Received December 3, 2015, accepted January 5, 2016. Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2016.2515719

VLSI Implementation of Fully Parallel LTE Turbo Decoders

AN LI[1], LUPING XIANG[1], TAIHAI CHEN[1], ROBERT G. MAUNDER[1], BASHIR M. AL-HASHIMI[2], (Fellow, IEEE), AND LAJOS HANZO[1], (Fellow, IEEE)

[1]Southampton Wireless Group, Department of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, U.K.
[2]Faculty of Physical Sciences and Engineering, University of Southampton, Southampton SO17 1BJ, U.K.

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

This work was supported in part by the Engineering and Physical Sciences Research Council, Swindon, U.K., under Grant EP/J015520/1 and Grant EP/L010550/1, in part by Technology Strategy Board, Swindon, U.K., under Grant TS/L009390/1, and in part by the European Research Council Advance Fellow Grant within the Beam-Me-Up Program. The research data for this paper can be found at http://dx.doi.org/10.5258/SOTON/385521.

ABSTRACT Turbo codes facilitate near-capacity transmission throughputs by achieving a reliable iterative forward error correction. However, owing to the serial data dependence imposed by the logarithmic Bahl–Cocke–Jelinek–Raviv algorithm, the limited processing throughputs of the conventional turbo decoder implementations impose a severe bottleneck upon the overall throughputs of real-time communication schemes. Motivated by this, we recently proposed a floating-point fully parallel turbo decoder (FPTD) algorithm, which eliminates the serial data dependence, allowing parallel processing and hence significantly reducing the number of clock cycles required. In this paper, we conceive a technique for reducing the critical datapath of the FPTD, and we propose a novel fixed-point version as well as its very large scale integration (VLSI) implementation. We also propose a novel technique, which allows the FPTD to also decode shorter frames employing compatible interleaver patterns. We strike beneficial tradeoffs amongst the latency, core area, and energy consumption by investigating the minimum bit widths and techniques for message log-likelihood ratio scaling and state metric normalization. Accordingly, the design flow and design tradeoffs considered in this paper are also applicable to other fixed-point implementations of error correction decoders. We demonstrate that upon using Taiwan Semiconductor Manufacturing Company (TSMC) 65-nm low-power technology for decoding the longest long-term evolution frames (6144 b) received over an additive white Gaussian noise channel having $E_b/N_0 = 1$ dB, the proposed fixed-point FPTD VLSI achieves a processing throughput of 21.9 Gb/s and a processing latency of 0.28 μs. These results are 17.1 times superior to those of the state-of-the-art benchmarker. Furthermore, the proposed fixed-point FPTD VLSI achieves an energy consumption of 2.69 μJ/frame and a normalized core area of 5 mm^2/Gb/s, which are 34% and 23% lower than those of the benchmarker, respectively.

INDEX TERMS Fully-parallel turbo decoder, VLSI design, LTE turbo code.

NOMENCLATURE

3GPP	3rd Generation Partnership Project
AWGN	Additive White Gaussian Noise
BCJR	Bahl-Cocke-Jelinek-Raviv
BER	Bit Error Rate
CMOS	Complementary Metal-Oxide Semiconductor
EPF	Energy Per Frame
FFT	Fast Fourier Transform
GPRS	General Packet Radio Service
LLR	Log-Likelihood Ratio
Log-BCJR	Logarithmic Bahl-Cocke-Jelinek-Raviv
LP	Low Power
LTE	Long-Term Evolution
LTE-A	Long-Term Evolution Advanced
OFDM	Orthogonal Frequency Division Multiplexing
UMTS	Universal Mobile Telecommunications System
VLSI	Very-Large-Scale Integration
WiMAX	Worldwide Interoperability for Microwave Access
TSMC	Taiwan Semiconductor Manufacturing Company

I. INTRODUCTION

Channel coding plays an important role in wireless communications, facilitating the correction of transmission errors imposed by hostile channels. In particular, turbo codes [1]–[4] are capable of facilitating near-capacity *transmission* throughputs, leading to widespread employment by state-of-the-art mobile telephony standards, such as WiMAX [5] and LTE [6]. However, the *processing* throughputs of the turbo decoder can impose a bottleneck upon the *overall* throughput in near-real-time interactive communication schemes. More specifically, the target *transmission* throughputs of mobile telephony standards have increased dramatically over the past two decades, from multi-kbps to multi-Gbps, as shown in Figure 1. In order to fulfill these *transmission* throughput targets, many high-throughput implementations of the turbo decoder have been proposed [8], [10], [11], [13], [14], [16], [18], [19]. However, none of those meet the throughput requirement of the next-generation standards. For instance, the target for the under-development Fifth-Generation (5G) [20] wireless communication standards is a fiber-like ultra-high throughput on the order of 10 Gbps per user, in addition to ultra-low latencies. However, the state-of-the-art VLSI implementations of the turbo decoder only achieve processing throughputs of 1.28 Gbps [16] or 2.15 Gbps [18], when decoding the longest frames ($N = 6144$ bits) supported by LTE, corresponding to processing latencies of $N/\text{throughput} = 4.8\ \mu$s and 2.86 μs, respectively. This may be attributed to the state-of-the-art turbo decoder VLSIs' reliance on the iterative operation of two parallel concatenated component decoders, in which the Logarithmic Bahl-Cock-Jelinek-Raviv (Log-BCJR) algorithm [21], [22] is employed. More specifically, the strict data dependencies of the classic Log-BCJR algorithm require highly serial processing, typically necessitating 64 to 192 clock cycles per iteration [18] and six iterations per frame. In order to facilitate the real-time processing of data having a *transmission* throughput of at least 10 Gbps, it would be necessary to operate several instances of these benchmarkers in parallel. However, this target *processing* throughput would only be achieved, when several frames were simultaneously available for decoding. At all other times, the unused instances would represent wasted core area and static energy consumption. Furthermore, the processing latency of these benchmarkers is not improved by operating them in parallel hence causing a bottleneck where the turbo decoding latency becomes several times higher than those of all other transmitter and receiver components, which natively support a processing throughput of 10 Gbps.

Motivated by natively achieving turbo decoding processing throughputs on the order of 10 Gbps and ultra-low processing latencies, we previously proposed a novel floating-point Fully-Parallel Turbo Decoder (FPTD) algorithm. Unlike turbo decoders based on the Log-BCJR algorithm, our FPTD algorithm does not have data dependencies within each half of each turbo decoding iteration [23]. This facilitates fully-parallel processing during each half-iteration, using only a single clock cycle, although the authors of [23] showed that the FPTD typically requires seven times as many iterations in order to achieve the same error correction capability as the state-of-the-art turbo decoding algorithm, as well as predicting that a VLSI implementation of the FPTD would have a lower clock frequency. Overall, the results of our previous algorithmic work suggest that the FPTD may achieve an up to 6.86-fold improved throughput and latency compared to those of the state-of-the-art turbo decoding algorithm [23], at the cost of an 2.9-fold increase of the computational complexity and a predicted 29.3-fold increase of the hardware resource requirement.

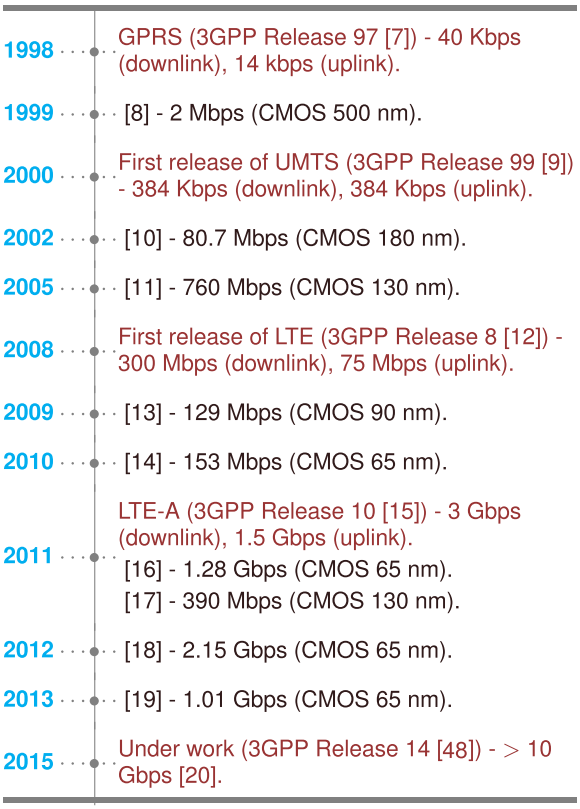


FIGURE 1. Selected throughput requirements of different mobile telephony standards, compared with those achieved by the existing turbo decoder implementations.

In this paper, we propose a VLSI implementation of the proposed FPTD, which is optimized for the LTE turbo code. In order to present this work clearly, we begin in Section I-A by providing an overview of the practical trade-offs that must be considered, when designing the algorithm and hardware implementation of turbo decoders. Following this, we highlight the contributions of this work and present the structure of this paper in Section I-B.

A. DESIGN TRADE-OFFS

Figure 2 provides an overview of the design aspects and their relationship with the design trade-offs, which must be

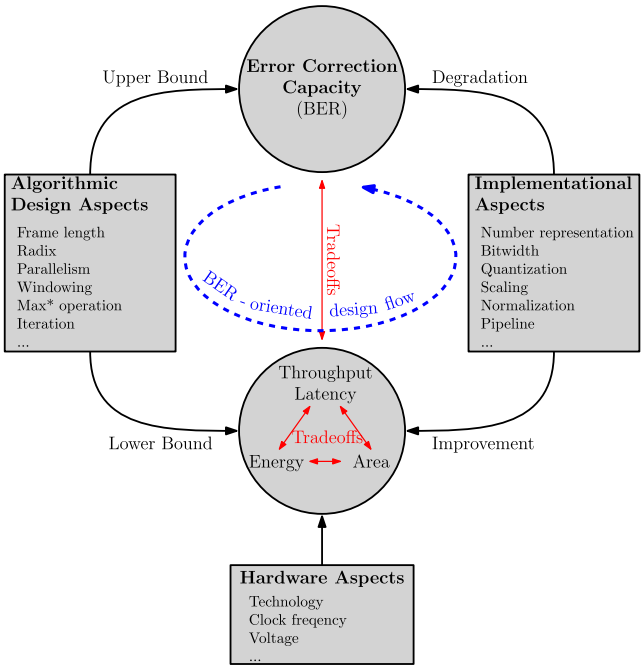


FIGURE 2. An overview of design aspects and their relationship with design trade-offs.

considered when designing and implementing a turbo decoder or any other error correction decoder. In particular, we have to strike a trade-off between the decoder's error correction capability and its hardware characteristics, in terms of processing throughput, processing latency, energy consumption and VLSI area. Additionally, these hardware characteristics also have trade-offs with each other. These trade-offs are related to three groups of the design aspects, namely the algorithmic design aspects, the implementational aspects and the hardware aspects, as shown in Figure 2. The algorithmic design aspects are independent to a degree of the hardware, hence they can be investigated purely at the algorithmic level. The corresponding error correction capability may be obtained as an upper bound on the capability that can be achieved in practice. Meanwhile, the algorithmic design provides estimates of the lower bounds on the hardware characteristics, as we will demonstrate in this treatise. By contrast, the implementational aspects are hardware-dependent considerations, which may result in a degradation of the error correction capability on one hand, but an improvement in the hardware characteristics on the other hand. The hardware aspects may be deemed to be predominantly hardware-related, hence they are independent of the algorithm design. Bearing in mind the above-mentioned trade-offs, a possible error-correction-capability-oriented design flow may be formulated, as shown by the dotted blue arrow in Figure 2. This design flow hinges on the initial investigating the algorithmic aspects, in order to optimize one or more of the hardware characteristics, namely the throughput, latency, energy consumption or VLSI area, subject to the constraint of meeting the desired error correction capability within a suitable margin. Following this, the implementation aspects are considered, in order to strike a balance between the degradation of the error correction capability and the impact on the hardware performance, ensuring that the final design still maintains the desired error correction capability. Finally, the specific detailed hardware aspects may be considered, to further balance the trade-offs amongst the throughput, latency, energy consumption and core area. This process may be iterated several times, until a final design meeting all requirements is obtained.

Although the above-mentioned design trade-offs have been discussed in some previous papers, they typically focus on a particular aspect of the design flow, considering either only the algorithmic level or the hardware level, failing to achieve a top-level holistic design flow of Figure 2. In particular, the authors of [3], [24], and [25] investigated how the error correction capability of the Log-BCJR turbo decoder is impacted by using various algorithm design techniques, such as windowing, parallelism and the careful configuration of the number of decoding iterations preformed. However, this was considered only at the algorithmic level, without giving cognizance to the implementational aspects. By contrast, the authors of [26]–[28] discussed how the implementational aspects of fixed-point bitwidth and the state metric normalization technique affect the error correction capability at the algorithmic level. Meanwhile, the authors of [8], [16], [18], [19], and [29]–[33] additionally characterized the hardware performance using post-layout simulations or measurements, when employing a particular combination of the above-mentioned techniques. Although these papers provided a practical clock frequency and supply voltage for the corresponding implementations, they did not consider a broader range of hardware aspects, such as the effect of varying the clock frequency or supply voltage.

B. CONTRIBUTIONS AND PAPER STRUCTURE

Against this background, we conceive an optimized FPTD algorithm for LTE turbo decoding, which reduces the critical datapath length of the FPTD proposed in [23]. On the basis of this, we propose a novel low-complexity and energy-efficient fixed-point version of the FPTD, as well as its VLSI implementation. While our previous work of [23] considered only the algorithmic design aspects of Figure 2, this paper adopts a holistic design flow, considering all the aspects and trade-offs of Figure 2. We investigate the minimum number of iterations and the minimum bit widths required by the proposed fixed-point FPTD VLSI, in order to maintain the same error correction capability as the state-of-the-art turbo decoder VLSI implementations, which operate on the basis of the Log-BCJR algorithm. Following this, a range of techniques that have been previously used for improving the operation of Log-BCJR turbo decoder implementations are shown to be eminently applicable to the proposed fixed-point FPTD VLSI as well. These state metric normalization and message LLR scaling techniques are shown to avoid fixed-point number overflows and to allow reduced bit widths to be used. Furthermore, we propose a novel bypass technique that

allows a hard-wried 6144-bit interleaver to be exploited also for decoding shorter frames having a particular set of compatible interleaver patterns. Finally, the trade-offs between the error correction capability and the processing throughput, processing latency, processing energy and the core area of the FPTD VLSI are characterized. The main experimental results of this work are listed as follows.

1) When using the TSMC 65 nm Low Power (LP) technology, the proposed fixed-point FPTD VLSI achieves a *processing* throughput of 21.9 Gbps, as well as a *processing* latency of 0.28 μs, when decoding the longest LTE frames (6144-bit) received over an AWGN channel having $E_b/N_0 = 1$ dB. These results are 17.1 times superior to the state-of-the-art Log-BCJR benchmarker of [16], which also employs TSMC 65 nm technology.
2) The proposed fixed-point FPTD VLSI imposes an energy consumption of 2.69 μJ per frame and has a normalized core area of 5 mm^2/Gbps, which are 34% and 23% lower than those of the state-of-the-art Log-BCJR benchmarker of [16], respectively. These results also significantly outperform the predictions made in our previous algorithmic work of [23].
3) The processing throughput and latency of the proposed fixed-point FPTD VLSI are 10.2 times better than those of a second state-of-the-art TSMC 65 nm benchmarker [18]. However, the normalized core area of the proposed VLSI is 42% larger than that of the benchmarker of [18], as we shall address in our future work. Note that energy consumption results are not provided for the benchmarker of [18], although this is likely to be even higher than that of [16], since the VLSI of [18] employs both a higher voltage and a higher clock frequency, while occupying a similar core area as in [16].

In analogy with the error-correction-capability-oriented design flow of Figure 2, the rest of the paper is organized as shown in Figure 3. In Section II, we briefly summarize our previously proposed FPTD algorithm of [23], detailing our novel optimizations for the LTE turbo code standard and providing the predicted lower bounds of the hardware characteristics of the VLSI FPTD implementation. Section III details the proposed fixed-point FPTD architecture, including its number representation, message LLR scaling, state metric normalization and bypass unit, as well as the implications for the FPTD's error correction capability. In Section IV, we compare the hardware characteristics of our proposed fixed-point FPTD VLSI, to those of the state-of-the-art turbo decoder VLSI implementations, as well as to the predictions made in Section II, in terms of processing throughput, processing latency, energy consumption and core area. Finally, we offer our conclusions in Section V.

II. FPTD ALGORITHM FOR LTE

In this section, we summarize our previously proposed FPTD algorithm of [23], for the case where it is adopted for the LTE turbo decoder. In Section II-A, we discuss the FPTD schematic of Figure 4. The algorithmic block and the termination unit of Figure 4 are described in Section II-B, together with a technique for reducing the critical datapath length. In Section II-C, we compare the resultant improved version of the FPTD algorithm to the original version of [23], as well as to the state-of-the-art Log-BCJR algorithms of [16] and [18] in terms of BER performance and computational complexity. Note that the bypass mechanism of Figure 4 will be described together with our other novel contributions in Section III.

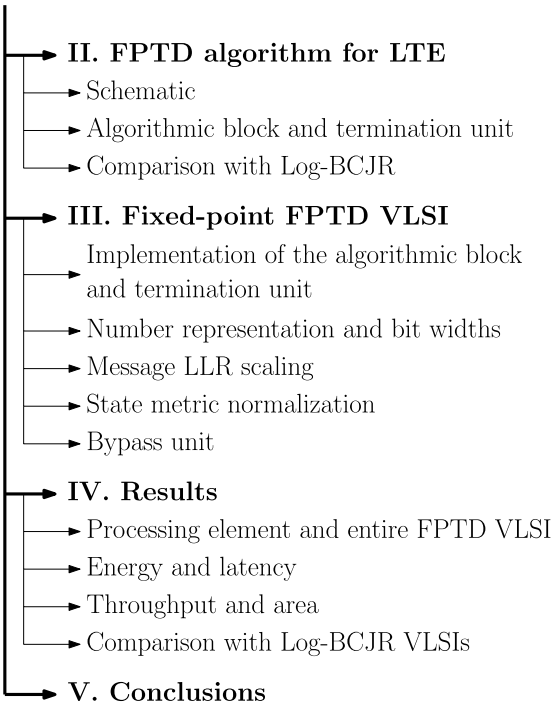


FIGURE 3. Paper structure.

