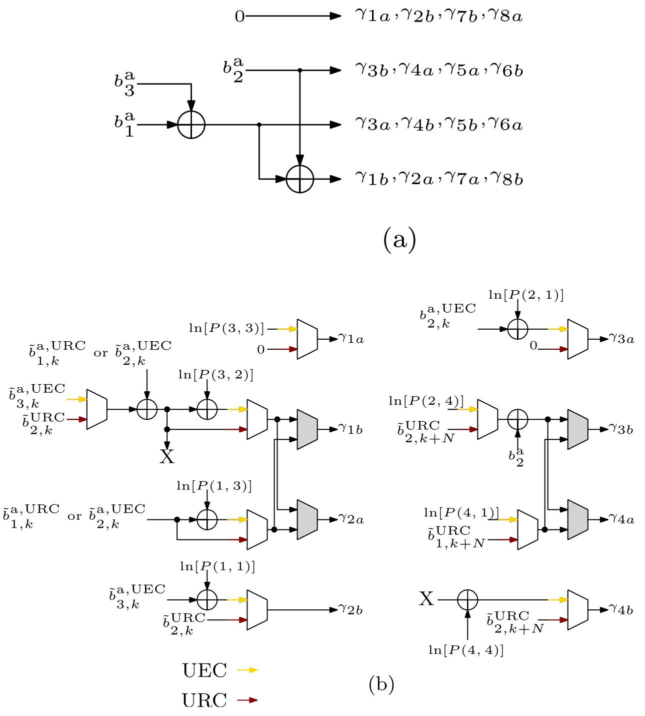

0
$\gamma_{1a}, \gamma_{2b}, \gamma_{7b}, \gamma_{8a}$
b^{a}_2
b^{a}_3
$\gamma_{3b}, \gamma_{4a}, \gamma_{5a}, \gamma_{6b}$
b^{a}_1
$\gamma_{3a}, \gamma_{4b}, \gamma_{5b}, \gamma_{6a}$
$\gamma_{1b}, \gamma_{2a}, \gamma_{7a}, \gamma_{8b}$
(a)
$\ln[P(3,3)]$
$\tilde{b}^{\mathrm{a,URC}}_{1,k}$ or $\tilde{b}^{\mathrm{a,UEC}}_{2,k}$
0
γ_{1a}
$\ln[P(3,2)]$
$\tilde{b}^{\mathrm{a,UEC}}_{3,k}$
$\tilde{b}^{\mathrm{URC}}_{2,k}$
X
γ_{1b}
$\ln[P(1,3)]$
$\tilde{b}^{\mathrm{a,URC}}_{1,k}$ or $\tilde{b}^{\mathrm{a,UEC}}_{2,k}$
γ_{2a}
$\ln[P(1,1)]$
$\tilde{b}^{\mathrm{a,UEC}}_{3,k}$
$\tilde{b}^{\mathrm{URC}}_{2,k}$
γ_{2b}
$\ln[P(2,1)]$
$b^{\mathrm{a,UEC}}_{2,k}$
0
γ_{3a}
$\ln[P(2,4)]$
$\tilde{b}^{\mathrm{URC}}_{2,k+N}$
b^{a}_2
γ_{3b}
$\ln[P(4,1)]$
$\tilde{b}^{\mathrm{URC}}_{1,k+N}$
γ_{4a}
X
$\tilde{b}^{\mathrm{URC}}_{2,k+N}$
$\ln[P(4,4)]$
γ_{4b}
UEC
URC
(b)

FIGURE 18. Schematic of the $\tilde{\gamma}$-calculation unit of Figure 16 for the (a) LTE and (b) UEC-URC scheme. Here, the 'a' subscript refers to the $\tilde{\gamma}_m$ value for the top-most transition which leaves state *m* on the trellis, while the 'b' subscript refers to the bottom-most transition.

In the UEC-URC scheme, each hardware processing element has to carry out both UEC and URC decoding, requiring the unshaded multiplexers of Figure 18b to switch between the different inputs that are necessary for UEC and URC decoding. This is necessary since the transitions of the UEC and URC trellises to not share the same inputs and outputs. Furthermore, the UEC trellis decoder also requires the addition of the conditional transition probabilities $\ln[P(m|m')]$, as described in Section II-B2. In the proposed implementation, these conditional transition probabilities are stored using 6-bit fixed point numbers.

2) $\tilde{\alpha}/\tilde{\beta}$ UNIT

The $\tilde{\alpha}/\tilde{\beta}$ units are characterized by the orange boxes in the top half of Figure 19, which detail the internal operation of the orange block of Figure 16. In the case of the UEC-URC scheme, each hardware processing element