A. SCHEMATIC

Figure 4 provides the schematic of our previously proposed FPTD algorithm of [23], but has been adapted to include the novel contributions of this paper. When decoding N-bit message frames, the FPTD algorithm comprises two rows of N identical algorithmic blocks, some of which are lightly-shaded in Figure 4, while others are darkly-shaded. The upper row is analogous to the upper decoder of the conventional Log-BCJR turbo decoder, while the lower row corresponds to the lower decoder, which are connected by an LTE interleaver, as shown in Figure 4. A termination unit is appended to the tail of each row, in order to comply with the LTE termination mechanism. As in the Log-BCJR algorithm, the FPTD algorithm operates on the basis of Logarithmic Likelihood Ratios (LLRs) [34], where each LLR of

$$\bar{b} = \ln \frac{\Pr(b=1)}{\Pr(b=0)} \tag{1}$$

conveys soft information pertaining to the corresponding bit b within the turbo encoder. Note that throughout the rest of this

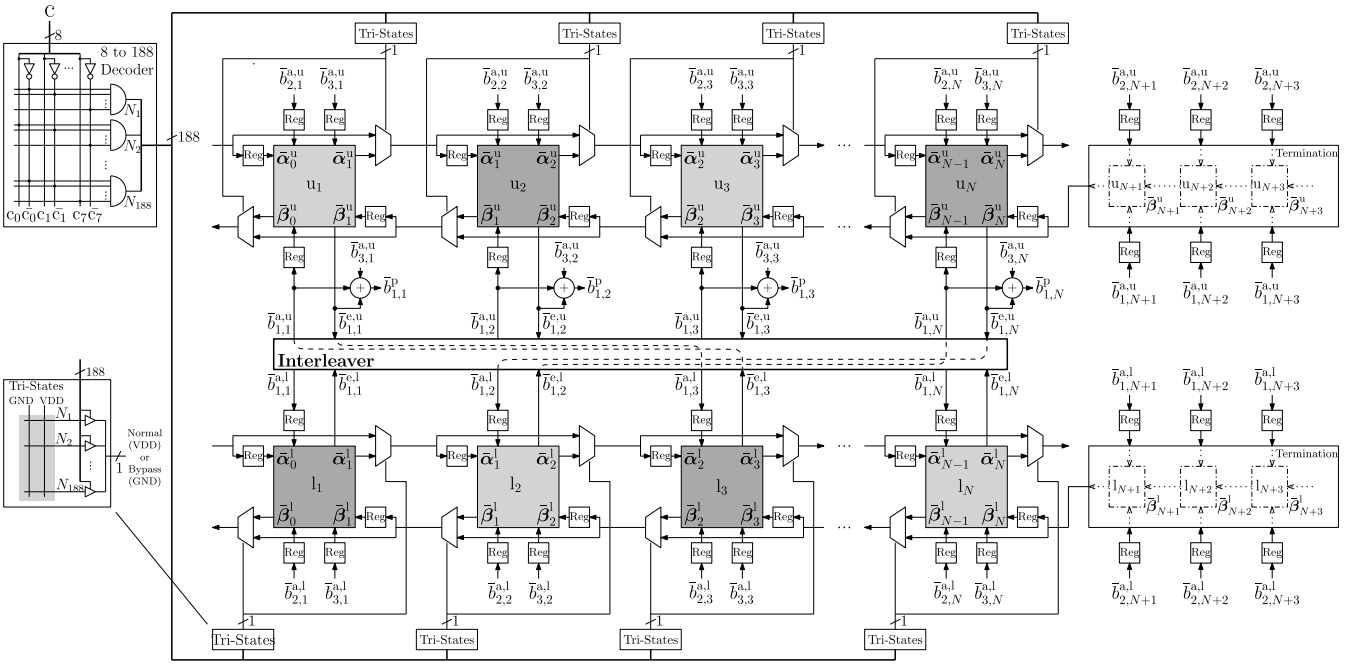


FIGURE 4. Schematic of the FPTD for LTE, including algorithmic blocks, termination units and bypass mechanism.

paper, the superscripts 'u' and 'l' seen in the notation of Figure 4 are used only when necessary to explicitly distinguish between the upper and lower components of the turbo code, but they are omitted in the discussions that apply equally to both. When decoding the frames comprising N bits, the upper and lower decoders each accept a set of $(N+3)$ *a priori* parity LLRs $[\bar{b}^{\mathrm{a}}_{2,k}]^{N+3}_{k=1}$, a set of N *a priori* systematic LLRs $[\bar{b}^{\mathrm{a}}_{3,k}]^{N}_{k=1}$ and a set of three *a priori* termination message LLRs $[\bar{b}^{\mathrm{a}}_{1,k}]^{N+3}_{k=N+1}$, where N will adopt one of 188 values in the range of [40, 6144] in the LTE turbo code. As shown in Figure 4, these *a priori* LLRs are provided by the demodulator and are stored in the corresponding registers throughout the decoding processing of the corresponding frame. Note that the set of lower systematic LLRs $[\bar{b}^{\mathrm{a,l}}_{3,k}]^{N}_{k=1}$ can be obtained by rearranging the order of LLRs in the upper systematic set $[\bar{b}^{\mathrm{a,u}}_{3,k}]^{N}_{k=1}$ using the interleaver π, where $\bar{b}^{\mathrm{a,l}}_{3,k} = \bar{b}^{\mathrm{a,u}}_{3,\pi(k)}$. Therefore, the FPTD requires only five sets of LLRs from the demodulator, namely $[\bar{b}^{\mathrm{a,u}}_{2,k}]^{N+3}_{k=1}$, $[\bar{b}^{\mathrm{a,u}}_{3,k}]^{N}_{k=1}$, $[\bar{b}^{\mathrm{a,u}}_{1,k}]^{N+3}_{k=N+1}$, $[\bar{b}^{\mathrm{a,l}}_{2,k}]^{N+3}_{k=1}$ and $[\bar{b}^{\mathrm{a,l}}_{1,k}]^{N+3}_{k=N+1}$, comprising a total of $(3N+12)$ LLRs, in accordance with the LTE standard.

As in Log-BCJR turbo decoders, the operation of the FPTD alternates between the processing of two half-iterations. However, in Log-BCJR turbo decoders, the first half-iteration corresponds to the operation of the upper decoder, while the second half-iteration corresponds to the lower decoder, where each half-iteration requires 32 to 96 clock cycles in the state-of-the-art designs of [18]. Although the shuffled iterative decoding scheme of [35] allows the two half-iterations to be operated concurrently, it still requires 32 to 96 clock cycles per iteration. By contrast, as shown in Figure 5, the first half-iteration of the FPTD algorithm corresponds to the simultaneous operation of the lightly-shaded algorithmic blocks shown in Figure 4 within a single clock cycle, namely the odd-indexed algorithmic blocks in the upper row and the even-indexed algorithmic blocks in the lower row. As shown in Figure 6, the second half-iteration corresponds to the simultaneous operation of the remaining algorithmic blocks within a single clock cycle, which are darkly-shaded in Figure 4. Accordingly, each iteration of our proposed FPTD algorithm requires only two clock cycles. Note that the schematics of Figures 5 and 6 are simplified relative to the detailed schematic of Figure 4, showing only the algorithmic blocks with their corresponding input and output datapath, for the sake of clarity. This odd-even operation is motivated by the odd-even interleaver philosophy [36] of the LTE turbo code. More specifically, the particular design of the LTE interleaver ensures that the odd-indexed algorithmic blocks in the upper row of Figure 4 are only connected to the odd-indexed algorithmic blocks in the lower row. Similarly, the even-indexed algorithmic blocks in the upper row are only connected to their even-indexed counterparts in the lower row. In other words, none of the lightly-shaded algorithmic blocks shown in Figure 5 are directly connected, either within a row or between the rows via the interleaver. Similarly, none of the dark-shaded algorithmic blocks shown in Figure 6 are directly connected. Owing to this, no directly connected algorithmic blocks are operated simultaneously in the FPTD algorithm, hence preventing wasted computations [23]. Note that this

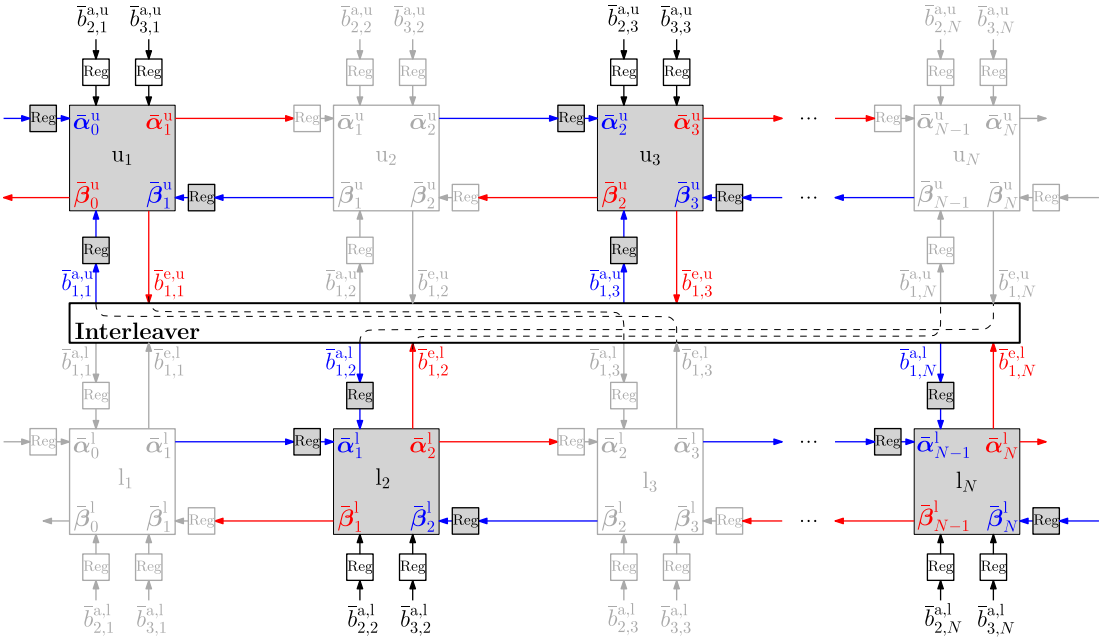


FIGURE 5. Lightly-shaded algorithmic blocks are operated concurrently in every odd clock cycle, corresponding to every first half-iteration. The input datapaths and output datapaths are colored in blue and red, respectively.

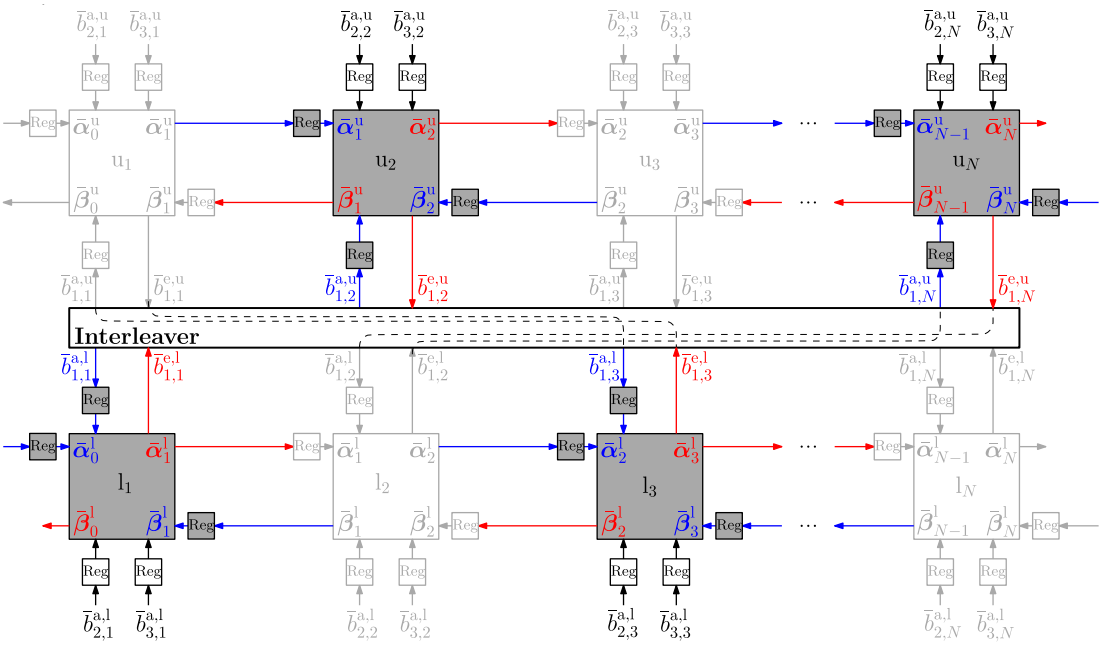


FIGURE 6. Darkly-shaded algorithmic blocks are operated concurrently in every even clock cycle, corresponding to every second half-iteration. The input datapaths and output datapaths are colored in blue and red, respectively.

proposed odd-even operation is applicable to any turbo code that employs an odd-even interleaver, such as that of the WiMAX turbo code. In particular the popular Almost Regular Permutation (ARP) interleaver and Quadratic Polynomial Permutation (QPP) interleaver designs both retain the odd-even feature [36].

During the decoding process, the extrinsic message LLRs $[\bar{b}^{\mathrm{e}}_{1,k}]_{k=1}^{N}$ are iteratively exchanged between the upper and lower decoders through the interleaver, in order to obtain the *a priori* message LLRs $[\bar{b}^{\mathrm{a}}_{1,k}]_{k=1}^{N}$, where $\bar{b}^{\mathrm{a,l}}_{1,k} = \bar{b}^{\mathrm{e,u}}_{1,\pi(k)}$ and $\bar{b}^{\mathrm{a,u}}_{1,\pi(k)} = \bar{b}^{\mathrm{e,l}}_{1,k}$. In addition to the *a priori* LLRs $\bar{b}^{\mathrm{a}}_{1,k}$,

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

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

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

$$\begin{aligned} \bar{b}^{\mathrm{e}}_{1,k} = & \left[\max^{*}_{\{(S_{k-1},S_k)|b_1(S_{k-1},S_k)=1\}} \left[b_2(S_{k-1}, S_k) \cdot \bar{b}^{\mathrm{a}}_{2,k} + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k) \right] \right] \\ & - \left[\max^{*}_{\{(S_{k-1},S_k)|b_1(S_{k-1},S_k)=0\}} \left[b_2(S_{k-1}, S_k) \cdot \bar{b}^{\mathrm{a}}_{2,k} + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k) \right] \right] \end{aligned} \tag{5}$$

$\bar{b}^{\mathrm{a}}_{2,k}$ and $\bar{b}^{\mathrm{a}}_{3,k}$, the $k^{\mathrm{th}} \in [1, N]$ algorithmic block in each decoder accepts a set of M forward state metrics $\bar{\boldsymbol{\alpha}}_{k-1} = [\bar{\alpha}_{k-1}(S_{k-1})]^{M-1}_{S_{k-1}=0}$ and a set of M backward state metrics $\bar{\boldsymbol{\beta}}_k = [\bar{\beta}_k(S_k)]^{M-1}_{S_k=0}$, where the LTE turbo code employs $M = 8$ states. For algorithmic blocks having an index of $k \in [2, N]$, $\bar{\alpha}_{k-1}$ is generated in the previous clock cycle by the preceding $(k-1)^{\mathrm{th}}$ algorithmic block in the same row. Likewise, for algorithmic blocks having an index of $k \in [1, N-1]$, $\bar{\beta}_k$ is generated in the previous clock cycle by the following $(k+1)^{\mathrm{st}}$ algorithmic block in the same row. As shown in Figure 4, registers are required for storing $[\bar{b}^{\mathrm{a}}_{1,k}]^{N}_{k=1}$, $[\bar{\boldsymbol{\alpha}}_{k-1}]^{N}_{k=2}$ and $[\bar{\boldsymbol{\beta}}_k]^{N-1}_{k=1}$ between the consecutive clock cycles, since they are generated by connected algorithmic blocks in the preceding clock cycle before they are used.

Since the *a priori* message LLRs $[\bar{b}^{\mathrm{a}}_{1,k}]^{N}_{k=1}$ are unavailable in the initial first half-iteration, they are initialized as $\bar{b}^{\mathrm{a}}_{1,k} = 0$, for algorithmic blocks having indices of $k \in [1, N]$. Similarly, the forward and backward state metrics gleaned from the neighboring algorithmic blocks are unavailable, hence these are also initialized as $\bar{\boldsymbol{\alpha}}_{k-1} = [0, 0, 0, \ldots, 0]$ for the algorithmic blocks having indices of $k \in [2, N]$ and as $\bar{\boldsymbol{\beta}}_k = [0, 0, 0, \ldots, 0]$ for the algorithmic blocks of indices $k \in [1, N-1]$. However, for the $k = 1^{\mathrm{st}}$ algorithmic block, we employ the forward state metrics $\bar{\boldsymbol{\alpha}}_0 = [0, -\infty, -\infty, \ldots, -\infty]$ in all decoding iterations, since the LTE trellis is guaranteed to start from an initial state of $S_0 = 0$. Note that $-\infty$ can be replaced by a negative constant having a suitably high magnitude, when a fixed-point number representation is employed. For the $k = N^{\mathrm{th}}$ algorithmic block, the backward state metrics $\bar{\boldsymbol{\beta}}_N$ are obtained using a termination unit, which is detailed in Section II-B. Following the completion of each iteration, a set of N *a posteriori* LLRs $[\bar{b}^{\mathrm{p}}_{1,k}]^{N}_{k=1}$ can be obtained as $\bar{b}^{\mathrm{p}}_{1,k} = \bar{b}^{\mathrm{e,u}}_{1,k} + \bar{b}^{\mathrm{a,u}}_{1,k} + \bar{b}^{\mathrm{a,u}}_{3,k}$, while the hard decision value for each bit may be obtained according to the binary test $\bar{b}^{\mathrm{p}}_{1,k} > 0$.

B. ALGORITHMIC BLOCK AND TERMINATION UNIT

Within each clock cycle during which an algorithmic block of Figure 4 is activated, it accepts inputs and generates outputs according to (2), (3), (4) and (5), as shown at the top of this page. As it will be detailed in Section III, (2)-(5) have been refined relative to the corresponding equations in [23], in order to improve the critical datapath length, when implementing the LTE turbo decoder. Here, (2) obtains a metric $\bar{\gamma}_k(S_{k-1}, S_k)$ for each transition between a pairing of states S_{k-1} and S_k, as shown in the LTE state transition diagram of Figure 7. This transition implies the binary value of $b_j(S_{k-1}, S_k)$ for the corresponding message, parity or systematic bit in the encoder, where $j \in [1, L]$ and $L = 3$ in the LTE turbo code. Note that the systematic bits are defined as having values that are identical to the corresponding message bits, giving $b_3(S_{k-1}, S_k) \equiv b_1(S_{k-1}, S_k)$. Following this, (3) and (4) may be employed to obtain the vectors of state metrics $\bar{\boldsymbol{\alpha}}_k$ and $\bar{\boldsymbol{\beta}}_{k-1}$, respectively. Here, $c(S_{k-1}, S_k)$ adopts a binary value of 1, if there is a transition between the states S_{k-1} and S_k in the state transition diagram of Figure 7. The Jacobian logarithm [22], [37] is defined as

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

which may be approximated as

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

in order to reduce the computational complexity, in analogy with the Max-Log BCJR [22], [37]. Finally, (5) may be

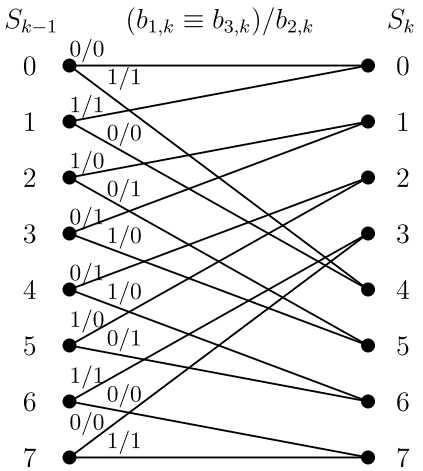


FIGURE 7. State transition diagram of the LTE turbo code.

employed for obtaining the extrinsic LLR $\bar{b}^{\mathrm{e}}_{1,k}$, where the associative property of the max* operator of (6) may be exploited to make it capable of simultaneously considering more than two operands.

Note that each row of algorithmic blocks shown in Figure 4 is appended with a termination unit, comprising three algorithmic blocks having indices of $(N+1)$, $(N+2)$ and $(N+3)$. These termination blocks employ only (2) in conjunction with $L = 2$ and (4), operating in a backward recursion fashion to successively calculate $\bar{\boldsymbol{\beta}}_{N+2}$, $\bar{\boldsymbol{\beta}}_{N+1}$ and $\bar{\boldsymbol{\beta}}_{N}$. Here, we employ $\bar{\boldsymbol{\beta}}_{N+3} = [0, -\infty, -\infty, \ldots, -\infty]$, since the LTE termination technique guarantees $S_{N+3} = 0$. As described in Section II-A, here $-\infty$ may be replaced by a negative constant having a suitably high magnitude, when a fixed-point number representation is employed. Note that the termination units can be operated before and independently of the iterative decoding process, since the required *a priori* LLRs $[\bar{b}^{\mathrm{a}}_{1,k}]^{N+3}_{k=N+1}$ and $[\bar{b}^{\mathrm{a}}_{2,k}]^{N+3}_{k=N+1}$ are provided only by the demodulator, with no data dependencies on the other N algorithmic blocks in the row. Owing to this, the resultant $\bar{\boldsymbol{\beta}}_{N}$ can be used throughout the iterative decoding process, with no need to operate the termination unit again, as described in Section II-A.

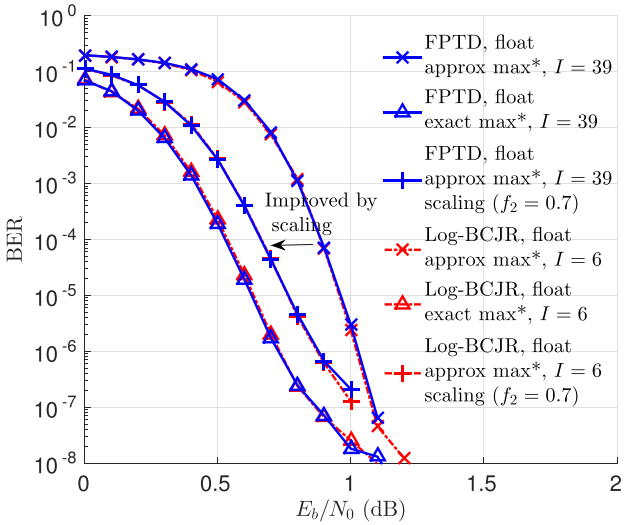


FIGURE 8. BER comparison between the floating-point FPTD and the floating-point Log-BCJR turbo decoder, considering exact max* operation of (6), approximate max* operation of (7) and approximate max* with message LLR scaling ($f_2 = 0.75$). The Log-BCJR turbo decoder employs the non-sliding windowing mechanism with a window size of 96 [16], [18]. The BER was simulated for the case of transmitting the longest LTE $N = 6144$-bit frames over an AWGN channel.

C. COMPARISON WITH LOG-BCJR

This section summarizes the key differences between the improved FPTD algorithm of (2)-(5), the FPTD algorithm of [23] and the state-of-the-art Log-BCJR turbo decoder algorithm, according to [23]. As shown in Table 1, the comparison considers the number of clock cycles per decoding iteration (T), the clock cycle duration (D), the number of decoding iterations required (I), the complexity per decoding iteration (C), the overall throughput ($\frac{1}{T \cdot D \cdot I}$), the overall latency ($T \cdot D \cdot I$), the overall complexity ($C \cdot I$) and the overall hardware resource requirement. Note that a more detailed comparison is offered in [23].

As discussed in Section II-A, the FPTD algorithm requires only two clock cycles per iteration ($T_{\mathrm{FPTD}} = 2$), since each half-iteration requires only a single clock cycle. By contrast, each iteration of the state-of-the-art Log-BCJR decoding algorithm of [18] requires $T_{\text{Log-BCJR}} = [64, 192]$ clock cycles, depending on the specific frame length of $N \in [2048, 6144]$. However, the duration of each clock cycle is dependent upon the length of the critical path through each algorithmic block in the FPTD algorithm. In [23], it was shown that the critical path of the FPTD algorithm comprises 7 datapath stages compared to the 3 of the state-of-the-art Log-BCJR decoder, giving $D_{\mathrm{FPTD}} = 7/3 D_{\text{Log-BCJR}}$. However, we will show in Section III that the refinements of (2)-(5) reduce the FPTD critical path to 6 stages, giving $D_{\mathrm{FPTD}} = 2 D_{\text{Log-BCJR}}$. Furthermore, the results of Section IV will demonstrate how this reduction in critical path length affects the maximum clock frequency. However, a different number of iterations is required for the FPTD algorithm to achieve the same error correction capability as the state-of-the-art Log-BCJR decoder. More specifically, the simulation results of Figure 8 show that when communicating over an AWGN channel, $I_{\mathrm{FPTD}} = 39$ iterations are required by the floating-point FPTD of [23] to achieve the same BER performance as a floating-point Log-BCJR turbo decoder using $I_{\text{Log-BCJR}} = 6$ iterations, for both the case of using the exact max* operation of (6) and the approximate max* operation of (7). Considering all of these aspects, the overall decoding throughput ($\frac{N}{T \cdot D \cdot I}$) and latency ($T \cdot D \cdot I$) of the reduced critical datapath length based FPTD algorithm can be predicted to be 7.38 times superior to those of the state-of-the-art Log-BCJR decoding algorithm of [18], for the case where $N = 6144$, as shown in Table 1. By contrast, the FPTD of [23] was predicted to be 6.86 times superior to the Log-BCJR decoder in [23], since $D_{\mathrm{FPTD}} = 7/3 D_{\text{Log-BCJR}}$ and because $I_{\mathrm{FPTD}} = 48$ and $I_{\text{Log-BCJR}} = 8$ iterations were assumed, which are appropriate for communication over a Rayleigh fading channel.

Furthermore, in [23], the complexity per iteration of the FPTD algorithm was quantified as $C_{\mathrm{FPTD}} = 155N$ addition, subtraction and max* operations, which is roughly 51.6% lower than that employed by the Log-BCJR decoder of [18], for which $C_{\text{Log-BCJR}} = 320N$. However, the overall complexity of the FPTD is $\frac{C_{\mathrm{FPTD}} \cdot I_{\mathrm{FPTD}}}{C_{\text{Log-BCJR}} \cdot I_{\text{Log-BCJR}}} = 3.15$ times higher than that of the Log-BCJR decoder of [18], when employing $I_{\mathrm{FPTD}} = 39$ and $I_{\text{Log-BCJR}} = 6$ iterations for decoding each frame. In Section IV, we will quantify how this complexity translates into VLSI energy consumption per frame. Despite this pessimistic complexity comparison, our experimental results show that the energy consumption of the FPTD is comparable to that of the state-of-the-art turbo decoder of [16]. It is not clear however, how the energy consumption compares to that of the state-of-the-art turbo decoder of [18],

IEEE Access
The journal for rapid open access publishing

TABLE 1. Comparison of various characteristics between the proposed FPTD and the state-of-the-art Log-BCJR LTE turbo decoder for three cases, namely the estimation presented in [23], the estimation presented in this work and the post-layout simulation results. The gain between the FPTD and the Log-BCJR decoder are shown in the brackets, for the case where $N = 6144$.

	Estimation in [23]		Estimation in this work		Post-layout	
	State-of-the-art LTE algorithm of [18]	FPTD algorithm of [23]	State-of-the-art LTE algorithm of [18]	Proposed LTE FPTD algorithm	State-of-the-art LTE VLSI of [18]	Proposed LTE FPTD VLSI
Clock cycles per iteration T	$N/32$	2	$N/32$	2	$N/32+22$	2
Clock cycle duration D	3 stages	7 stages	3 stages	6 stages	$\frac{1}{450\ \text{MHz}}$	$\frac{1}{2\times100\ \text{MHz}}$
Complexity per decoding iteration C	$320N$	$155N$	$320N$	$155N$	-	-
Decoding iterations I	8	48	6	39	6	39
Overall throughput $(\frac{N}{T\cdot D\cdot I})$	$4/3$ (1x)	$N/672$ (6.86x)	$16/9$ (1x)	$N/468$ (7.38x)	2.15 Gbps (1x)	15.8 Gbps (7.35x)
Overall latency $(T\cdot D\cdot I)$	$3N/4$ (6.86x)	672 (1x)	$9N/16$ (7.38x)	468 (1x)	2.86 μs (7.33x)	0.39 μs (1x)
Overall complexity $(C\cdot I)$	$2560N$ (1x)	$7440N$ (2.91x)	$1920N$ (1x)	$6045N$ (3.15x)	4.1 μJ [16] (1x)	3.74 μJ (0.91x)
Overall resource Combinational Logic + Register + RAM	$\frac{14N}{3}+144576$ (1x)	$825N$ (29.3x)	$\frac{14N}{3}+144576$ (1x)	$825N$ (29.3x)	7.7 mm^2 (1x)	109 mm^2 (14.2x)
Normalized area (resource/throughput)	$\frac{7N}{2}+108432$ (1x)	554400 (4.27x)	$\frac{21N}{8}+81324$ (1x)	386100 (3.96x)	3.6 mm^2/Gbps (1x)	6.9 mm^2/Gbps (1.92x)

since its energy consumption was not quantified in that paper.

Moreover, the authors of [23] estimated that the overall normalized VLSI core area (measured in mm^2/Gbps) required by the FPTD algorithm may be 4.27 times higher than that of the state-of-the-art turbo decoder of [18], when $N = 6144$ and the estimated throughput gain factor of 6.86 provided in [23] is considered. However, this factor of 4.27 normalized area expansion is reduced to 3.96, when considering the estimated throughput gain factor of 7.38, obtained for the enhanced FPTD algorithm proposed in this paper, as shown in Table 1. Further to this, the post-layout results of Section IV will show that the proposed FPTD VLSI actually has a significantly lower normalized area than this pessimistic prediction.

III. FIXED-POINT FPTD VLSI

In this section, we propose a VLSI implementation of the refined FPTD algorithm of Section II, based on the schematics of Figures 9 and 10. More specifically, Figure 9 depicts the proposed processing element, which closely approximates the function of each algorithmic block shown in Figure 4. Meanwhile, Figure 10 portrays the proposed implementation of the three algorithmic blocks that form each termination unit shown in Figure 4. In order to implement the odd-even operation described in Section II, the FPTD VLSI core is designed to perform the first half-iteration by operating the processing elements of Figure 5 on the rising clock edge, then operating the remaining processing elements of Figure 6 on the falling clock edge in order to perform the second half-iteration. Thus, each iteration requires $T = 2$ clock edges rather than $T = 2$ clock cycles as discussed in Section II. For the sake of simplicity, our following discussions redefine the notation D as the duration between clock edges, rather than the clock cycle duration, accordingly. Furthermore, in this work we assume that the FPTD VLSI core is integrated with the demodulator and other physical layer components onto a single chip, enabling all the channel LLRs of $[\bar{b}^{\mathrm{a,u}}_{2,k}]^{N+3}_{k=1}$, $[\bar{b}^{\mathrm{a,u}}_{3,k}]^{N}_{k=1}$, $[\bar{b}^{\mathrm{a,u}}_{1,k}]^{N+3}_{k=N+1}$, $[\bar{b}^{\mathrm{a,l}}_{2,k}]^{N+3}_{k=1}$ and $[\bar{b}^{\mathrm{a,l}}_{1,k}]^{N+3}_{k=N+1}$ to be fed from the demodulator to the FPTD core in parallel, within a single clock cycle. Indeed, the Fast Fourier Transform (FFT) operation used in Orthogonal Frequency Division Multiplexing (OFDM) based demodulation natively outputs all the channel LLRs in parallel [38].

The rest of this section is structured as follows. In Section III-A, we discuss the implementation of (2), (3), (4), (5) within the proposed processing element of Figure 9, as well as the implementation of (2) and (4) within the

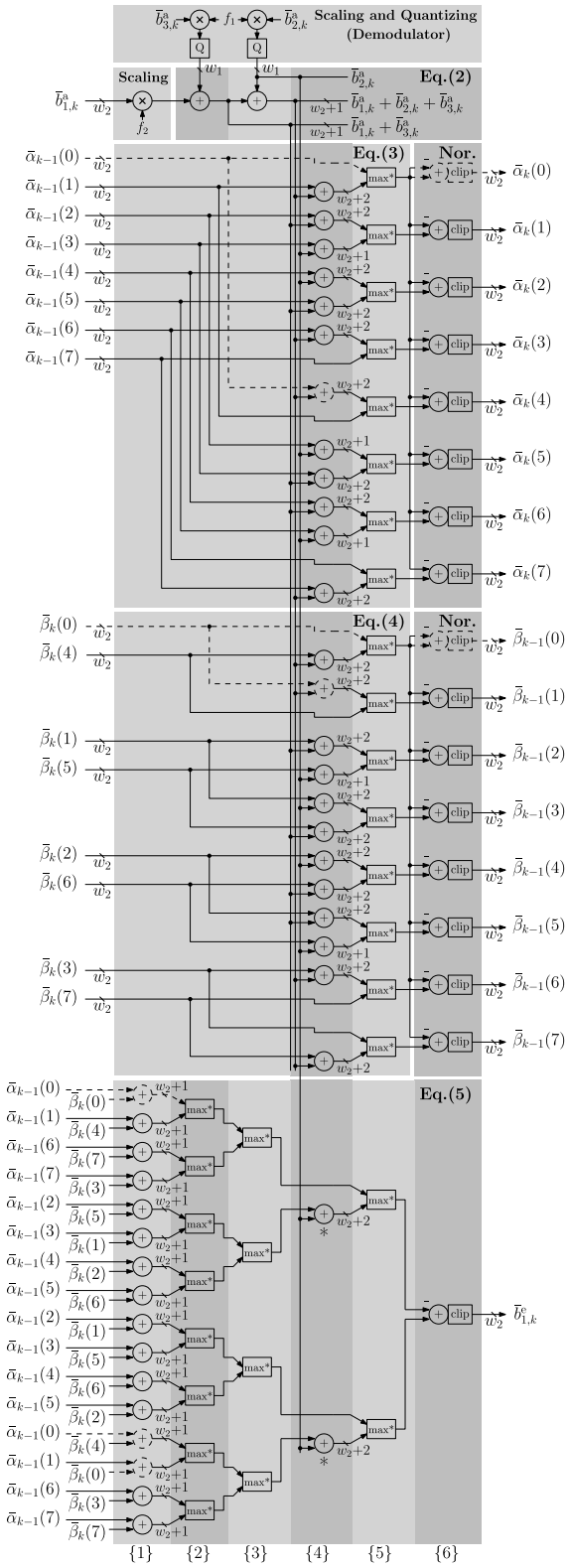


FIGURE 9. The datapath for the k^{th} processing element of the proposed fixed-point FPTD for the case of the LTE turbo code. The six datapath stages are distinguished by the dark/light shading and are indexed as shown in the curly brackets.

proposed termination unit of Figure 10. In Section III-B, we discuss the specific number representation technique and the corresponding bit widths used in Figures 9 and 10.