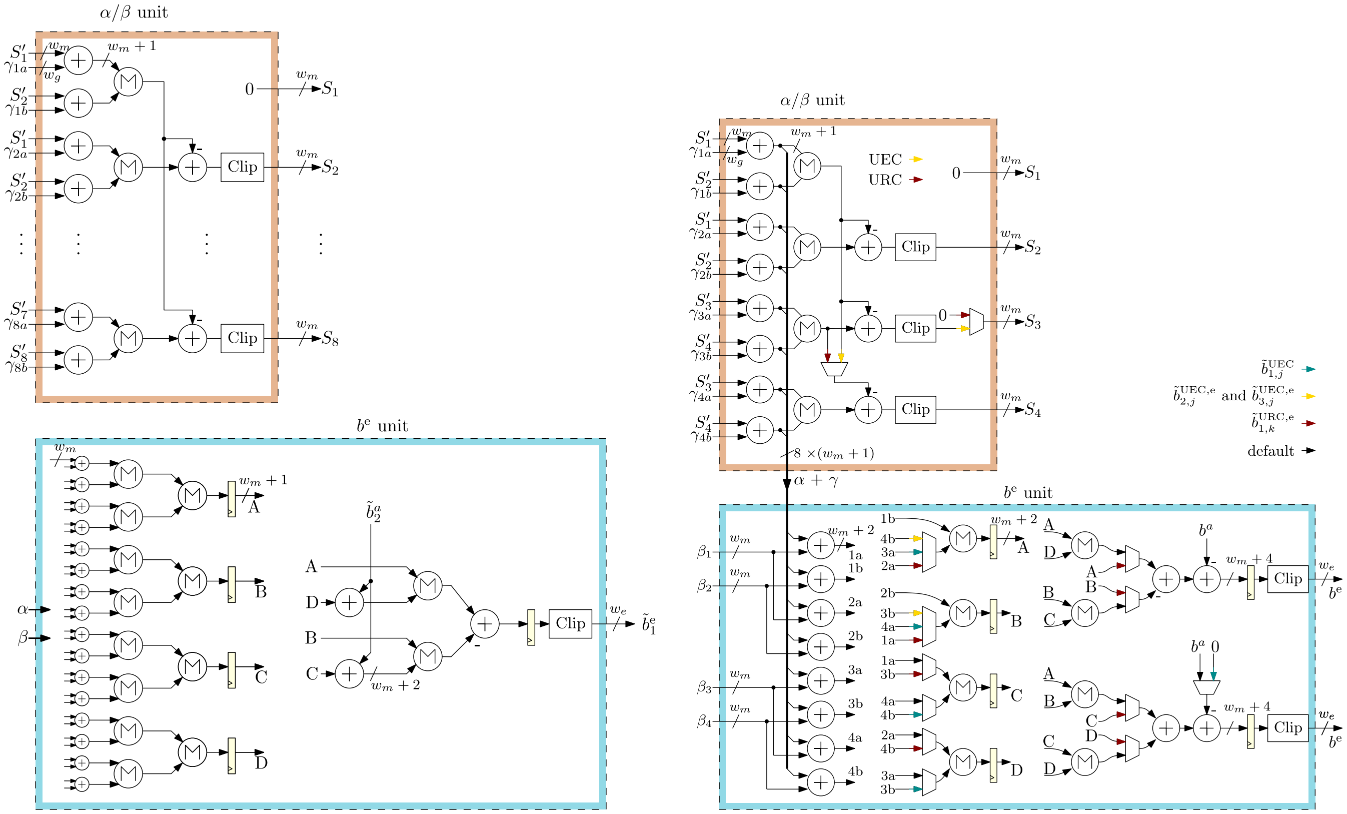


FIGURE 19. The $\tilde{\alpha}/\tilde{\beta}$ unit and $\tilde{b}^{\mathrm{e}}$ unit for (a) the LTE turbo scheme, and (b) the UEC-URC scheme. Here, the colored outlines correspond to the similarly colored blocks of Figures 10, 16 and 17.

switches between the UEC trellis decoder and URC decoder every 4 clock cycles. Accordingly, the $\tilde{\alpha}/\tilde{\beta}$ unit is designed to switch between carrying out all of the operations of either one UEC trellis stage or of two URC trellis stages. As shown in Figure 6, two URC trellis stages can be processed at the same time to give a similar structure to one UEC trellis stage, but with some different transition connections. More specifically, the trellis of Figure 6a can be converted into two copies of the trellis of Figure 6b by simply switching the transitions associated with the central two states. Furthermore, by switching these two central states, the trellis may be mirrored from left to right, allowing the same connections to be used for both $\tilde{\alpha}$ and $\tilde{\beta}$ calculations. This switching of trellis states is implemented using the multiplexers of Figure 16. In the case of the LTE scheme, state switching is also undertaken by multiplexers shown in Figure 16, in order to produce a mirrored trellis, allowing the same connections to be used for both $\tilde{\alpha}$ and $\tilde{\beta}$ calculations.

The $\tilde{\alpha}/\tilde{\beta}$ unit's inputs S' all have bit widths of w_m, as investigated in Section III-E. However, the bit-widths within the $\tilde{\alpha}/\tilde{\beta}$ unit grow following successive additions, in order to ensure that they do not overflow. Following this, clipping is employed to restore the bit widths of the outputs S to w_m. Note that the two multiplexers in the UEC-URC $\tilde{\alpha}/\tilde{\beta}$ unit of Figure 19b change the operation of the clipping circuit, depending on whether outputs 3 and 4 are part of the same trellis as outputs 1 and 2, as in the case of the UEC, but not for the URC.

3) $\tilde{b}^{\mathrm{e}}$ UNIT

Figure 19 illustrates the $\tilde{b}^{\mathrm{e}}$-calculation unit of both the LTE and UEC-URC schemes, which is used for generating the extrinsic LLRs. Here, we pipeline the $\tilde{b}^{\mathrm{e}}$ unit into two stages, in order to facilitate a high clock frequency and hence a high hardware efficiency. More specifically, this pipelining ensures that the propagation delay in each of the two $\tilde{b}^{\mathrm{e}}$ stages is of a similar length to those of the other parts of the decoder. This pipelining scheme was detailed in Section IV-B, which considered the clock cycle by clock cycle scheduling of each hardware processing element. The connections required within the $\tilde{b}^{\mathrm{e}}$ unit for generating extrinsic LLRs are dictated by the specifics of the LTE, URC or UEC trellis. More particularly, the input and output bits associated with each transition identify which specific state and transition metrics have to be combined, according to (12) and (17). In the case of the UEC-URC scheme, the required combinations of state and transition metrics are different, depending on whether the $\tilde{b}^{\mathrm{e}}$ unit is generating $\tilde{b}^{\mathrm{UEC}}_{1,k}$, $\tilde{b}^{\mathrm{e,UEC}}_{2,k}$ or $\tilde{b}^{\mathrm{e,URC}}_{1,k}$. The required flexibility is provided by multiplexers in the first pipeline stage of Figure 19, which are used for selecting which particular pairs of metrics are input to the max calculation, as well as by multiplexers in the second stage, which bypass the second max calculation, since it is not required in the case of the URC. The final subtraction in (13) and (18) is undertaken before the register in the second pipeline stage. Since the second $\tilde{b}^{\mathrm{e}}$ output can be used for generating either the *a posteriori* LLR b_1^{UEC} or the extrinsic LLR $\tilde{b}_3^{\mathrm{e,UEC}}$, a multiplexer is required for selection between '0' or the *a priori* LLR $\tilde{b}_3^{\mathrm{a,UEC}}$, respectively.

In the case of the LTE scheme, the operations of Figure 19a have been reordered [17] compared to those of the UEC-URC scheme, in order to reduce the critical path-length, but at the expense of a slightly increased hardware resource requirement. This technique dispenses with adding $\tilde{\gamma}$ values during the $\tilde{\delta}$ calculation of (12) and (17) as well as with subtracting the *a priori* LLR in the $\tilde{b}^{\mathrm{e}}$ calculation of (13) and (18). Instead, the LLR $\tilde{b}_2^{\mathrm{a}}$ is added during the $\tilde{b}^{\mathrm{e}}$ calculation in Figure 19. Note that this technique cannot be used for the UEC-URC scheme, owing to the conditional probabilities $\ln[P(m_{k-1}|m_k)]$, which arise from the non-equiprobable transitions of the UEC code. Instead, the UEC-URC scheme passes the $(\tilde{\alpha} + \tilde{\gamma})$ values from the $\tilde{\alpha}/\tilde{\beta}$ unit to the $\tilde{b}^{\mathrm{e}}$ unit, in order to reduce the required hardware resources.

TABLE 2. Comparison of the proposed approach with the state-of-the-art Log-BCJR decoder and the fully parallel decoder of [15], when used to decode the $N = 6144$-bit LTE turbo code.

	Estimation in [17]		Estimation in this work
	State-of-the-art LTE algorithm of [18]	LTE fully parallel of [15], [17], [16]	Proposed paired schedule
Number of parallel hardware processing elements	64	$N^{\star}$	$N/2$
Clock cycles per iteration T	$N/32$	2	8
Clock cycle duration D	3 stages	6 stages	3 stages
Complexity per decoding iteration C	$320N$	$155N$	$155N$
Decoding iterations I	6	39	28
Combinational requirement X	14144	$80N$	$30N$
Register requirement Y	1792	$21N$	$26N$
RAM requirement Z	$14N/3$+8320	0	0
Overall throughput $N/(T \cdot D \cdot I)$	16/9 (1x)	$N/468$ (7.38x)	N/672 (5.14x)
Overall latency ($T \cdot D \cdot I$)	$9N/16$ (7.38x)	468 (1x)	672 (1.44x)
Overall complexity $(C \cdot I)$	$1920N$ (1x)	$6045N$ (3.15x)	$4340N$ (2.26x)
Overall resource $w \max(\frac{X}{2}, \frac{Y}{2}, \frac{Z}{160})$	$7072w$ (1x)	$40Nw$ (34.8x)	$15Nw$ (13.0x)
Hardware efficiency (resource/throughput)	3978 (1x)	18720 (4.71x)	10080 (2.53x)

*The work of [17] uses $2N$ processing elements, which offers a reduced power consumption but increased area.

D. COMPARISON TO SCHEDULING IN THE EXISTING STATE-OF-THE-ART IMPLEMENTATIONS

Table 2 characterizes the performance of the proposed algorithmic and hardware scheduling, and compares this both to the state-of-the-art Log-BCJR [18] implementation and to the fully parallel decoder of [15], [17], when implementing the LTE turbo decoder. Compared to the fully parallel turbo decoder of [15], [17], our proposed implementation requires 4x as many clock cycles per iteration, owing to two design decisions. Firstly, each processing element in the proposed approach decodes two trellis stages, while the processing elements of [15], [17] only decode a single trellis stage. This allows our proposed decoder to support longer frame lengths N than the design of [15], [17] using the same amount of hardware. Secondly, the hardware scheduling of the proposed approach contains more pipelining, which results in a clock cycle duration D that is half that of [17]. Furthermore, our additional pipelining improves the hardware reuse, therefore reducing the required hardware resources whilst increasing hardware efficiency, without reducing the throughput.