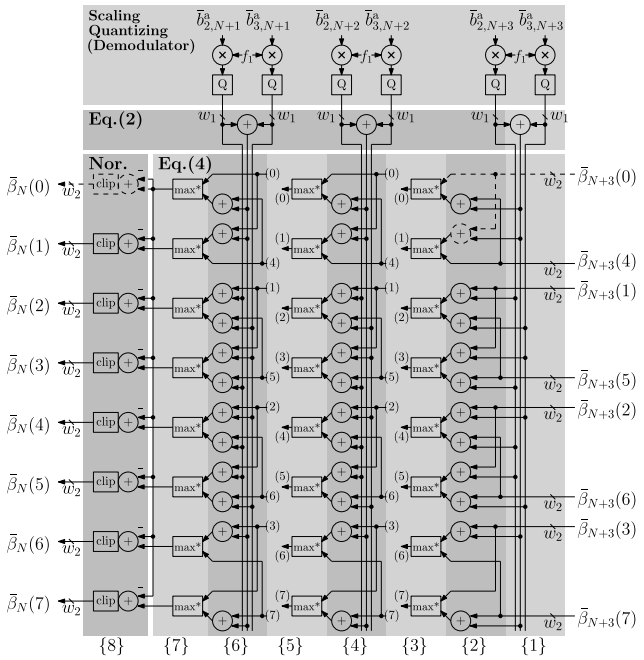


FIGURE 10. The datapath for the termination unit of the proposed fixed-point FPTD for the case of the LTE turbo code. The eight datapath stages are distinguished by the dark/light shading and are indexed as shown in the curly brackets.

Additionally, the quantization of the *a priori* channel LLRs ($\bar{b}^{\text{a}}_{2,k}$, $\bar{b}^{\text{a}}_{3,k}$), as well as the clipping of the extrinsic message LLR ($\bar{b}^{\text{e}}_{1,k}$) and extrinsic state metrics ($\bar{\boldsymbol{\alpha}}_k$, $\bar{\boldsymbol{\beta}}_{k-1}$) are also detailed in Section III-B. Section III-C introduces the scaling technique that is applied to the *a priori* message LLR $\bar{b}^{\text{a}}_{1,k}$ in Figure 9, for the sake of improving the BER performance. In Section III-D, we consider normalization techniques for controlling the dynamic range of the extrinsic state metrics ($\bar{\boldsymbol{\alpha}}_k$, $\bar{\boldsymbol{\beta}}_{k-1}$) of each processing element, in order to prevent overflow, when the fixed-point number representation is used. In Section III-E, we propose a novel bypass mechanism, which allows an FPTD having a hard-wired interleaver to additionally support various shorter frame lengths, having compatible interleaver designs.

A. IMPLEMENTATION OF THE ALGORITHMIC BLOCK AND TERMINATION UNIT

Figure 9 depicts the datapath of the processing element proposed for computing (2), (3), (4), (5) within each algorithmic block of Figure 4. All processing of Figure 9 is completed using only a single clock edge, which causes the signals to propagate through six equal-length datapath stages within the duration D. The first stage implements a multiplication for message LLR scaling, which will be detailed in Section III-C. Since $b_3(S_{k-1}, S_k) \equiv b_1(S_{k-1}, S_k)$ for the LTE turbo code, there are only four possible values that $\bar{\gamma}_k(S_{k-1}, S_k)$ can adopt as a result of (2), namely $\bar{b}^{\text{a}}_{2,k}$, $\bar{b}^{\text{a}}_{1,k}+\bar{b}^{\text{a}}_{3,k}$, $\bar{b}^{\text{a}}_{1,k}+\bar{b}^{\text{a}}_{2,k}+\bar{b}^{\text{a}}_{3,k}$ and zero. Therefore, (2) can be implemented as two consecutive additions, occupying the second and the third datapath stages, as shown in Figure 9.

Following the computation of (2), each extrinsic state metric of (3) and (4) can be calculated using two parallel additions followed by a max* operation, which occupy the fourth and fifth datapath stages, respectively. Note that some transitions have a metric $\bar{\gamma}_k(S_{k-1}, S_k)$ of zero, allowing the corresponding additions in (3) and (4) to be omitted in order to reduce the complexity, imposed as shown in Figure 9. Note that the approximate max* operation of (7) is used for the computation of (3) and (4), owing to its lower computational complexity than the exact max* operation of (6). More explicitly, the max*$(\bar{\delta}_1, \bar{\delta}_2)$ computation of (7) can be calculated by using a generic ripple adder to calculate $\bar{\delta}_1 - \bar{\delta}_2$, then using a multiplexer to select either $\bar{\delta}_1$ or $\bar{\delta}_2$ depending on the polarity of the subtraction result. Owing to this, a datapath stage implementing the approximate max* operation of (7) can be said to have a similar length to the one implementing a generic ripple addition. Note that following (3) and (4), a state metric normalization step is performed in the sixth datapath stage in order to manage overflow, as will be discussed in Section III-D.

By comparison, (5) comprises three stages of additions and three stages of max* operations, requiring the six datapath stages, as shown in Figure 9. Note that the total number of additions required by (5) can be reduced by exploiting the relationship max*(A + C, B + C) = max*(A, B) + C [39], which holds for both the exact max* of (6) and the approximate max* of (7). More specifically, the term $b_2(S_{k-1}, S_k) \cdot \bar{b}^{\mathrm{a}}_{2,k} + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k)$ in (5) requires sixteen additions for computing $\bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k)$ for the sixteen transitions and some extra additions for adding the term $b_2(S_{k-1}, S_k) \cdot \bar{b}^{\mathrm{a}}_{2,k}$, as shown in Figure 9. More specifically, eight of the transitions in Figure 9 correspond to a bit value of $b_2(S_{k-1}, S_k) = 0$, which results in the term $b_2(S_{k-1}, S_k) \cdot \bar{b}^{\mathrm{a}}_{2,k} = 0$, allowing the corresponding additions to be omitted. Furthermore, by grouping together the remaining eight transitions that correspond to a bit value of $b_2(S_{k-1}, S_k) = 1$, the addition of $\bar{b}^{\mathrm{a}}_{2,k}$ to the corresponding $\bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k)$ terms can be carried out after the following max* operations. Owing to this, only two additions are required for the $b_2(S_{k-1}, S_k) \cdot \bar{b}^{\mathrm{a}}_{2,k}$ terms in (5), as indicated using * at the fourth datapath stage of Figure 9.

Note that the algorithmic block of Figure 9 requires only six datapath stages, rather than the seven or eight stages that are identified for the FPTD algorithm in [23]. This reduction is achieved by merging [23, eq. (2) and (5)], in order to form (5) in this work. Furthermore, this modification reduces the average number of additions and subtractions required by each algorithmic block from 47.5 to 45, excluding the message LLR scaling and state metric normalization operations that will be described in Sections III-C and III-D. Note however that the modifications proposed here are optimized for the LTE turbo code and may not be applicable to other codes, such as the duo-binary WiMAX turbo code.

By contrast, the termination unit of Figure 10 requires eight datapath stages in order to implement the three consecutive algorithmic blocks, which operate on the basis of only (2) in conjunction with $L = 2$ and (4), in order to convert the termination LLRs $\bar{b}^{\mathrm{a}}_{1,N+1}$, $\bar{b}^{\mathrm{a}}_{1,N+2}$, $\bar{b}^{\mathrm{a}}_{1,N+3}$, $\bar{b}^{\mathrm{a}}_{2,N+1}$, $\bar{b}^{\mathrm{a}}_{2,N+2}$ and $\bar{b}^{\mathrm{a}}_{2,N+3}$ into the extrinsic backward state metrics $\bar{\boldsymbol{\beta}}_N$. As shown in Figure 10, the first datapath stage is used for calculating (2) for all three termination blocks. Then the following six datapath stages are used for calculating (4) for the three algorithmic blocks in a backward recursive manner, where calculating (4) for each algorithmic block requires two datapath stages. The final datapath stage is occupied by normalization, which will be described in Section III-D. Note that although the termination delay of the unit's eight datapath stages is longer than that of the six stages used by the processing element of Figure 9, the termination unit does not dictate the critical path length of the FPTD, which remains six datapath stages. This is because the termination units only need to be operated once before the iterative decoding process begins, as described in Section II-B. Intuitively, this would imply that the termination units impose a delay of two clock edges before the iterative decoding process was begin. However, in the proposed implementation the operation of the termination units starts at the same time as the iterative decoding process. Therefore, the termination units do not impose a delay of two clock edges before the iterative decoding process can begin, but the correct backward state metrics $\bar{\boldsymbol{\beta}}_N$ cannot be guaranteed during the first decoding iteration, which is performed during the first two clock edges. However, our experimental results show that this does not cause any BER degradation.

B. NUMBER REPRESENTATION AND BIT WIDTHS

Our FPTD VLSI core operates on the basis of fixed-point arithmetic, which is motivated by the observation that the LLRs typically have a low dynamic range [40]. More specifically, the two's complement number representation is employed owing to its efficiency for addition, subtraction and maximum calculations, which dominate the processing elements of Figures 9 and 10. In the proposed FPTD VLSI, the fixed-point numbers have various bit widths w, allowing the representation of values in the range $[-2^{w-1}, 2^{w-1} - 1]$. More specifically, a bit width w_1 is employed for the *a priori* parity LLRs $\bar{b}^{\mathrm{a}}_{2,k}$ and the systematic LLRs $\bar{b}^{\mathrm{a}}_{3,k}$, while $w_2 > w_1$ is employed for the *a priori* and extrinsic message LLRs $\bar{b}^{\mathrm{a}}_{1,k}$ and $\bar{b}^{\mathrm{e}}_{1,k}$, as well as for the *a priori* and extrinsic state metrics $\bar{\boldsymbol{\alpha}}_{k-1}$, $\bar{\boldsymbol{\alpha}}_k$, $\bar{\boldsymbol{\beta}}_k$ and $\bar{\boldsymbol{\beta}}_{k-1}$.

As shown at the top of Figure 9, the *a priori* parity LLR $\bar{b}^{\mathrm{a}}_{2,k}$ and the systematic LLR $\bar{b}^{\mathrm{a}}_{3,k}$ are provided by the demodulator, in which a quantizer is employed for converting the real-valued LLRs to fixed-point LLRs having the bit width w_1. It is assumed that the modulator applies noise-dependent scaling [40] to the *a priori* LLRs $\bar{b}^{\mathrm{a}}_{2,k}$ and $\bar{b}^{\mathrm{a}}_{3,k}$ prior to the quantizer, in order to prevent a significant BER performance degradation owing to quantization distortion. More specifically, the linear scaling factor

of $f_1 = v \cdot (x \cdot E_b/N_0 + y)$ is employed for communication over an AWGN channel, where $v = 2^{w_1-1}$ is the range corresponding to the resolution of the quantizer, while x and y are coefficients. For the quantizer having bit widths of $w_1 = \{3, 4, 5, 6\}$ bits, the optimal values of these coefficients are $x = \{0.0375, 0.0275, 0.0275, 0.0275\}$ and $y = \{0.39, 0.3, 0.27, 0.25\}$, as discussed in [40]. Note that since these scaling and quantization operations are assumed to be performed by the demodulator, they are not implemented in the proposed FPTD VLSI core.

While the values of $\bar{b}^{\mathrm{a}}_{2,k}$ and $\bar{b}^{\mathrm{a}}_{3,k}$ do not change during the iterative decoding process, the magnitudes of the *a priori* and extrinsic message LLRs $\bar{b}^{\mathrm{a}}_{1,k}$ and $\bar{b}^{\mathrm{e}}_{1,k}$ tend to grow in successive iterations. Therefore, $\bar{b}^{\mathrm{a}}_{1,k}$ and $\bar{b}^{\mathrm{e}}_{1,k}$ are likely to reach significantly higher magnitudes than $\bar{b}^{\mathrm{a}}_{2,k}$ and $\bar{b}^{\mathrm{a}}_{3,k}$, during the iterative decoding process. Owing to this, the BER results of Figure 13 reveal that in order to prevent saturation or overflow causing error floors at high E_b/N_0 values, the LLRs $\bar{b}^{\mathrm{a}}_{1,k}$ and $\bar{b}^{\mathrm{e}}_{1,k}$ require bit widths that are at least two bits wider than those of the *a priori* channel LLRs $\bar{b}^{\mathrm{a}}_{2,k}$ and $\bar{b}^{\mathrm{a}}_{3,k}$, giving $w_2 \geqslant w_1 + 2$. Note that Figure 13 will be discussed in greater detail in Section III-D.

Upon adding two fixed-point numbers, overflow can be avoided by setting the bit width of the result to be one bit wider than the widest of the two operands. When adopting this strategy for the calculations of (2)-(5), the widest bit width required for the intermediate values within the processing elements is $(w_2 + 2)$, in the case where $w_2 = (w_1 + 2)$. More specifically, the intermediate variable $(\bar{b}^{\mathrm{a}}_{1,k} + \bar{b}^{\mathrm{a}}_{3,k})$ of (2) requires $(w_2 + 1)$ bits, as shown in Figure 9. Similarly, the intermediate variable $(\bar{b}^{\mathrm{a}}_{1,k} + \bar{b}^{\mathrm{a}}_{2,k} + \bar{b}^{\mathrm{a}}_{3,k})$ of (2) also requires $(w_2 + 1)$ bits, since the variable $(\bar{b}^{\mathrm{a}}_{2,k} + \bar{b}^{\mathrm{a}}_{3,k})$ requires $(w_1 + 1)$ bits, which is shorter than the w_2 bits of $\bar{b}^{\mathrm{a}}_{1,k}$ when $w_2 \geqslant w_1 + 1$.

For the calculations of (3) and (4), the sixteen results of the first stage of additions (in the fourth datapath stage) require different bit widths up to $(w_2 + 2)$. More specifically, some of the sixteen results do not require an addition, allowing the bit width to be maintained at w_2. By contrast, $(w_2 + 1)$ bits are required for the results obtained by adding the *a priori* state metric with the *a priori* LLR $\bar{b}^{\mathrm{a}}_{2,k}$. Meanwhile, $(w_2 + 2)$ bits are required for the results obtained by adding the *a priori* state metric to the intermediate variables $(\bar{b}^{\mathrm{a}}_{1,k} + \bar{b}^{\mathrm{a}}_{3,k})$ or $(\bar{b}^{\mathrm{a}}_{1,k} + \bar{b}^{\mathrm{a}}_{2,k} + \bar{b}^{\mathrm{a}}_{3,k})$, as shown in Figure 9. Following these additions, (3) and (4) perform the approximate max* operation of (7) in the fifth datapath stage, although this does not require an extra bit, since the output of (7) is given by replicating one of its operands. As will be described in Section III-D, the normalization of (3) and (4) is achieved by performing a subtraction in the sixth datapath stage, which requires an additional bit. However, this is followed by clipping, which reduces the bit width of the extrinsic state metrics from up to $(w_2 + 2)$ bits to w_2 bits. Here, clipping is achieved by reducing the magnitude of any values that are outside the range that can be represented using the bit width of w_2 to the boundary values of that range, namely to -2^{w_2-1} for a negative number having an excessive magnitude and $2^{w_2-1} - 1$ for a positive number. When calculating (5), bit widths of $(w_2 + 1)$ and $(w_2 + 2)$ are respectively required for the intermediate values that result from the additions in the first and fourth datapath stages, as shown in Figure 9. Similarly, the final subtraction in (5) is followed by clipping, which guarantees that the final extrinsic LLR $\bar{b}^{\mathrm{e}}_{1,k}$ has the same bit width w_2 as the *a priori* LLR $\bar{b}^{\mathrm{a}}_{1,k}$.

Note that although state-of-the-art Log-BCJR turbo decoder implementations typically use a wider bit width for the forward state metrics and backward state metrics than that used for the message LLRs, our experimental results show that the proposed FPTD does not benefit from any BER improvement, when using unequal bit widths for the state metrics and the message LLRs. This may be because in all datapaths used for calculating the extrinsic state metrics and the extrinsic message LLRs, the bitwidth grows from w_2 to $w_2 + 2$ before the clipping, as shown in Figure 9. As a result, the clipping cuts off two out of $w_2 + 2$ bits, hence reducing the dynamic range of every calculated value by a factor of $2/(w_2 + 2)$. However, if a wider bitwidth of $w_3 > w_2$ is used for the state metrics, then $w_3 + 2 - w_2$ out of $w_3 + 2$ bits must be clipped from the extrinsic LLRs, in order to maintain the desired bitwidth of w_2 and vice versa. This results in a dynamic range loss by a factor of $(w_3 + 2 - w_2)/(w_3 + 2)$, which is higher than $2/(w_2 + 2)$, when $w_3 > w_2$. Therefore, using different bitwidths for the state metrics and the message LLRs in the proposed datapath of Figure 9 imposes a higher dynamic range loss for one compared to the other, which may result in BER degradations, rather than improvements as in the state-of-the-art Log-BCJR turbo decoder implementations.

Additionally, the BER performance of the proposed fixed-point FPTD having various bit widths of $(w_1, w_2) = \{(3, 5), (4, 6), (5, 7), (6, 8)\}$ is compared in Figure 11. It may be observed that the BER performance improves significantly upon increasing the bit width from (3, 5) to (4, 6) and then to (5, 7). However, the improvement becomes much smaller upon increasing the bit width any further.

C. MESSAGE LLR SCALING

In addition to the noise-dependent scaling that is applied to $\bar{b}^{\mathrm{a}}_{2,k}$ and $\bar{b}^{\mathrm{a}}_{3,k}$ by the demodulator as described in Section III-B, the BER performance of conventional Log-BCJR turbo decoders that employ the approximate max* operation of (7) can be improved by scaling the *a priori* LLR $\bar{b}^{\mathrm{a}}_{1,k}$ [41]. The optimal scaling factor was found to be $f_2 = 0.7$ in [41] for the case of conventional Log-BCJR turbo decoders employing floating-point arithmetic. Inspired by this, our BER simulations of Figure 8 show that applying a scaling factor of $f_2 = 0.7$ to $\bar{b}^{\mathrm{a}}_{1,k}$ is also beneficial for the floating-point FPTD algorithm employing the approximate max* operation of (7), offering about 0.2 dB BER gain in

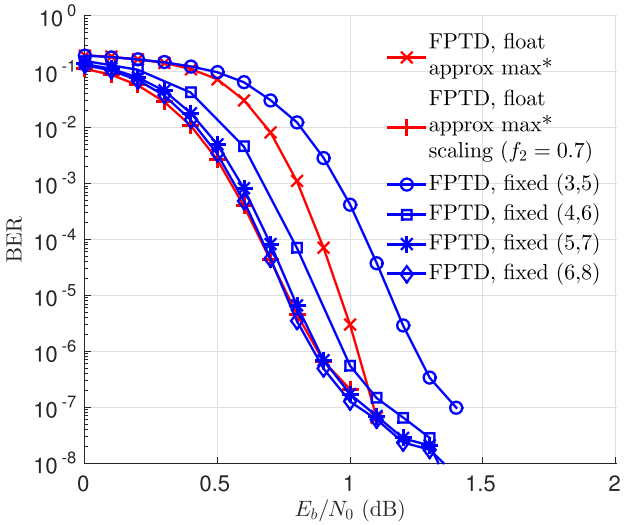


FIGURE 11. BER performance of the fixed-point FPTD using the approximate max* operation of (7), message LLR scaling ($f_2 = 0.75$), *state-zero* state metric normalization and various bit widths (w_1, w_2). The BER performance is compared to that of the floating-point FPTD using the approximate max* operation of (7), both with and without message LLR scaling ($f_2 = 0.7$). The BER was simulated for the case of transmitting $N = 6144$-bit frames over an AWGN channel, when performing $I = 39$ decoding iterations.

			1	0	0	1	0	1 $_\triangle$	−27
×						0 $_\triangle$	1	1	0.75
Sign Extension	1	1	1	0	0	1	0	1	
+	①	1	0	0	1	0	1		<< 1
Truncate	1	0	1	0	1	1 $_\triangle$	1	1	−20.25
	1	0	1	0	1	1 $_\triangle$			−21

FIGURE 12. An example of two's complement multiplication, where the multiplicand is an integer and the multiplier is 0.75. The truncation of *floor* is applied to the product after the decimal point ($_\triangle$).

the turbo-cliff region. Furthermore, our results of Figure 11 show that the proposed fixed-point FPTD VLSI benefits from applying a scaling factor of $f_2 = 0.75$. In the case of employing bits widths of $(w_1, w_2) = (6, 8)$, this results in the same BER performance as the floating-point FPTD having a scaling factor of $f_2 = 0.7$. Note that a scaling factor of $f_2 = 0.75$ requires lower hardware complexity than one of $f_2 = 0.7$ for the case of fixed-point implementation. More specifically, by exploiting the two's complement multiplication arithmetic illustrated in Figure 12, the message LLR scaling factor of 0.75 may be applied to the *a priori* LLR $\bar{b}^{\mathrm{a}}_{1,k}$ using two steps. In the first step, $\bar{b}^{\mathrm{a}}_{1,k}$ is added to a replica of itself that has been shifted to the left by one bit position, according to $\bar{b}^{\mathrm{a}}_{1,k} + (\bar{b}^{\mathrm{a}}_{1,k} \ll 1)$. Then in a second step, a *floor* truncation [30] is applied to the two least significant bits of the result, which maintains the same bit width as that employed before the message LLR scaling. Here, the bit shifting can be carried out by hard-wiring, since the scaling factor of $f_2 = 0.75$ is fixed throughout the iterative decoding process. Note that our experiments show that floor truncation imposes no sensible degradation on the BER performance of the proposed fixed-point FPTD, compared to employing other truncation methods, such as *ceil*, *fix* and *round* [30]. Furthermore, *floor* truncation can be implemented with the aid of hard-wiring as well, hence avoiding the requirement for any additional hardware apart from an adder. For this reason, message LLR scaling occupies only the first datapath stage of Figure 9, which has the same length as all other datapath stages.

D. STATE METRIC NORMALIZATION

When performing successive iterations during the iterative decoding process, the values of the extrinsic state metrics $\bar{\boldsymbol{\alpha}}_k$ and $\bar{\boldsymbol{\beta}}_{k-1}$ can grow without upper bound [28]. In order to prevent any potential BER error floors that may be caused by saturation or overflow, state metric normalization is required for reducing the magnitudes of $\bar{\boldsymbol{\alpha}}_k$ and $\bar{\boldsymbol{\beta}}_{k-1}$ in order to ensure that they remain within the range that is supported by their bit width w_2. As shown in Figure 9, normalization is performed in the sixth datapath stage within each processing element. This is achieved by subtracting a constant from all extrinsic forward state metrics $[\bar{\alpha}_k(S_k)]_{S_k=0}^{M-1}$ and all extrinsic backward state metrics $[\bar{\beta}_{k-1}(S_{k-1})]_{S_{k-1}=0}^{M-1}$, where $M = 8$ for LTE [27], [28]. Note that subtracting a constant value from the extrinsic state metrics does not change the information that they convey, since this is carried by their differences, rather than by their absolute values. Note that the subtracted constants may adopt different values for the forward and backward extrinsic state metrics of $\bar{\boldsymbol{\alpha}}_k$ as well as $\bar{\boldsymbol{\beta}}_{k-1}$ and may adopt different values in processing elements having different indices $k \in [1, N]$.

Conventionally, the constant subtracted from a set of $M = 8$ extrinsic state metrics is either their maximum [28] or their minimum [27]. However, both methods impose a computational overhead for obtaining the maximum or the minimum, which would require extra circuits and additional datapath stages in the fixed-point FPTD, hence increasing its core area and propagation delay D. More specifically, searching for the maximum or minimum of $M = 8$ extrinsic state metrics would require three successive pairwise operations, occupying three datapath stages, which would significantly increase the number of datapath stages employed in the processing element of Figure 9 beyond stage six. Although an improved maximum-finding algorithm [42] may be employed to reduce this degradation, it would still extend the critical path length and increase the area of the proposed FPTD VLSI. Alternatively, the *modulo* normalization technique [18], [28] is capable of normalizing the state metrics without requiring any subtraction, imposing no computational overhead. However, in order to avoid any BER degradation, the bit width used for representing the state metrics has to be increased by at least two bits in the fixed-point Log-BCJR turbo decoder implementations, as well as in the proposed fixed-point FPTD VLSI. This would cause

an overall increase in bitwidth that is required by the proposed fixed-point FPTD VLSI, resulting in a longer critical path length and hence implying a performance degradation in terms of processing throughput and processing latency. Owing to this, we instead employ the *state-zero* state metric normalization method of [10] for the proposed FPTD VLSI. More specifically, the values of $\bar{\alpha}_k(0)$ and $\bar{\beta}_{k-1}(0)$ are respectively selected for normalizing the forward state metrics $\bar{\boldsymbol{\alpha}}_k$ and the backward state metrics $\bar{\boldsymbol{\beta}}_{k-1}$ of the k^{th} processing element, as shown in Figure 9. As suggested in [10], there are two main advantages to this approach. Firstly, apart from the subtraction itself, no additional computational overhead is imposed by finding the maximum or minimum value, for example. Secondly, after the *state-zero* normalization, zero-values are guaranteed for the first extrinsic state metrics $\bar{\alpha}_k(0) = 0$ and $\bar{\beta}_{k-1}(0) = 0$. In our proposed FPTD VLSI, this allows the registers and additions involving $\bar{\alpha}_{k-1}(0)$ and $\bar{\beta}_k(0)$ to be simply removed, saving two w_2-bit registers and seven additions per processing element, as shown by the dotted lines in Figure 9. Furthermore, this approach guarantees a constant value of zero for one of the operands in three of the max* operations, simplifying them to using the sign bit of the other non-zero operand for selecting which specific operand is output.

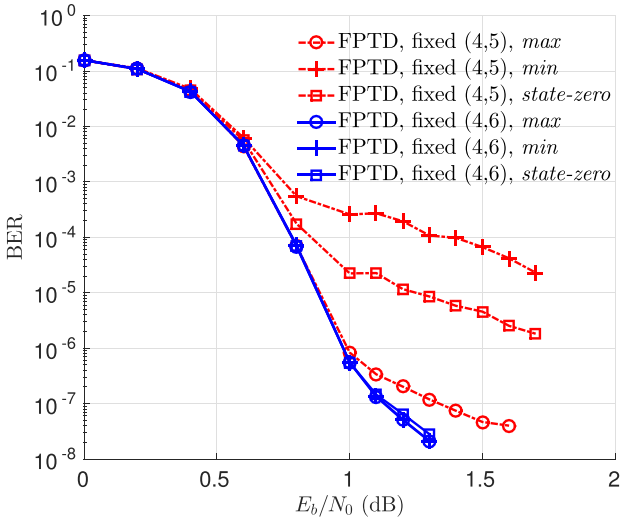


FIGURE 13. BER comparison of the fixed-point FPTD using the approximate max* operation of (7) and message LLR scaling ($f_2 = 0.75$) with three different state metric normalization methods, namely *max*, *min* and *state-zero*, as well as two different bit widths of $(w_1, w_2) = (4,5)$ and (4,6). The BER was simulated for the case of transmitting $N = 6144$-bit frames over an AWGN channel, when performing $I = 39$ decoding iterations.

Furthermore, Figure 13 shows that the *max*, *min* and *state-zero* state metric normalization methods yield the same BER performance, when the bit width w_2 is sufficiently high, having a value of $w_2 = (w_1 + 2)$. However, these normalization methods offer different BER performances in the error floor region, when w_2 is not sufficient. More specifically, when the fixed-point FPTD uses bit widths of (4, 5), the *max* normalization method offers the lowest error floor, although the *state-zero* normalization method is still superior to the *min* normalization method, in this case.

E. BYPASS UNIT

A hard-wired $N = 6144$-bit LTE interleaver is used for the proposed FPTD VLSI implementation, in order to minimize the corresponding hardware complexity. While it may seem that the employment of a hard-wired interleaver would prevent the decoding of frames having other lengths and interleaver patterns, we propose a bypass mechanism that also allows shorter frame lengths having compatible interleaver patterns to be supported by the proposed implementation. More explicitly, each processing element employs a set of $M \cdot w_2$ binary multiplexers along the paths of both the forward and the backward extrinsic state metrics, as shown in Figure 9. Each set of multiplexers is used for providing the corresponding registers with one of two selectable inputs, namely either the state metrics provided by the concatenated processing elements, or a set of state metrics that are provided from further down the row of processing elements. These two inputs correspond to two different operational modes for each processing element, namely the *normal* and *bypass* modes. More explicitly, if the *normal* mode is selected for the k^{th} processing element, then it will process both forward and backward state metrics. By contrast, if the *bypass* mode is selected, then a direct link is bridged over for the state metrics between the $(k-1)^{\text{st}}$ and $(k+1)^{\text{st}}$ processing elements, bypassing the k^{th} processing element.

Upon decoding frames having a frame length of $N < 6144$, only a set of N processing elements are required in each row, allowing the remaining processing elements to be switched off during the iterative decoding process. Accordingly, this may be achieved by selecting the *bypass* mode for the deactivated processing elements, in order to guarantee that both the forward and backward state metrics can propagate to all activated processing elements. Note that we assume that all *a priori* systematic LLRs and *a priori* parity LLRs representing the N-bit frame can be correctly fed to the corresponding registers shown in Figure 4 from the demodulator. By carefully selecting, which specific processing elements are placed in the *bypass* mode, different compatible interleaver patterns can be implemented. In the simple example of Figure 14, the FPTD employs $N = 10$-bit in conjunction with the hard-wired interleaver having the pattern of $\pi_1 = \{7, 8, 9, 6, 3, 10, 1, 4, 5, 2\}$, which may be configured to decode $N = 4$-bit frames having the interleaver pattern $\pi_2 = \{3, 4, 1, 2\}$. This is achieved by selecting the *normal* mode for the processing elements in the lower row having the indices $k = \{1, 2, 5, 8\}$, while for the processing elements in the upper row the indices $k = \{3, 4, 7, 8\}$ may be employed. Meanwhile, the *bypass* mode is selected for the remaining processing elements having the indices of $k = \{3, 4, 6, 7, 9, 10\}$ in the lower row and $\pi_1(k) = \{1, 2, 5, 6, 9, 10\}$ in the upper row.

The control of this bypass mechanism may be implemented as shown in Figure 4. More specifically, when the

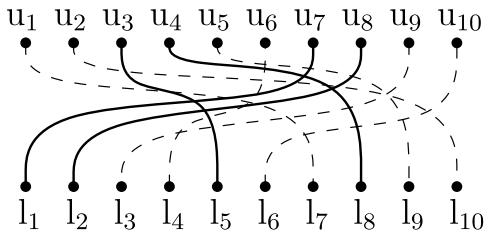


FIGURE 14. An example of decoding short frames using a FPTD, in which both upper and lower decoders comprise $N = 10$ processing elements that are connected by the hard-wired interleaver of $\pi_1 = \{7, 8, 9, 6, 3, 10, 1, 4, 5, 2\}$, as shown by both the solid and dashed lines. When decoding frames having $N = 4$ with the interleaver of $\pi_2 = \{3, 4, 1, 2\}$, only the lower processing elements having the indices $k = \{1, 2, 5, 8\}$ and the upper processing elements having the indices $k = \{3, 4, 7, 8\}$ are employed, using the interleaver connections shown by the solid lines.

FPTD is required to support K different interleaver patterns, these may be selected using control bits. As shown in Figure 4, a decoding circuit may be employed for accepting the $\lceil \log_2(K) \rceil$ control bits $\boldsymbol{C}$, which processes K Boolean outputs of $\boldsymbol{N} = \{N_1, N_2, \ldots, N_K\}$ corresponding to the K different interleaver patterns. In any particular configuration, only one of the K outputs is asserted, while the others remain at zero. As shown in Figure 9, this allows the bypass unit of each processing element to be controlled by a corresponding tristate box, which comprises K tristate gates, each controlled by the corresponding Boolean signal gleaned from $\boldsymbol{N}$. When selected, each tristate passes a predefined binary value to the bypass unit of the corresponding processing element, in order to select either the *normal* or the *bypass* mode of operation. Here, each tristate may be implemented using a single transmission gate, where a single p-type MOSFET transistor may be used for outputting a logical one (VDD) or a single n-type MOSFET transistor may be used for passing a logical zero (GND). Note that the connections in the shaded region of each tristate box shown in Figure 4 may be predefined, according to the requirements of the particular interleaver patterns supported.

However, a complex offline search is required in order to determine the specific configuration necessitated by supporting a particular interleaver pattern. For example, the LTE turbo code supports $K = 188$ different interleaver patterns, each having a different length N in the range 40 to 6144 bits. Our preliminary results have shown that by using the bypass mechanism, a FPTD having the $N = 6144$-bit LTE interleaver pattern can be also configured to support all LTE interleaver patterns having the lengths of $N \in [40, 200]$. This has been determined using an algorithm that searches for a configuration of the bypass mechanism that maps a particular shorter interleaver pattern into a specific longer interleaver pattern.

In the first step of the algorithm, the search space is reduced by eliminating mappings of particular connections in the shorter interleaver into particular connections in the longer interleaver that would make the overall mapping impossible. For example, consider the mapping of the interleaver $\pi_2 = \{3, 4, 1, 2\}$ into the interleaver $\pi_1 = \{7, 8, 9, 6, 3, 10, 1, 4, 5, 2\}$ of Figure 14. Here, the connection $\pi_1(3) = 9$ of the interleaver π_1 connects the block in the lower row having the index $k = 3$ to the block in the upper row having the index $\pi_1(k) = 9$. This connection cannot be mapped to the connection $\pi_2(3) = 1$ of the shorter interleaver π_2. This is because $\pi_2(3) = 1$ must be mapped to a connection in π_1 to a block in the upper row that has at least three more blocks to its right, in order to leave room for the connections $\pi_2(4) = 2$, $\pi_2(1) = 3$ and $\pi_2(2) = 4$. Likewise, there are several other mappings that are impossible, because they do not leave enough room at the right-hand end of the top row. Furthermore, some other mappings are impossible because they do not leave enough room at the left-hand end of the top row, or at either end of the bottom row. Once all of these mappings have been eliminated, as a result some other mappings may become impossible. For example, since $\pi_1(3) = 9$ cannot be mapped to $\pi_2(3) = 1$ as described above, $\pi_1(4) = 6$ cannot be mapped to $\pi_2(4) = 2$. This is because $\pi_1(4) = 6$ connects to a block so far to the left of the bottom row that mapping it to $\pi_2(4) = 2$ would require the connections of π_2 to the first three blocks in the lower row to be mapped to the connections of π_1 to the first three blocks in the lower row. In particular, this would require $\pi_1(3) = 9$ to be mapped to $\pi_2(3) = 1$, but this has been identified as being impossible, as described above. In this way, the process can iterate, with the elimination of each potential mapping triggering the elimination of further potential mappings and so on. This process can continue, until no more eliminations are triggered. If it is determined that a particular connection in the shorter interleaver π_2 cannot be mapped to any connections in the longer interleaver π_1, then this reveals that the mapping of π_2 into π_1 is impossible, hence halting the algorithm. Table 2 shows the valid mappings of the connections of π_2 to the connections of π_1, as identified during the first step of the algorithm. Note that the technique used for identifying eliminations described above corresponds to maintaining a triangular arrangement of zeros in the bottom left and top right of both Table 2(a) and 2(b).