The complexity C per decoding iteration of the proposed approach is the same as that of [15], [17], as shown in Table 2. This is because both implementations perform the same operations of (4)-(8), although here we propose a different activation order for these equations. As explored in Section III-C, our novel scheduling means that the proposed approach achieves the same BER performance as the benchmarkers using only 28 decoding iterations I, in contrast to the 39 required by the fully-parallel turbo decoder of [17].

Table 2 also characterizes the breakdown of the hardware resource requirements into combinational X, registers Y and RAM Z requirements. As detailed in [15], the combinational requirement X is obtained by quantifying the number of adder and max circuits employed, while, the register requirement Y is quantified in terms of the number of values that must be held between successive clock cycles. Finally, Z is determined by quantifying the number of values that must be stored in RAM. The ASIC implementation of [17] comprises $2N$ hardware processing elements, each dedicated to one trellis stage of either the upper or lower decoder. This decoder operates on the basis of the odd-even activation order and so each hardware processing element is inactive in alternate clock cycles. The reduced switching of this approach reduces energy consumption, but leads to a larger hardware requirement. Meanwhile, the FPGA implementation of [16] comprises N hardware processing elements, each of which alternate between performing decoding for both the upper and lower decoder. By contrast, the proposed decoder comprises $N/2$ hardware processing elements, each of which processes two trellis stages of the upper decoder and two trellis stages of the lower decoder. This difference leads to a combinational requirement X that is half of that required by the fully parallel turbo decoder of [16], for a given frame length N. The combinational requirement X is further reduced in the proposed decoder by reusing the same hardware for the $\tilde{\alpha}$ and $\tilde{\beta}$ calculations, as discussed in Section IV-B. Note that owing

TABLE 3. FPGA implementation of the proposed LTE turbo decoder when performing 28 decoding iterations as well as of the proposed UEC-URC decoder when performing 20 iterations using the Altera Stratix IV EP4SE230 FPGA. Here, the combinational and register utilization quantifies the percentage of the 182,400 ALUTs and registers used, respectively.

	LTE decoder							UEC-URC decoder							
Frame length N	48	160	200	320	360	400	440	48	160	240	320	400	440	460	500
Interleaver type	B	A	B	A	B	B	A	A	A	A	A	A	A	A	A
Clock frequency (MHz)	195	189	182	179	170	171	156	202.5	194	177.6	173	165.7	161.9	156.5	140.9
Total Utilization (%)	11	35	43	70	78	84	98	10	34	52	65	81	88	89	100
Combinational Utilization (%)	9	30	37	60	68	75	83	9	29	44	58	72	79	82	89
Register Utilization (%)	5	17	22	35	39	44	48	6	18	27	37	45	50	52	56
Throughput (Mbps)	42	135	163	256	273	305	306	61	194	266	346	414	445	450	440
Latency (μs)	1.15	1.19	1.23	1.25	1.32	1.31	1.44	0.79	0.82	0.9	0.92	0.97	0.99	1.02	1.14

to its increased use of pipelining, the proposed design requires more registers per hardware processing element than the fully-parallel turbo decoder of [15], [17]. However, Section V will demonstrate that the combinational requirement is the limiting factor in the FPGA implementation of both the proposed and benchmarker designs. Owing to this, the use of more registers to achieve superior performance represents a more desirable utility of the FPGA resources.

The combinational requirement X, register requirement Y and RAM requirement Z may be combined to predict the overall resource requirement of each design considered. Here, we target the Altera Stratix IV FPGA, which is comprised of a large number of Adaptive Logic Modules (ALMs). Each ALM comprises two single-bit registers and two single-bit adders with a corresponding Look-Up Table (LUT). Since the combinational requirement X represents the number of additions and max operations required, we may estimate that $wX/2$ ALMs are required to fulfill the combinational requirement, where w is the average bit-width used in the decoder. Likewise, the number of ALMs required to fulfill the register requirement may be estimated by $wY/2$. In the Stratix IV device targeted by this work, there are 160 bits of RAM available for each ALM. Therefore, the total resource requirement may be approximated by $w\max(\frac{X}{2}, \frac{Y}{2}, \frac{Z}{160})$. Note that we combine the three elements using the maximum, since the FPGA has a limited amount of each hardware resource type and one of these will inevitably impose the ultimate limitation, as the degree of parallelism is increased [37]. Note that this analysis can only provide an approximation, but it may be applied equally to the three designs considered, allowing a fair comparison. This analysis provides a lower bound on the required hardware, since it does not consider the resources required by multiplexers or the restrictions imposed by routing, which may lead to the inefficient exploitation of resources. As shown in Table 2, this analysis reveals that the overall resource requirement of the proposed design is just 25% of that of the fully-parallel turbo decoder of [15], [17].

Table 2 shows that the overall throughput of our proposed design is 5 times greater than that of the state of the art Log-BCJR decoder of [18], but 0.70 times that of the fully parallel turbo decoder of [17] for a given frame length N. Although the proposed design requires fewer decoding iterations than that of [17], this reduced throughput may be expected since each decoding iteration of the proposed design has a duration *TD* which is double that of [17]. Note that this increased decoding iteration duration *TD* also results in a slightly longer latency than that of [17]. However, owing to the significantly reduced hardware requirement of the proposed design, its overall hardware efficiency is almost double that of the fully parallel turbo decoder of [15]–[17]. While the hardware efficiency of the proposed design is half that of the state-of-the-art Log-BCJR decoder of [18], we have achieved a significantly higher throughput and a much lower latency.

V. RESULTS

In this section, we characterize the FPGA implementation of the proposed LTE turbo decoder and UEC-URC decoder of Figures 3 and 5, respectively. Table 3 shows the key performance criteria of the proposed implementations using a midrange Altera Stratix IV EP4SE230 FPGA with 91k ALMs [38]. This table characterizes the proposed FPGA implementation of the LTE turbo decoder of Figure 3 using 28 decoding iterations, which was found in Section III-C to offer the same error correction capability as a Log-BCJR decoder performing 6 decoding iterations. Table 3 also characterizes the proposed FPGA implementation of the UEC-URC scheme of Figure 5 using 20 decoding iterations.

As shown in Table 3, the proposed LTE turbo decoder implementation achieves a maximum throughput of 306 Mbps with a latency of 1.44 μs. Table 3 also shows that the proposed UEC-URC decoder implementation achieves a maximum throughput of 450 Mbps at a latency of 1 μs, which readily fulfills the requirements of low latency, high throughput video applications. Table 3 characterizes the combinational and register utilization, which quantify the percentage of the FPGA's combinational Adaptive Look-Up Tables (ALUTs) and registers used, respectively. Table 3 also characterizes the total hardware utilization of the proposed

LTE and UEC-URC decoders, where the percentage of ALMs used by the designs are quantified. Note that each ALM comprises two ALUTs and two registers. However, due to routing constraints, the percentage of ALMs used is higher than the maximum of the combinational and register usage, since the FPGA tool cannot produce the most area efficient design without severely degrading the clock frequency. Note that the throughput drops slightly as the design reaches 100% hardware utilization, when longer frames are targeted. This may be explained by congestion within the FPGA, which also degrades the achievable clock frequency. Note that the Stratix IV EP4SE230 FPGA used for implementing the design is a mid-range device, whilst more powerful FPGAs containing up to 325k ALMs are also available in the Stratix IV series. For the LTE turbo decoder implementation, a maximum frame length of $N = 440$ is supported. In contrast, the FPGA fully parallel turbo decoder of [16] achieved a frame length of $N = 720$, using a FPGA with 3.5 times more resources available. An FPGA was used to confirm the SER performance results of Figure 15, which were obtained in simulation.

Note that there is some discrepancy when comparing both the combinational requirement X and register requirement Y - which are quantified in Table 2 - to the actual implementation results of Table 3. More specifically, since the analysis of Table 2 only considers the combinational contribution of the addition and max operations in the datapath, this analysis underestimates the actual combinational requirement by not considering other combinational contributions, such as that from multiplexers in the datapath, the multiplexers in the interleaver, or the logic circuits required by the controller. Table 2 predicts the register contribution more closely, since the only registers not considered in the analysis of Table 2 are those required by the controller. Note that these discrepancies exist for both the proposed architecture and the estimation of [16], however the analysis of Table 2 is useful for comparing the different algorithms before their implementation.

Since this paper presents the first FPGA implementation of a UEC-URC decoder, it cannot be compared with any previous work. However, we may compare the proposed FPGA implementation of the LTE turbo decoder with those of [16] and [39]. More specifically, Table 4 characterizes the proposed FPGA implementation of the LTE decoder and compares this work with the FPGA-based fully parallel turbo decoder of [16], as well as the FPGA-based implementation of the conventional Log-BCJR turbo decoder of [39]. Table 4 characterizes the performance of the proposed turbo decoder when performing $I = 20$ iterations, which is the number required to offer the same error correction capability as $I = 28$ iterations of the decoder of [16] and $I = 5$ iterations of the decoder of [39]. Here, we compare the FPGA hardware utilization using only the combinational utilization, which is quantified by the number of ALUTs used. While this disregards the register and RAM usage, it is the combinational usage that imposes the greatest resource requirement upon all three designs, therefore constituting the limiting factor of both the design size and the grade of parallelism. Furthermore, [39] does not quantify the overall device utilization, preventing any other comparison. In order to compare the relative performance of the three considered FPGA implementations, Table 4 characterizes the throughput when the decoders perform a fixed number of iterations I, without early stopping. We note however that the fully parallel turbo decoder implementation of [16] is capable of using a CRC check to detect when the iterative decoding process has been successful and can be stopped early. Furthermore, the implementation of [39] is fully flexible and can support all the frame lengths specified by the LTE standard, while the proposed implementation and the work of [16] can only support a single frame length.