In the second step of the algorithm, a brute-force search of the reduced search space from the first step is employed in order to find an overall mapping of the shorter interleaver pattern onto the longer interleaver pattern. This process must consider not only whether individual connections from the longer interleaver patten can be mapped to particular connections in the shorter pattern as in the first step, but also whether their combination maintains the ordering of the blocks in the top and bottom rows. For example, the mappings $\pi_1(5) = 3$ to $\pi_2(1) = 3$, $\pi_1(2) = 8$ to $\pi_2(2) = 4$, $\pi_1(8) = 4$ to $\pi_2(3) = 1$ and $\pi_1(10) = 2$ to $\pi_2(4) = 2$ are all individually valid, as may be identified during the first step described above. Indeed, all of these mappings form part of a legitimate mapping of π_2 into π_1. However, they cannot form parts of the same mapping. In particular, this is because these mappings are listed in order of increasing k for $\pi_2(k)$, but the

TABLE 2. Valid mappings between connections of the interleaver $\pi_2 = \{3, 4, 1, 2\}$ to the connections of the interleaver $\pi_1 = \{7, 8, 9, 6, 3, 10, 1, 4, 5, 2\}$ of Figure 14. (a) The point of view from the bottom row of blocks. (b) The point of view from the top row of blocks.

	k	1	2	3	4	5	6	7	8	9	10
	$\pi_1(k)$	7	8	9	6	3	10	1	4	5	2
k	$\pi_2(k)$										
1	3	**1**	1	1	1	1	0	0	0	0	0
2	4	0	**1**	1	1	0	1	0	0	0	0
3	1	0	0	0	0	**1**	0	1	1	0	0
4	2	0	0	0	0	0	0	0	**1**	1	1

(a)

	$\pi_1(k)$	1	2	3	4	5	6	7	8	9	10
	k	7	10	5	8	9	4	1	2	3	6
$\pi_2(k)$	k										
1	3	1	0	**1**	1	0	0	0	0	0	0
2	4	0	1	0	**1**	1	0	0	0	0	0
3	1	0	0	1	0	0	1	**1**	1	1	0
4	2	0	0	0	0	0	1	0	**1**	1	1

(b)

resultant ordering of k for $\pi_1(k)$ is $\{5, 2, 8, 10\}$, which is not in increasing order. Therefore, this mapping does not maintain the correct ordering of the blocks in the bottom row. Likewise, this mapping does not maintain the correct ordering of the blocks in the top row. By contrast, the mapping shown in Figure 14 uses the mappings $\pi_1(1) = 7$ to $\pi_2(1) = 3$, $\pi_1(2) = 8$ to $\pi_2(2) = 4$, $\pi_1(5) = 3$ to $\pi_2(3) = 1$ and $\pi_1(8) = 4$ to $\pi_2(4) = 2$, which does maintain the correct ordering of the blocks in both rows. These mappings are highlighted in bold in Table 2. Note that this highlighting cascades from the top left to the bottom right of both Table 2(a) and 2(b), which indicates that it corresponds to a valid mapping. The brute-force search continues until the first legitimate mapping is found, whereupon the corresponding *normal* or *bypass* mode can be determined for each block of the FPTD.

However, this algorithm has revealed that a fully-parallel turbo decoder having the $N = 6144$-bit LTE interleaver pattern cannot be configured to support the LTE interleavers having the lengths of $N \in [784, 6080]$. Unfortunately, the complexity of our algorithm becomes excessive for the remaining LTE interleaver patterns having the lengths of $N \in [208, 768]$ and so it is not clear which of these patterns are supported by a fully-parallel turbo decoder having the $N = 6144$-bit LTE interleaver pattern. Our future work will refine the above algorithm in order to reduce its complexity for these intermediate interleaver lengths. Furthermore, we will search for the shortest fully-parallel turbo decoder interleaver pattern that can be configured using the bypass mechanism to support all $K = 188$ LTE interleaver patterns.

IV. RESULTS

In this section, we characterize the proposed FPTD VLSI core, when implemented using the TSMC 65nm LP process. These results are compared to a pair of state-of-the-art implementations of the Log-BCJR turbo decoder disseminated in [16] and [18], both of which also use the TSMC 65nm process technology. The proposed FPTD VLSI employs the bit widths of $(w_1, w_2) = (4, 6)$, which offers the same BER performance as the benchmarkers, namely a BER of 10^{-6} at an AWGN E_b/N_0 of 1 dB, as shown in Figure 11. Note that it is not feasible to perform post-layout energy consumption simulation for an entire FPTD VLSI, comprising sufficient processing elements for supporting $N = 6144$-bit LTE frames. Owing to this, our post-layout simulations consider a single processing element of Figure 9, including all the corresponding registers and the bypass unit of Figure 4, allowing both the energy consumption and the area to be characterized. We then scale these results for estimating both the energy consumption and the area for an entire $N = 6144$-bit FPTD VLSI. This approach is validated in Section IV-A by comparing the scaled results to implementations of the entire decoders having the frame lengths of $N \in \{40, 80, 160\}$. Following this, we characterize the performance of the FPTD VLSI in terms of its energy, latency, throughput and area in Sections IV-B and IV-C. Finally, Section IV-D compares the simulated FPTD VLSI's performance to the predictions made in Section II-C, as well as to some other recent Log-BCJR turbo decoder VLSI implementations.

A. PROCESSING ELEMENT AND ENTIRE FPTD VLSI

Figure 15 shows a post-layout view of a single FPTD processing element of the proposed FPTD VLSI core, designed for operating at $f_{\text{clk}} = 100$ MHz. It is 228 μm in height and 38.2 μm in width, giving an area of $\text{Area}^{\text{PE}} = 0.0087$ mm^2. Note however that this area depends on the clock frequency f_{clk} that the processing element is designed for, as will be detailed in Section IV-C. The ports are placed along the four edges of the core, adopting similar positions to those shown in Figure 9. Note that there are no input or output pads shown in the layout of Figure 15, since the entire FPTD core is assumed to be integrated into a baseband chip that also includes the demodulator. More specifically, as shown in Figure 15, the 4-bit *a priori* LLRs $\bar{b}^{\text{a}}_{2,k}$ and $\bar{b}^{\text{a}}_{3,k}$, as well as the control signals of *clock*, *reset* and *bypass* are positioned near the top edge. The 6-bit *a priori* message LLR $\bar{b}^{\text{a}}_{1,k}$ and extrinsic message LLR $\bar{b}^{\text{e}}_{1,k}$ are located near the bottom edge. In addition to these, the seven 6-bit *a priori* forward state metrics $\bar{\boldsymbol{\alpha}}_{k-1}$ and the seven 6-bit extrinsic backward state metrics $\bar{\boldsymbol{\beta}}_{k-1}$ are located along the left edge, whereas the seven 6-bit extrinsic forward state metrics $\bar{\boldsymbol{\alpha}}_k$ and the seven 6-bit *a priori* backward state metrics $\bar{\boldsymbol{\beta}}_k$ are located along the right edge. Note that the state metrics of $\bar{\alpha}_{k-1}(0)$, $\bar{\alpha}_k(0)$, $\bar{\beta}_{k-1}(0)$ and $\bar{\beta}_k(0)$ do not have to be transfered between the adjacent algorithmic blocks, since they are guaranteed to have zero-values, owing to the *state-zero* state metric normalization method described in Section III-D. Furthermore,

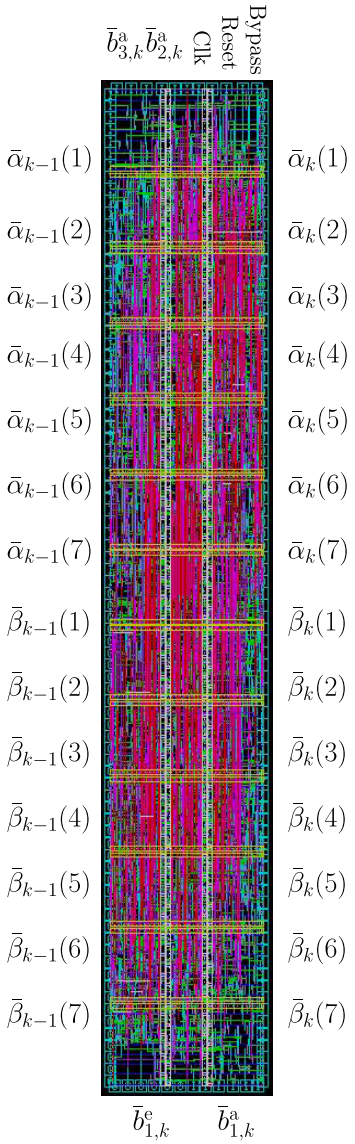


FIGURE 15. A post-layout view of a single FPTD processing element.

the locations of the ports for the forward and backward state metrics are configured for ensuring when two processing elements are placed immediately side by side, the corresponding *a priori* and extrinsic state metrics are correctly connected.

Furthermore, $2N$ of the processing elements of Figure 15 were tessellated in order to synthesize fully-fledged FPTD decoders for the frame lengths of $N \in \{40, 80, 160\}$. Owing to the tessellation, the area required by these FPTD decoders was found to be closely approximated by $2N \cdot \text{Area}^{\text{PE}}$. When also considering the tristate box and the 8-to-188 decoder shown in Figure 4, the overall core area of the proposed FPTD VLSI may be estimated according to $\text{Area}^{\text{FPTD}} = 2N \cdot (\text{Area}^{\text{PE}} + \text{Area}^{\text{Tristate box}}) + \text{Area}^{\text{8-188 decoder}}$, where Area^{PE} is the post-layout area of a single processing element obtained by the layout tool, while $\text{Area}^{\text{Tristate box}}$ and $\text{Area}^{\text{8-188 decoder}}$ are the areas for a tristate box and for the 8-to-188 decoder, respectively. For obtaining the area for an entire FPTD VLSI, both Area^{PE} and $\text{Area}^{\text{Tristate box}}$ are scaled by $2N$, since there are in total $2N$ processing elements and each processing element has an associated tristate box. Note that as discussed in Section III-E, the proposed tristate boxes comprise a number of individual n-type/p-type MOSFET transmission gates, which cannot be synthesized using a standard digital design flow based upon the standard digital gate library. It was for this reason that the bypass controller, including tristate boxes and 8-to-188 decoder, were not actually implemented in the $N \in \{40, 80, 160\}$-bit FPTD VLSIs described above. However, their areas may be estimated as follows. Since each tristate box includes $K = 188$ n-type/p-type MOSFET transistors, the area of each tristate box can be calculated as $\text{Area}^{\text{Tristate box}} = 120.3\ \mu\text{m}^2$, given that the area of an n-type/p-type MOSFET transistor is approximately $0.64\ \mu\text{m}^2$. Here, the area for a single n-type/p-type MOSFET transistor is assumed to be half the area quoted for an inverter in the TSMC 65nm datasheet [43], since this comprises one n-type transistor and one p-type transistor [44]. Furthermore, as shown in Figure 4, the $\lceil \log_2(K) \rceil = 8$ to $K = 188$ decoder may be implemented using 188 8-input AND gates in addition to eight inverters, occupying an overall area of $\text{Area}^{\text{8-188 decoder}} = 2792.6\ \mu\text{m}^2$, given that the areas for an 8-input AND gate and for an inverter are quoted in the datasheet as $14.8\ \mu\text{m}^2$ and $1.28\ \mu\text{m}^2$, respectively. In order to estimate the layout-area overhead, we enlarge the areas of the tristate box and the 8-to-188 decoder by 15%, giving $\text{Area}^{\text{Tristate box}} = 141.5\ \mu\text{m}^2$ and $\text{Area}^{\text{8-188 decoder}} = 3285.4\ \mu\text{m}^2$. Note that the routing for the hard-wired interleaver is assumed to be accommodated in the metal layers of the FPTD VLSI core, as was achieved in the $N \in \{40, 80, 160\}$-bit FPTDs described above. Owing to this, the interleaver does not require any additional area. In the case where the clock frequency of $f_{\text{clk}} = 100$ MHz is used for decoding $N = 6144$-bit frames, the total area of the proposed FPTD VLSI core becomes $\text{Area}^{\text{FPTD}} = 109\ \text{mm}^2$, although this value depends on the clock frequency f_{clk} that the VLSI is designed for, as will be detailed in Section IV-C. Note that this area is small compared to the baseband ASICs that are used in state-of-the-art LTE base stations, although it is large compared to ASICs used in mobile devices.

The energy consumption of the above-mentioned $N = \{40, 80, 160\}$-bit FPTD VLSIs were estimated under the typical operational conditions of 1.2 V and 25 °C. Figure 16 shows the dynamic energy consumption per clock edge (a half-iteration) of each processing element in each row, averaged over the iterative decoding process. These results were obtained for a clock frequency of 100 MHz using PrimeTime [45] by averaging over $I = 39$ decoding iterations of 100 frames, comprising *a priori* LLRs received from an AWGN channel having an E_b/N_0 of 1 dB. As shown in Figure 16, the processing elements near the two ends of each row consume less dynamic energy than those located in the middle, regardless of the frame length N. This may be attributed to the specific fixed values that are used for the

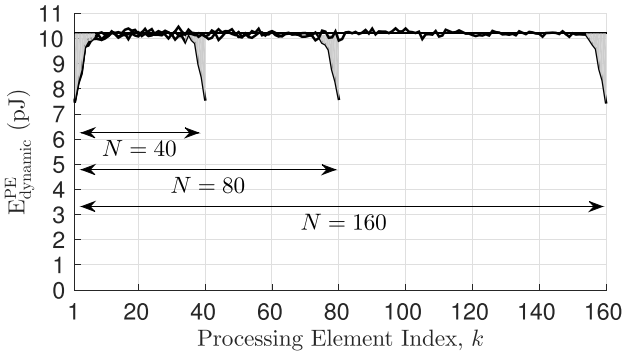


FIGURE 16. Distribution of dynamic energy consumption per clock edge for all processing elements having an index of $k \in [1, N]$ in the proposed FPTD VLSI, where $N \in \{40, 80, 160\}$. The energy consumption for each index k is obtained for V = 1.2 V and f_{clk} = 100 MHz as the average of the k^{th} upper processing element and the k^{th} lower processing element, during $I = 39$ decoding iterations of 100 frames, transmitted over an AWGN channel having $E_b/N_0 = 1$ dB.

a priori forward state metrics $\bar{\alpha}_0$ and the *a priori* backward state metrics $\bar{\beta}_N$ at the two ends of the rows. Owing to this, the logic switching that takes place during the iterative decoding process in the processing elements near the two ends is significantly lower than that which takes place in the other processing elements. Note that the static energy consumption is not quantified in Figure 16, although this may be expected to be uniformly distributed over all processing elements, since they are identical.

Owing to the nearly uniform distribution of dynamic and static energy consumption, we estimate the energy of an entire FPTD VLSI having an arbitrary length N by scaling the average energy consumption of an individual processing element located in the middle of a row. More specifically, the FPTD VLSI's overall energy consumption per clock edge (a half-iteration) can be obtained as $E^{FPTD}_{half\text{-}iteration} = N \cdot E^{PE}_{dynamic} + 2N \cdot E^{PE}_{static}$, where E^{PE} is the averaged energy per clock edge for a single processing element. The dynamic energy consumption $E^{PE}_{dynamic}$ and the static energy consumption E^{PE}_{static} are respectively scaled by N and $2N$, since the FPTD VLSI of Figure 4 comprises $2N$ processing elements, which consume static energy all the time, but only N of the processing elements consume dynamic energy on each clock edge, owing to the odd-even operation. This approach slightly overestimates the energy consumption owing to the effect of the area shaded in Figure 16, although this can be neglected, when N is sufficiently large. In the case where $E_b/N_0 = 1$ dB, V = 1.2 V and $f_{clk} = 100$ MHz, Figure 16 suggests that each processing element has a dynamic energy consumption of $E^{PE}_{dynamic} = 10.2$ pJ per clock edge, although this value depends on the values of E_b/N_0, V and f_{clk}, as we will show in Section IV-B. Likewise, our experiments reveal that each processing element has an average static energy consumption of $E^{PE}_{static} = 0.02$ pJ per clock edge, when $f_{clk} = 100$ MHz. Note that the energy consumption of the termination units is omitted from our analysis, since they are operated only once at the beginning of the iterative decoding process, consuming only a negligible amount of dynamic energy. Furthermore, the static energy consumption of the termination units can be neglected, since our experimental results reveal that this is two orders of magnitude lower than the dynamic energy consumption of an individual processing element. Similarly, the energy consumption of the bypass controller is also omitted for the same reason.

B. ENERGY AND LATENCY

The energy consumption per frame of the proposed entire FPTD VLSI may be obtained by accumulating the energy dissipation of every half-iteration according to $E^{FPTD}_{frame} = 2I \cdot E^{FPTD}_{half\text{-}iteration}$, where the required number of iterations I depends upon the E_b/N_0 value of the channel and $E^{FPTD}_{half\text{-}iteration}$ is the above-mentioned average energy consumption per clock edge, which also depends on the E_b/N_0 value of the channel, as we will show below. For example, in the case where $E_b/N_0 = 1$ dB and $f_{clk} = 100$ MHz, the overall energy per frame can be estimated as $E^{FPTD}_{frame} = 2I \cdot (N \cdot E^{PE}_{dynamic} + 2N \cdot E^{PE}_{static}) = 4.91\ \mu$J, in the case where $N = 6144$-bit frames are decoded using $I = 39$ iterations.

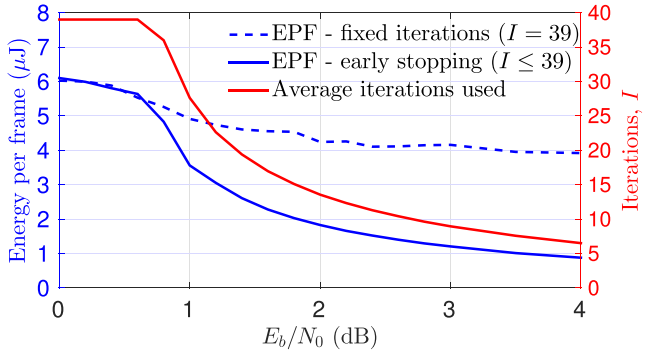


FIGURE 17. The average number of iterations required for different E_b/N_0 values, as well as the Energy consumption Per Frame (EPF) for the proposed FPTD VLSI, where $N = 6144$, V = 1.2 V and f_{clk} = 100 MHz.

Figure 17 characterizes the energy consumption per frame of the proposed FPTD VLSI as a function of E_b/N_0, when decoding the longest $N = 6144$-bit LTE frames, using a clock frequency of $f_{clk} = 100$ MHz and under the typical operational conditions of 1.2 V and 25 °C. Two different decoding approaches are compared here. In the first approach, a fixed number of $I = 39$ iterations are performed for every frame. However, as shown in Figure 17, the energy consumption per frame reduces as the E_b/N_0 value is increased, when employing the fixed-iteration approach. This is because typically fewer transmission errors occur at higher E_b/N_0 values, allowing the frame to be more easily decoded using less circuit switching. In a second decoding approach, the LTE standard's Cyclic Redundancy Check (CRC) [6], [46] may be employed to curtail iterative decoding, as soon as the message is successfully decoded, allowing fewer than $I_{FPTD} = 39$ iterations to be used by the FPTD at higher E_b/N_0 values. When

adopting this early-stopping approach, the average number of iterations performed when decoding 6144-bit frames is characterized in Figure 17 as a function of E_b/N_0. These results show that the FPTD algorithm uses $I_{\text{FPTD}} = 39$ iterations at low E_b/N_0 values, but this reduces rapidly to approximately $I = 20$ iterations at E_b/N_0 values in the turbo-cliff region ($E_b/N_0 \in [0.8, 1.4]$) and then decreasing gracefully to fewer than $I = 10$ iterations beyond the turbo-cliff region. The energy consumed when employing early stopping has a similar trend to the fixed-iteration approach, although more significant energy consumption reductions can be achieved. This may be attributed to the early-stopping mechanism, which eliminates all redundant iterations. This reduction begins at $E_b/N_0 = 0.8$ dB of the turbo-cliff region and becomes more significant as E_b/N_0 is increased.

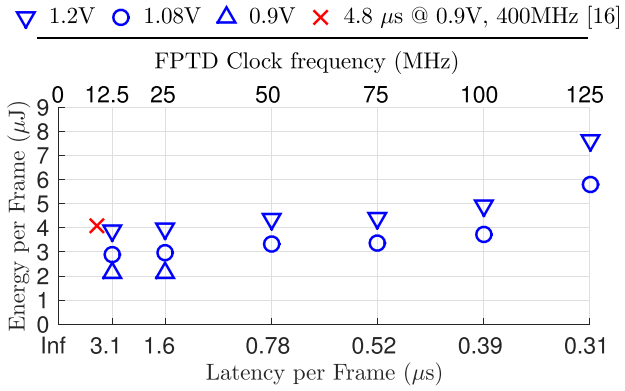


FIGURE 18. Comparison of energy consumption for the (4, 6) fixed-point FPTD VLSI when employing the fixed-iteration approach with $I = 39$ and for the benchmarker of [16], for the case of room temperature 25 °C, where TSMC 65nm is used. For the proposed FPTD VLSI, the supply voltages of 1.2 V, 1.08 V and 0.9 V are considered.

Moreover, Figures 18 and 19 quantify the energy consumption of the fixed-iteration and the early-stopping approaches, when the FPTD VLSI is employed for communication over an AWGN channel having $E_b/N_0 = 1$ dB. More specifically, post-layout simulation results are presented for the scenarios, where the proposed FPTD VLSI is synthesized using the following six different target clock frequencies of $f_{\text{clk}} \in \{12.5, 25, 50, 75, 100, 125\}$ MHz and the three different supply voltages of 1.2 V, 1.08 V and 0.9 V. Note that when the clock frequency approaches the theoretically highest value that can be achieved for the corresponding supply voltage, the synthesis tool typically invokes a higher number of gates and relies on gates having a larger driving capability, resulting in a dramatic increase both in energy consumption and in core area. Owing to this, the energy consumption can be seen to dramatically increase, when the synthesis frequency is increased from 100 MHz to 125 MHz. For example, when V = 1.08 V, the energy consumption increases from 3.74 μJ to 5.79 μJ for the fixed-iteration approach and from 2.69 μJ to 4.11 μJ for the early-stopping approach, as shown in Figures 18 and 19, respectively. On the other hand, when operating at low clock frequencies, such as 12.5 MHz and 25 MHz, the proposed FPTD VLSI may

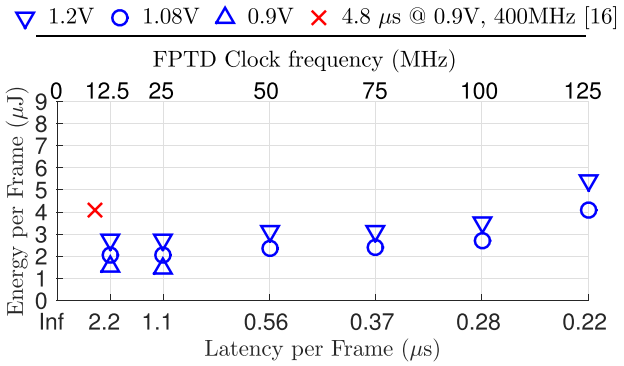


FIGURE 19. Comparison of energy consumption for the (4, 6) fixed-point FPTD VLSI employing the early-stopping approach giving $I_{\text{average}} = 28$ at $E_b/N_0 = 1$dB and for the benchmarker of [16], for the case of room temperature 25 °C, where TSMC 65nm is used. For the proposed FPTD VLSI, the supply voltages of 1.2 V, 1.08 V and 0.9 V are considered.

be powered by a low supply voltage of 0.9 V, hence achieving a significant energy reduction, consuming only 2.14 μJ for the fixed-iteration approach and 1.52 μJ for the early-stopping approach, as shown in Figures 18 and 19, respectively.

The decoding latency per frame of the proposed FPTD VLSI can be defined as Latency $= 2I \cdot D = I/f_{\text{clk}}$, since each iteration requires two clock edges, each having a duration of D. Note that owing to its serial operation, this latency is independent of the frame length N, which is in contrast to a conventional Log-BCJR turbo decoder. The proposed FPTD VLSI employing the fixed-iteration approach in conjunction with $I = 39$ results in various processing latencies per frame in the range of 3.1 μs to 0.31 μs, when operating at various clock frequencies f_{clk} from 12.5 MHz to 125 MHz, as shown in Figure 18. Here, 0.31 μs is the lowest processing latency that the proposed FPTD VLSI can achieve when decoding $N = 6144$-bit frames, which is 15.5 times and 9.2 times faster than the pair of benchmarkers in [16] and [18], which have processing latencies per $N = 6144$-bit frame of 4.8 μs and 2.86 μs, respectively. This advantage becomes even more significant when adopting the early-stopping approach. In this case, the lowest latency becomes 0.22 μs for $f_{\text{clk}} = 125$ MHz, which is 21.8 and 13 times faster than the benchmarkers of [16] and [18], respectively. As shown in Figure 19, when considering the trade off between energy consumption and decoding latency, 100 MHz may be considered to be a practical operational frequency. In this case, the proposed FPTD VLSI employing the fixed-iteration approach with $I = 39$ offers a 12.3 times lower processing latency than the benchmarker of [16] and consumes only 91% of its energy per frame of 4.1 μJ, even though the proposed FPTD VLSI uses a higher voltage of 1.08 V than the 0.9 V of the benchmarker. When employing the early-stopping approach, the latency improvement becomes a factor of 17.1 and the energy consumption reduces to 66% of that of the benchmarker of 4.1 μJ. Note that the processing latency and the corresponding energy consumption per frame may vary

from frame to frame, when the early-stopping approach is employed. However, these are bounded by those defined for the case of employing the fixed-iteration approach, since the early-stopping approach uses a maximum of $I = 39$ iterations. Note also that the energy consumption was not characterized in [18], hence we are unable to perform a similar comparison with its benchmarker. However, it may be expected that the benchmarker of [18] requires significantly more energy per frame than that of [16], since it operates at a 12.5% higher clock frequency of 450 MHz and it is powered by a 22% higher supply voltage of 1.1 V.

Note that when a processing element in the proposed FPTD VLSI is operated in the *bypass* mode, it consumes only approximately 8% of the energy that is consumed in the *normal* mode. This small energy consumption is dominated by the multiplexers of the proposed bypass mechanism, as shown in Figure 4. Additionally, a bypass unit imposes a propagation delay of 0.15 ns, which is about 3% of the clock edge duration D when $f_{\text{clk}} = 100$ MHz. However, this may degrade the achievable clock frequency, when a number of consecutive processing elements in a row are operated in the *bypass* mode. Owing to this, our future work will consider bypassing multiple processing elements at once, rather than individually bypassing consecutive processing elements. However, this may impose an additional controller complexity.

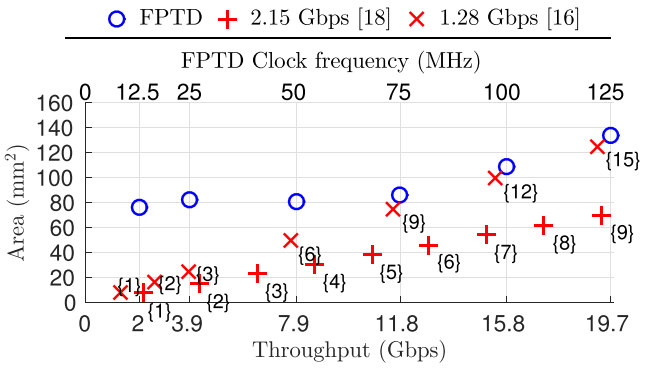


FIGURE 20. Comparison of core area for the (4, 6) fixed-point FPTD VLSI when employing the fixed-iteration approach with $I = 39$ and for the benchmarkers of [16] and [18], where TSMC 65nm is used. Note that the numbers of parallel benchmarker decoders required to achieve the same throughputs as the FPTD are presented in curly brackets.

C. THROUGHPUT AND AREA

The processing throughput of the proposed FPTD VLSI can be defined as Throughput $= \frac{N \cdot f_{\text{clk}}}{I} = \frac{N}{\text{Latency}}$. For the case of employing the fixed-iteration approach with $I = 39$, Figure 20 shows that the proposed FPTD VLSI can achieve throughputs in the range of 2 Gbps to 19.7 Gbps, when operating at clock frequencies in the range of 12.5 MHz to 125 MHz and when decoding the longest LTE frame length of $N = 6144$ bits. If the early-stopping approach is used instead of the fixed-point approach, then the processing throughputs increase to a range from 2.7 Gbps to 27.4 Gbps

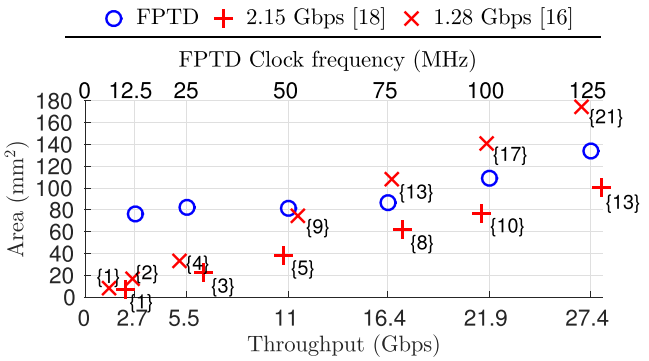


FIGURE 21. Comparison of core area for the (4, 6) fixed-point FPTD VLSI employing the early-stopping approach giving $I_{\text{average}} = 28$ at $E_b/N_0 = 1$ dB and for the benchmarkers of [16] and [18], where TSMC 65nm is used. Note that the numbers of parallel benchmarker decoders required to achieve the same throughputs as the FPTD are presented in curly brackets.

at $E_b/N_0 = 1$ dB, where $I_{\text{average}} = 28$, as shown in Figure 21. The area of the proposed FPTD VLSI exhibits a similar trend to its energy consumption. More specifically, when operating at the upper-limit approaching frequency of 125 MHz, a high number of gates having stronger driving capability are required, hence resulting in a larger VLSI area, as shown in Figures 20 and 21. On the other hand however, the area remains constant, when operating at lower clock frequencies. The clock frequency of 100 MHz may be considered to offer a beneficial trade off, resulting in an area of 109 mm^2 and a processing throughput of 15.8 Gbps, when employing the fixed-iteration approach with $I = 39$, as well as a throughput of 21.9 Gbps, when employing the early-stopping approach.

In order to achieve these throughputs using the conventional Log-BCJR decoders of [16] and [18], it is necessary to operate several of these decoders in parallel for decoding several independent frames at a time, although this is only achievable if this number of frames happens to be available at the same time. In this case, the overall core area required for this parallel operation approach is given by the area of a single turbo decoder from [16] or [18], multiplied by the number of parallel decoders required. In order to reflect this, the numbers shown in the curly brackets of Figures 20 and 21 indicate the number of parallel benchmarker decoders required for achieving the same throughputs as the proposed FPTD VLSI. In particular, the parallel operation of the twelve decoders of [16] or seven decoders of [18] is required for achieving a processing throughput of 15.8 Gbps, resulting in VLSI areas of 100 mm^2 and 54 mm^2, respectively as shown in Figure 20. As a result, the proposed FPTD VLSI employing the fixed-iteration approach with $I = 39$ has 9% and 100% larger area compared to the benchmarkers of [16] and [18], although the proposed design has the advantage of 12.3 and 7.3 times lower processing latency, respectively. Furthermore, the energy consumption of the proposed FPTD VLSI is comparable to that of the benchmarker of [16]

TABLE 3. Comparison between the proposed FPTD VLSI and different hardware implementations of the conventional Log-BCJR LTE turbo decoder.

Implementations	This work	T. Ilnseher 2012 [18]	Y. Sun 2011 [16]	M. May 2010 [14]	S. Belfanti 2013 [19]	G. Wang 2014 [47]	C. Studer 2011 [17]
Radix/Parallelism	2/6144	4/32	2/64	2/8	4/16	4/64	4/8
Iterations I	39	6	6	6.5	5.5	5.5	5.5
Technology [nm]	65	65	65	65	65	45	130
Voltage [V]	1.08	1.1	0.9	1.1	1.2	0.81	1.2
Clock frequency [MHz]	100	450	400	300	410	600	302
Core area [mm^2]	109	7.7	8.3	2.1	2.49	2.43 (5.07^a)	3.56 (0.89^a)
Throughput [Gbps]	15.8	2.15	1.28	0.15 (0.16^b)	1.01 (0.93^b)	1.67 (1.06^{ab})	0.39 (0.72^{ab})
Power [mW]	9618	-	845	300	966	870	789
Energy per frame [μJ]	3.74	-	4.1	12.3 (11.3^b)	5.88 (6.38^b)	3.20 (8.96^{ab})	12.4 (5.45^{ab})
Normalized area [mm^2/Gbps]	6.9	3.6	6.48	14.0 (13.1^b)	2.46 (2.68^b)	1.46 (4.78^{ab})	9.15 (1.24^{ab})

[a] Technology scaling to 65nm CMOS with V $= 1.08$ V, Area $\sim 1/s^2$, t_{pd} (propogation delay) $\sim 1/s$, $P_{\text{dynamic}} \sim 1/(s \cdot \text{V}_s^2)$, where s is the node size of the CMOS technology [19].

[b] Scaling linearly to $I = 6$ iterations.

and it is likely to be significantly superior to that of the benchmarker of [18], as discussed above.

Furthermore, the parallel operation of the 17 decoders of [16] or 10 decoders of [18] is required for achieving a throughput of 21.9 Gbps, like that of the proposed FPTD VLSI employing the early-stopping approach. Therefore, the proposed FPTD core's area of 109 mm^2 is 23% smaller than the area of 141 mm^2 for the benchmarker decoder of [16] and offers a 17.1 times lower latency and a 21% lower energy consumption. Although the proposed FPTD VLSI is 42% larger than the 77 mm^2 area of the benchmarker decoder of [18] in this case, it offers a 10.2 times lower processing latency and a considerably lower energy consumption.