TABLE 4. **Comparison between the proposed FPGA implementation of the LTE turbo decoder, an FPGA implementation of the fully parallel turbo decoder of [16], as well as the state of the art FPGA implementation of the conventional Log-BCJR turbo decoder.**

	This work (N=400)	Fully parallel of [16]	Log-BCJR of [39]
FPGA	EP4SE230 (182 kALUT)	EP4SE820 (650 kALUT)	EP4SE820 (650 kALUT)
Iterations I	20	28	5
Clock frequency (MHz)	171	65	102
Resource usage (kALUT)	137	644	254
Throughput after I iterations (Mbps)	425	835	524
Hardware efficiency (kALUT/Mbps)	0.32	0.77	0.48

Of the three FPGA implementations compared, it is the proposed design that achieves the best hardware efficiency of 0.32 kALUT/Mbps, compared to 0.77 kALUT/Mbps for the fully parallel turbo decoder of [16] and 0.48 kALUT/Mbps for the Log-BCJR turbo decoder of [39]. Indeed, the turbo decoding algorithm and architecture proposed here achieves a 2.4-fold improvement in hardware efficiency over those of the fully parallel turbo decoder of [16], which is similar to the expected gain predicted in Section IV-D. Our algorithm and architecture achieves this gain by requiring fewer decoding iterations and by employing more efficient pipelining, which results in a much higher clock frequency. Furthermore, compared to the fully-parallel FPGA LTE turbo decoder implementation of [16], the register utilization of the proposed implementation is much closer to its combinational utilization, demonstrating better usage of the FPGA's resources.

Figure 20 shows a comparison of the performance characteristics of the proposed architecture and the architectures of the benchmarkers. This diagram considers the throughput achievable for a given frame length; the area efficiency; the flexibility of the architecture to support different interleaver designs; the maximum frame length supported given a limited amount of hardware resource; the energy efficiency; and the

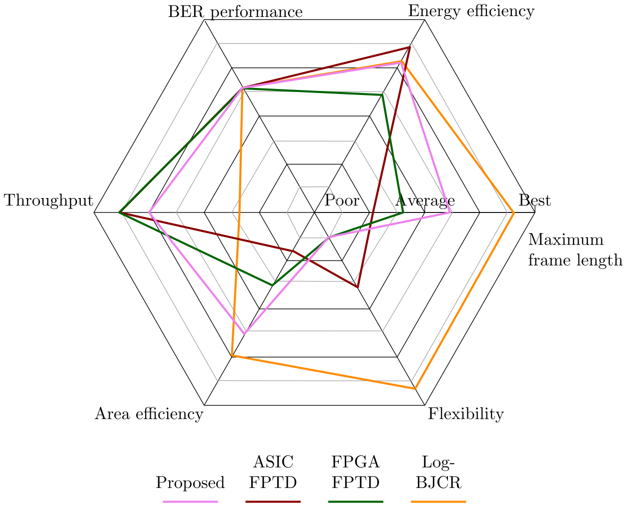


FIGURE 20. A comparison of the key performance characteristics of the proposed paired scheduling decoder, the ASIC [17] and FPGA [16] implementations of the FPTD, and the state-of-the-art BCJR decoder [39].

BER performance. Here, all schemes can achieve the same BER performance, albeit after performing a differing number of iterations, which impacts upon the other characteristics. This diagram shows that the ASIC FPTD architecture of [17] has traded-off reduced area efficiency and maximum frame length, in order to achieve better energy efficiency compared to the FPGA FPTD architecture of [16]. Likewise, the proposed paired scheduling architecture offers a slightly reduced throughput for a given frame length, but with the advantage of being able to support a larger maximum frame length and greater area efficiency, compared to the FPTD architectures of [16] and [17].

VI. CONCLUSIONS

In this paper we have shown that in addition to its application in the LTE turbo decoder, the fully parallel iterative decoder may be extended to the decoding of a UEC-URC code. We have shown for both the LTE turbo code and UEC-URC code that the fully parallel scheduling can be modified to allow each processing block to operate two trellis stages. This enables improved pipelining and reduces the number of decoding iterations required for achieving strong error correction, leading to a 2.4-fold hardware efficiency improvement compared to the implementation of the original fully parallel decoding approach. In particular, we have jointly considered the algorithm and its implementation. By reusing the same hardware to process both the forward state metrics $\tilde{\alpha}$ and $\tilde{\beta}$, the hardware resource requirement is significantly reduced and pipelining can be used without impacting upon the error correction performance. This novel pipelining technique also enables a better utilization of the FPGA's resources, by making a more equal use of register and combinational resources, compared to previous designs which under-used the available register hardware. This technique also allows significantly longer frame lengths to be supported within a given FPGA. Our implementation achieves a throughput of 306 Mbps and a latency of 1.44 μs for the LTE turbo decoder, as well as a throughput of 450 Mbps with 1.1 μs latency for the UEC-URC code, when targeting a midrange FPGA.

REFERENCES

[1] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Jul. 1948.

[2] J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM J. Res. Develop.*, vol. 23, no. 2, pp. 149–162, Mar. 1979.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proc. IEEE Int. Conf. Commun.*, vol. 2. Geneva, Switzerland, May 1993, pp. 1064–1070.

[4] M. F. Brejza, L. Li, R. G. Maunder, B. M. Al-Hashimi, C. Berrou, and L. Hanzo, "20 years of turbo coding and energy-aware design guidelines for energy-constrained wireless applications," *IEEE Commun. Surveys Tut.*, vol. 18, no. 1, pp. 8–28, 1st Quart., 2016.