D. COMPARISON WITH LOG-BCJR VLSIs

As described in Sections IV-A and IV-B, a supply voltage of V $= 1.08$ V and a clock frequency of $f_{\text{clk}} = 100$ MHz offers an attractive trade-off between throughput, latency, area and energy consumption. These corresponding characteristics were quantified in Sections IV-B as well as IV-C and are summarized in Table 1, in order to allow a comparison with the characteristics that were predicted in Section II-C, based on the proposed FPTD algorithm of Section II-B. In order to facilitate this comparison, Table 1 also quantifies the corresponding characteristics of the state-of-the-art LTE turbo decoder VLSI of [18]. As shown in Table 1, the state-of-the-art LTE turbo decoding algorithm of [18] requires $T = N/32$ clock cycles per decoding iteration. However, the VLSI implementation of [18] imposes an additional latency of 22 clock cycles per iteration, which is required for the pipelining and control overhead. The clock frequency of this VLSI implementation is 450 MHz, giving a clock cycle duration of $D = \frac{1}{450\text{ MHz}}$, as shown in Table 1. This VLSI implementation performs $I = 6$ decoding iterations, giving an overall throughput of 2.15 Gbps and an overall latency of 2.86 μs, as shown in Table 1. By contrast, the proposed LTE FPTD VLSI requires only $T = 2$ clock edges per iteration. Since its clock frequency is 100 MHz, the frequency of clock edges is 2×100 MHz and the clock edge duration is $D = \frac{1}{2\times 100\text{ MHz}}$. When employing the fixed-iteration approach, the proposed LTE FPTD performs $I = 39$ iterations, giving an overall throughput of 15.8 Gbps and an overall latency of 0.39 μs, as shown in Table 1. These characteristics are about 7.33 times superior to those of the state-of-the-art LTE turbo decoding VLSI implementation of [18]. This significant improvement was accurately predicted in Section II-C, where the expected improvement was 7.38 times.

However, the predictions of Section II-C related to the energy consumption and to the core area were very pessimistic. More specifically, although the overall computational complexity $C \cdot I$ of the proposed LTE FPTD algorithm is 3.15 times higher than that of the algorithm of [18], this does not translate into a correspondingly higher energy consumption. In fact, Table 1 shows that the energy consumption of the proposed LTE FPTD algorithm is lower than that of even the most energy efficient of all state-of-the-art LTE turbo

decoder VLSI implementations, namely that of [16] as we shall discuss below. Note that although [18] does not quantify the energy consumption of its VLSI implementation, this is likely to be significantly higher than that of [16], as discussed in Section IV-B. Likewise, Section II-C predicted that the normalized core area of the proposed LTE FPTD VLSI would be 3.96 times higher than that of [18], but Table 1 shows that it was actually only 1.92 times higher. The pessimism of the predictions made in Section II-C may be explained by its inability to predict the reduced dynamic energy consumption in later decoding iterations, as characterized in Figure 17. Furthermore, the area prediction of Section II-C uses a pessimistic model for the RAM area, as discussed in [23]. This area prediction also does not consider the differences between the two VLSI implementations in bitwidths, state metric normalization, interleaver implementation and controller implementation.

Table 3 compares the post-layout characteristics of the proposed LTE FPTD VLSI core with several state-of-the-art LTE turbo decoder VLSI implementations based on the Log-BCJR algorithm. Note that [23] does not present a VLSI implementation of the original FPTD algorithm, therefore it is not included in Table 3. In order to facilitate fair comparisons with the other implementations, the characteristics of the implementations using technologies other than CMOS 65 nm and using a number of iterations other than $I = 6$ have been scaled, as shown in the brackets of Table 3. Note that the proposed LTE FPTD VLSI implementation achieves the highest processing throughput, compared to all other implementations listed in Table 3. Furthermore, the proposed implementation has a lower energy consumption than the best of the other implementations, which is that of [16]. Moreover, although the proposed FPTD implementation has the largest core area, its normalized area is similar to that of [16] and is lower than that of [14].

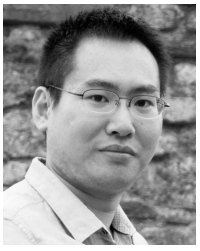
V. CONCLUSIONS

In this paper, we have proposed a fixed-point version of the LTE FPTD algorithm of [23]. We have used the design flow of Figure 2 to propose a novel VLSI implementation of the LTE FPTD, which strikes an attractive trade-off between throughput, latency, core area and energy consumption. We have investigated the techniques of message ling and state metric normalization, which improve the BER performance and prevent potential overflow, respectively. Furthermore, a bypass mechanism is proposed for allowing a hard-wired interleaver to support the decoding of frames having different lengths and interleaver patterns. Various bit widths (w_1, w_2) were simulated and (4, 6) was identified as offering an attractive trade-off between a good BER performance and a low implementation complexity. Therefore, the fixed-point FPTD employing the bit widths of (4, 6) was implemented using TSMC 65nm LP technology. When operating at 100 MHz with the supply voltage of 1.08 V, the proposed FPTD VLSI was found to offer a throughput and a latency that are 17.1 times superior to that of [16], while consuming only 66% energy per frame and having a normalized core area that is 23% smaller. The throughput and latency of the proposed FPTD VLSI are 10.2 times superior to that of [18], while likely offering a significantly superior energy consumption, although the normalized core area for the proposed VLSI is 42% larger than that for the benchmarker of [18]. Note that although the core area and the energy dissipated by the proposed $N = 6144$-bit FPTD VLSI have been determined by scaling those of a single processing element, this scaling has been carefully validated by comparing it to those of the $N = \{40, 80, 160\}$-bit FPTD VLSIs. Although the area of the proposed LTE FPTD VLSI core is larger than the capability of mobile devices at the time of writing, it is small compared to the baseband ASICs used in state-of-the-art LTE base stations, which demand a high processing throughput, a low processing latency and a low energy consumption. Our future work will further develop the parametrization of the bypass scheme introduced in Section III-E as well as techniques that can potentially further reduce the core area. In particular, a 50% reduction in area may be achieved by reusing the same processing element hardware to alternate between the processing of algorithmic blocks from the upper and lower row in alternate clock edges, although this will be achieved at increasing the switching in the circuit and hence the energy consumption.

REFERENCES

[1] J. P. Woodard and L. Hanzo, "Comparative study of turbo decoding techniques: An overview," *IEEE Trans. Veh. Technol.*, vol. 49, no. 6, pp. 2208–2233, Nov. 2000.
[2] M. Brejza, L. Li, R. Maunder, B. 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 Tuts.*, vol. PP, no. 99, pp. 1–1, Jun. 2015.
[3] Z. He, P. Fortier, and S. Roy, "Highly-parallel decoding architectures for convolutional turbo codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 10, pp. 1147–1151, Oct. 2006.
[4] O. Muller, A. Baghdadi, and M. Jezequel, "From parallelism levels to a multi-ASIP architecture for turbo decoding," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 1, pp. 92–102, Jan. 2009.
[5] *IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Broadband Wireless Access Systems*, IEEE Standard 802.16-2009, 2012.
[6] *LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and Channel Coding*, ETSI, Sophia Antipolis, France, Feb. 2013.
[7] ETSI, "Digital cellular telecommunications system (phase 2+); Radio network planning aspects (GSM 03.30 version 6.0.1 release 1997)," Eur. Telecommun. Standards Inst., Sophia Antipolis, France, Tech. Rep. 101 362, p. 136, Jul. 1998.
[8] G. Masera, G. Piccinini, M. R. Roch, and M. Zamboni, "VLSI architectures for turbo codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 3, pp. 369–379, Sep. 1999.
[9] ETSI, "Universal mobile telecommunications system (UMTS); Technical specifications and technical reports for a UTRAN-based 3GPP system," Eur. Telecommun. Standards Inst., Sophia Antipolis, France, Tech. Rep. 121 101, Jan. 2010, p. 120.
[10] A. Giulietti, B. Bougard, V. Derudder, S. Dupont, J.-W. Weijers, and L. Van der Perre, "A 80 Mb/s low-power scalable turbo codec core," in *Proc. IEEE Custom Integr. Circuits Conf.*, Orlando, FL, USA, May 2002, pp. 389–392.
[11] G. Prescher, T. Gemmeke, and T. G. Noll, "A parametrizable low-power high-throughput turbo-decoder," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 5. Philadelphia, PA, USA, Mar. 2005, pp. 25–28.
[12] 3GPP, "Overview of 3GPP release 8 V0.3.3," 3GPP Mobile Competence Centre c/o ETSI, Sophia Antipolis, France, Tech. Rep. 8, Sep. 2014.

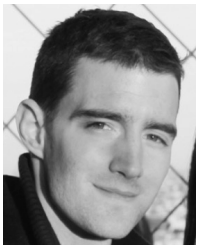
[13] C.-C. Wong, Y.-Y. Lee, and H.-C. Chang, "A 188-size 2.1 mm^2 reconfigurable turbo decoder chip with parallel architecture for 3GPP LTE system," in *Proc. Symp. VLSI Circuits*, Kyoto, Japan, Jun. 2009, pp. 288–289.
[14] M. May, T. Ilnseher, N. Wehn, and W. Raab, "A 150 Mbit/s 3GPP LTE turbo code decoder," in *Proc. Conf. Design, Autom. Test Eur.*, Dresden, Germany, Mar. 2010, pp. 1420–1425.
[15] 3GPP, "Overview of 3GPP release 10 V0.2.1," 3GPP Mobile Competence Centre c/o ETSI, Sophia Antipolis, France, Tech. Rep. 10, Jun. 2014.
[16] Y. Sun and J. R. Cavallaro, "Efficient hardware implementation of a highly-parallel 3GPP LTE/LTE-advance turbo decoder," *Integr., VLSI J.*, vol. 44, no. 4, pp. 305–315, Sep. 2011.
[17] 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.
[18] T. Ilnseher, F. Kienle, C. Weis, and N. Wehn, "A 2.15 GBit/s turbo code decoder for LTE advanced base station applications," in *Proc. 7th Int. Symp. Turbo Codes Iterative Inf. Process. (ISTC)*, Gothenburg, Sweden, Aug. 2012, pp. 21–25.
[19] S. Belfanti, C. Roth, M. Gautschi, C. Benkeser, and Q. Huang, "A 1 Gbps LTE-advanced turbo-decoder ASIC in 65 nm CMOS," in *Proc. Symp. VLSI Circuits (VLSIC)*, Kyoto, Japan, Jun. 2013, pp. C284–C285.
[20] Huawei, "5G: A technology vision," Huawei Technologies Co., Shenzhen, China, White Paper M3-023985, Nov. 2013.
[21] L. Li, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A low-complexity turbo decoder architecture for energy-efficient wireless sensor networks," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 1, pp. 14–22, Jan. 2013.
[22] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and sub-optimal maximum *a posteriori* algorithms suitable for turbo decoding," *Eur. Trans. Telecommun.*, vol. 8, no. 2, pp. 119–125, Mar. 1997.
[23] R. G. Maunder, "A fully-parallel turbo decoding algorithm," *IEEE Trans. Commun.*, vol. 63, no. 8, pp. 2762–2775, Aug. 2015.
[24] R. Achiba, M. Mortazavi, and W. Fizell, "Turbo code performance and design trade-offs," in *Proc. 21st Century Military Commun. Archit. Technol. Inf.*, vol. 1. Los Angeles, CA, USA, Oct. 2000, pp. 174–180.
[25] A. Worm, P. Hoeher, and N. Wehn, "Turbo-decoding without SNR estimation," *IEEE Commun. Lett.*, vol. 4, no. 6, pp. 193–195, Jun. 2000.
[26] Y. Wu and B. D. Woerner, "Analysis of internal data width requirements for SISO decoding modules," in *Proc. 52nd Veh. Technol. Conf.*, vol. 5. Boston, MA, USA, Sep. 2000, pp. 2265–2270.
[27] A. Worm, H. Michel, F. Gilbert, G. Kreiselmaier, M. Thul, and N. Wehn, "Advanced implementation issues of turbo-decoders," in *Proc. Int. Symp. Turbo-Codes Rel. Topics*, Brest, France, Sep. 2000, pp. 351–354.
[28] Y. Wu, B. D. Woerner, and T. K. Blankenship, "Data width requirements in SISO decoding with module normalization," *IEEE Trans. Commun.*, vol. 49, no. 11, pp. 1861–1868, Nov. 2001.
[29] Z. Wang, H. Suzuki, and K. K. Parhi, "VLSI implementation issues of TURBO decoder design for wireless applications," in *Proc. IEEE Workshop Signal Process. Syst. Design Implement.*, Taipei, Taiwan, Oct. 1999, pp. 503–512.
[30] P. H.-Y. Wu and S. M. Pisuk, "Implementation of a low complexity, low power, integer-based turbo decoder," in *Proc. IEEE Global Telecommun. Conf.* , vol. 2. San Antonio, TX, USA, Nov. 2001, pp. 946–951.
[31] X. Chen, Y. Chen, Y. Li, Y. Huang, and X. Zeng, "A 691 Mbps 1.392 mm^2 configurable radix-16 turbo decoder ASIC for 3GPP-LTE and WiMAX systems in 65 nm CMOS," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Singapore, Nov. 2013, pp. 157–160.
[32] C. Roth, S. Belfanti, C. Benkeser, and Q. Huang, "Efficient parallel turbo-decoding for high-throughput wireless systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 6, pp. 1824–1835, Jun. 2014.
[33] C. Studer, S. Fateh, C. Benkeser, and Q. Huang, "Implementation trade-offs of soft-input soft-output MAP decoders for convolutional codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 11, pp. 2774–2783, Nov. 2012.
[34] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.
[35] J. Zhang and M. P. C. Fossorier, "Shuffled iterative decoding," *IEEE Trans. Commun.*, vol. 53, no. 2, pp. 209–213, Feb. 2005.
[36] A. Nimbalker, Y. Blankenship, B. Classon, and T. K. Blankenship, "ARP and QPP interleavers for LTE turbo coding," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Mar. 2008, pp. 1032–1037.
[37] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proc. IEEE Int. Conf. Commun. ICC*, vol. 2. Seattle, WA, USA, Jun. 1995, pp. 1009–1013.
[38] I. Koffman and V. Roman, "Broadband wireless access solutions based on OFDM access in IEEE 802.16," *IEEE Commun. Mag.*, vol. 40, no. 4, pp. 96–103, Apr. 2002.
[39] J. Erfanian, S. Pasupathy, and G. Gulak, "Reduced complexity symbol detectors with parallel structure for ISI channels," *IEEE Trans. Commun.*, vol. 42, no. 234, pp. 1661–1671, Apr. 1994.
[40] Y. Wu and B. D. Woerner, "The influence of quantization and fixed point arithmetic upon the BER performance of turbo codes," in *Proc. IEEE 49th Veh. Technol. Conf.*, vol. 2. Houston, TX, USA, Jul. 1999, pp. 1683–1687.
[41] J. Vogt and A. Finger, "Improving the max-log-MAP turbo decoder," *Electron. Lett.*, vol. 36, no. 23, pp. 1937–1939, Nov. 2000.
[42] L. G. Amaru, M. Martina, and G. Masera, "High speed architectures for finding the first two maximum/minimum values," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 12, pp. 2342–2346, Dec. 2012.
[43] *TSMC 65 nm CLN65LP Ultra High Density Standard Cell Library Databook*, ARM Embedded Technol. Pvt. Ltd., Bengaluru, India, 2010.
[44] A. Tangel and K. Choi, "'The CMOS inverter' as a comparator in ADC designs," *Analog Integr. Circuits Signal Process.*, vol. 39, no. 2, pp. 147–155, May 2004.
[45] *PrimeTime User Guide*, Synopsys, Mountain View, CA, USA, Dec. 2013.
[46] M. Grymel and S. B. Furber, "A novel programmable parallel CRC circuit," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 10, pp. 1898–1902, Oct. 2011.
[47] G. Wang, H. Shen, Y. Sun, J. R. Cavallaro, A. Vosoughi, and Y. Guo, "Parallel interleaver design for a high throughput HSPA+/LTE multi-standard turbo decoder," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 5, pp. 1376–1389, May 2014.
[48] 3GPP, "Overview of 3GPP release 14 V0.0.1," 3GPP Mobile Competence Centre c/o ETSI, Sophia Antipolis, France, Tech. Rep. 14, Sep. 2014.



AN LI received the B.Eng. (Hons.) degree in electronics engineering from the University of Southampton, in 2011, and the M.Phil. degree from the University of Cambridge, in 2012. He is currently pursuing the Ph.D. degree with the Wireless Communication Research Group, University of Southampton. His research interests include parallel turbo decoding algorithms and their implementations upon VLSI, FPGA, and GPGPU.

LUPING XIANG received the B.Eng. (Hons.) degree from Xiamen University, China, in 2015. He is currently pursuing the Ph.D. degree with the Southampton Wireless Group, University of Southampton. His research interests include fully parallel turbo coding algorithm.

TAIHAI CHEN received the B.Eng. (Hons.) degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2009, and the M.Sc. (Hons.) and Ph.D. degrees from the University of Southampton, Southampton, U.K., in 2010 and 2015, respectively. He is currently a Post-Doctoral Researcher with the Southampton Wireless Group, University of Southampton. His research interests include turbo coding VLSI design, biomedical signal processing, pattern recognition, machine learning, and remote healthcare.



ROBERT G. MAUNDER (S'03–M'08–SM'12) received the B.Eng. (Hons.) degree in electronics engineering in 2003, and the Ph.D. degree in wireless communications in 2007. He has studied with the Electronics and Computer Science Department, University of Southampton, U.K., since 2000. He became a Lecturer in 2007, and an Associate Professor in 2013. His research interests include joint source/channel coding, iterative decoding, irregular coding, and modulation techniques.

BASHIR M. AL-HASHIMI (M'99–SM'01–F'09) is a Professor of Computer Engineering and the Dean of the Faculty of Physical Sciences and Engineering with the University of Southampton, U.K. He is an ARM Professor of Computer Engineering and the Co-Director of the ARM-ECS Research Centre. His research interests include methods, algorithms, and design automation tools for energy efficient of embedded computing systems. He has authored over 300 technical papers, authored or co-authored five books, and has graduated 31 Ph.D. students.

LAJOS HANZO (F'04) holds the Chair of Telecommunications with Southampton University, U.K. He has co-authored over 1500 IEEE Xplore entries, 20 IEEE Press & Wiley books, graduated over 100 Ph.D. students, and has an H-index of 58 and over 23 000 citations. He is a RS Wolfson Fellow.