[5] M. Fresia, F. Perez-Cruz, H. V. Poor, and S. Verdu, "Joint source and channel coding," *IEEE Signal Process. Mag.*, vol. 27, no. 6, pp. 104–113, Nov. 2010.

[6] V. Buttigieg and P. G. Farrell, "Variable-length error-correcting codes," *IEE Proc.-Commun.*, vol. 147, no. 4, pp. 211–215, Aug. 2000.

[7] R. G. Maunder, W. Zhang, T. Wang, and L. Hanzo, "A unary error correction code for the near-capacity joint source and channel coding of symbol values from an infinite set," *IEEE Trans. Commun.*, vol. 61, no. 5, pp. 1977–1987, May 2013.

[8] W. Zhang, R. G. Maunder, and L. Hanzo, "On the complexity of unary error correction codes for the near-capacity transmission of symbol values from an infinite set," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Apr. 2013, pp. 1–6.

[9] T. Wang, W. Zhang, R. G. Maunder, and L. Hanzo, "Near-capacity joint source and channel coding of symbol values from an infinite source set using Elias gamma error correction codes," *IEEE Trans. Commun.*, vol. 62, no. 1, pp. 280–292, Jan. 2014.

[10] W. Zhang, M. F. Brejza, T. Wang, R. G. Maunder, and L. Hanzo, "Irregular trellis for the near-capacity unary error correction coding of symbol values from an infinite set," *IEEE Trans. Commun.*, vol. 63, no. 12, pp. 5073–5088, Dec. 2015.

[11] M. A. Bernard and B. D. Sharma, "Some combinatorial results on variable length error-correcting codes," in *Proc. ARS Combinatoria*, 1988, pp. 181–194.

[12] J. L. Massey, "Joint source and channel coding," in *Communication Systems and Random Process Theory*, J. K. Skwirzynski, Ed. Amsterdam, The Netherlands: Sijthoff and Noordhoff, Dec. 1978, pp. 279–293.

[13] R. Bauer and J. Hagenauer, "Symbol-by-symbol MAP decoding of variable length codes," in *Proc. 3rd ITG Conf. Source Channel Coding*, 2000, pp. 111–116.

[14] N. Gortz, "Iterative source-channel decoding using soft-in/soft-out decoders," in *Proc. IEEE Int. Symp. Inf. Theory*, Sorrento, Italy, Jun. 2000, p. 173.

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

[16] A. Li, P. Hailes, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "1.5 Gbit/s FPGA implementation of a fully-parallel turbo decoder designed for mission-critical machine-type communication applications," *IEEE Access*, vol. 4, pp. 5452–5473, 2016.

[17] A. Li, L. Xiang, T. Chen, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "VLSI implementation of fully parallel LTE turbo decoders," *IEEE Access*, vol. 4, pp. 323–346, Jan. 2016. [Online]. Available: http://eprints.soton.ac.uk/386016/

[18] T. Ilnseher, F. Kienle, C. Weis, and N. Wehn, "A 2.15GBit/s turbo code decoder for LTE advanced base station applications," in *Proc. 7th Int. Symp. Turbo Codes Iterative Inf. Process. (ISTC)*, Aug. 2012, pp. 21–25.

[19] C. Studer, C. Benkeser, S. Belfanti, and Q. Huang, "Design and implementation of a parallel turbo-decoder ASIC for 3GPP-LTE," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 8–17, Jan. 2011.

[20] M. A. Bickerstaff *et al.*, "A unified turbo/Viterbi channel decoder for 3GPP mobile wireless in 0.18-μm CMOS," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1555–1564, Nov. 2002.

[21] J. Vogt and A. Finger, "Improving the max-log-MAP turbo decoder," *Electron. Lett.*, vol. 36, no. 23, pp. 1937–1939, Nov. 2000.

[22] J.-H. Kim and I.-C. Park, "A unified parallel radix-4 turbo decoder for mobile WiMAX and 3GPP-LTE," in *Proc. IEEE Custom Integr. Circuits Conf.*, San Jose, CA, USA, Sep. 2009, pp. 487–490.

[23] S. S. Pietrobon, "Efficient implementation of continuous MAP decoders and a synchronisation technique for turbo decoders," in *Proc. Int. Symp. Inf. Theory Appl.*, 1996, pp. 586–589.

[24] *Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and Channel Coding, Version 12.0.0*, document ETSI TS 136 212 LTE, 2013.

[25] P. Elias, "Coding for noisy channels," *IRE Conv. Rec.*, vol. 3, no. 4, pp. 37–46, 1955.

[26] R. Garello, F. Chiaraluce, P. Pierleoni, M. Scaloni, and S. Benedetto, "On error floor and free distance of turbo codes," in *Proc. IEEE Int. Conf. Commun.*, vol. 1. Jun. 2001, pp. 45–49.

[27] B. Sklar, "Rayleigh fading channels in mobile digital communication systems. I. Characterization," *IEEE Commun. Mag.*, vol. 35, no. 9, pp. 136–146, Sep. 1997.

[28] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.

[29] N. L. Johnson, A. W. Kemp, and S. Kotz, *Univariate Discrete Distributions*. New York, NY, USA: Wiley, 2005.

[30] ITU-T. (2015). *Series H: Audiovisual and Multimedia Systems, Infrastructure of Audiovisual Services—Coding of Moving Video, High Efficiency Video Coding*. [Online]. Available: http://www.itu.int/rec/T-REC-H.265-201504-I

[31] L. L. Hanzo, R. G. Maunder, J. Wang, and L.-L. Yang, *Near-Capacity Variable-Length Coding*. Chichester, U.K.: Wiley, Oct. 2010.

[32] S. Hong, J. Yi, and W. E. Stark, "VLSI design and implementation of low-complexity adaptive turbo-code encoder and decoder for wireless mobile communication applications," in *Proc. IEEE Workshop Signal Process. Syst.*, Oct. 1998, pp. 233–242.

[33] C. Benkeser, A. Burg, T. Cupaiuolo, and Q. Huang, "Design and optimization of an HSDPA turbo decoder ASIC," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 98–106, Jan. 2009.

[34] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, "A 24Mb/s radix-4 logMAP turbo decoder for 3GPP-HSDPA mobile wireless," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 1. Feb. 2003, pp. 150–151.

[35] D. W. Kim, T. W. Kwon, J. R. Choi, and J. J. Kong, "A modified two-step SOVA-based turbo decoder with a fixed scaling factor," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Geneva, Switzerland, May 2000, pp. 37–40.

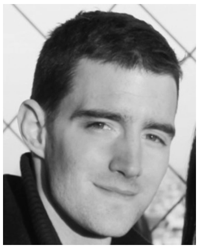
[36] M. van Dijk, A. J. E. M. Janssen, and A. G. C. Koppelaar, "Correcting systematic mismatches in computed log-likelihood ratios," *Eur. Trans. Telecommun.*, vol. 14, no. 3, pp. 227–244, May/Jun. 2003.

[37] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A survey of FPGA-based LDPC decoders," *IEEE Commun. Surveys Tut.*, vol. 18, no. 2, pp. 1098–1122, 2nd Quart., 2016.

[38] Altera. (2012). *Chapter 1. Overview for the Stratix IV Device Family*. [Online]. Available: https://www.altera.com/products/fpga/stratix-series/stratix-iv/support.html

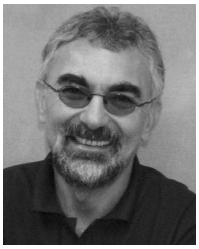
[39] L. F. Gonzalez-Perez, L. C. Yllescas-Calderon, and R. Parra-Michel, "Parallel and configurable turbo decoder implementation for 3GPP-LTE," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs (ReConFig)*, Cancún, Mexico, Dec. 2013, pp. 1–6.

MATTHEW F. BREJZA received the B.Eng. degree (Hons.) in electronic engineering from the University of Southampton, U.K., in 2012, where he is currently pursuing the Ph.D. degree with the Communications Research Group, School of Electronics and Computer Science. His research interests include flexible hardware implementation, source and channel coding, and their applications in low power data communications.



ROBERT G. MAUNDER received the B.Eng. degree (Hons.) in electronic engineering and the Ph.D. degree in wireless communications from the School of Electronics and Computer Science, University of Southampton, U.K., in 2003 and 2007, respectively. He was a Lecturer in 2007 and an Associated Professor in 2013. His research interests include joint source/channel coding, iterative decoding, irregular coding, and modulation techniques.

BASHIR M. AL-HASHIMI is currently an ARM Professor of Computer Engineering and the Dean of the Faculty of Physical Sciences and Engineering, University of Southampton. He has authored over 300 technical papers, authored or co-authored five books and has graduated 31 Ph.D. students. In 2009, he was a elected fellow of the IEEE for significant contributions to the design and test of low-power circuits and systems. He was a recipient of the Royal Society Wolfson Research Merit Award (2014-2019).



LAJOS HANZO (F'04) received the D.Sc. degree in electronics in 1976 and the Ph.D. degree in 1983. During his 40-year career in telecommunications he has held various research and academic posts in Hungary, Germany, and U.K. Since 1986, he has been with the School of Electronics and Computer Science, University of Southampton, U.K. He has successfully supervised 110 Ph.D. students. In 2016 he was admitted to the Hungarian Academy of Science. He is currently directing a 60-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, U.K., the European IST Programme and the Mobile Virtual Centre of Excellence, U.K., the European Research Council's Advanced Fellow Grant, and the Royal Society's Wolfson Research Merit Award. His research is funded by the European Research Council's Senior Research Fellow Grant. He is an enthusiastic supporter of industrial and academic liaison and he offers a range of industrial courses. He has co-authored 20 John Wiley/IEEE Press books on mobile radio communications totaling in excess of 10 000 pages, published over 1600 research entries at the IEEE Xplore. He was FREng, FIET, and a fellow of the EURASIP. In 2009, he received the honorary doctorate by the Technical University of Budapest and in 2015 by the University of Edinburgh. He served as the TPC Chair and the General Chair of the IEEE conferences, presented keynote lectures and has been received a number of distinctions. He is also a Governor of the IEEE VTS. From 2008 to 2012, he was the Editor-in-Chief of the IEEE Press and a Chaired Professor at Tsinghua University, Beijing. He is the Chair in telecommunications with the University of Southampton.