1.5 Gbit/s FPGA Implementation of a Fully-Parallel Turbo Decoder Designed for Mission-Critical Machine-Type Communication Applications

An Li[1], Peter Hailes[1], Robert G. Maunder[1], Bashir M. Al-Hashimi[2], and Lajos Hanzo[1]

[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.*

***Abstract*—In wireless communication schemes, turbo codes facilitate near-capacity *transmission* throughputs by achieving reliable forward error correction. However, owing to the serial data dependencies imposed by the underlying Logarithmic Bahl-Cocke-Jelinek-Raviv (Log-BCJR) algorithm, the limited *processing* throughputs of conventional turbo decoder implementations impose a severe bottleneck upon the *overall* throughputs of real-time wireless communication schemes. Motivated by this, we recently proposed a Fully Parallel Turbo Decoder (FPTD) algorithm, which eliminates these serial data dependencies, allowing parallel processing and hence offering a significantly higher processing throughput. In this paper, we propose a novel resource-efficient version of the FPTD algorithm, which reduces its computational resource requirement by 50%, which enhancing its suitability for Field-Programmable Gate Array (FPGA) implementations. We propose a model FPGA implementation. When using a Stratix IV FPGA, the proposed FPTD FPGA implementation achieves an average throughput of 1.53 Gbit/s and an average latency of 0.56 μs, when decoding frames comprising N=720 bits. These are respectively 13.2 times and 11.1 times superior to those of the state-of-the-art FPGA implementation of the Log-BCJR Long-Term Evolution (LTE) turbo decoder, when decoding frames of the same frame length at the same error correction capability. Furthermore, our proposed FPTD FPGA implementation achieves a normalized resource usage of 0.42 $\frac{\text{kALUTs}}{\text{Mbit/s}}$, which is 5.2 times superior to that of the benchmarker decoder. Furthermore, when decoding the shortest N=40-bit LTE frames, the proposed FPTD FPGA implementation achieves an average throughput of 442 Mbit/s and an average latency of 0.18 μs, which are respectively 21.1 times and 10.6 times superior to those of the benchmarker decoder. In this case, the normalized resource usage of 0.08 $\frac{\text{kALUTs}}{\text{Mbit/s}}$ is 146.4 times superior to that of the benchmarker decoder.**

***Index Terms*—Fully-parallel turbo decoder, FPGA, LTE, turbo decoding**

The financial support of the EPSRC, Swindon UK under the grants EP/J015520/1, EP/L010550/1 and the TSB, Swindon UK under the grant TS/L009390/1 is gratefully acknowledged. The research data for this paper is available at http://dx.doi.org/10.5258/SOTON/396816.

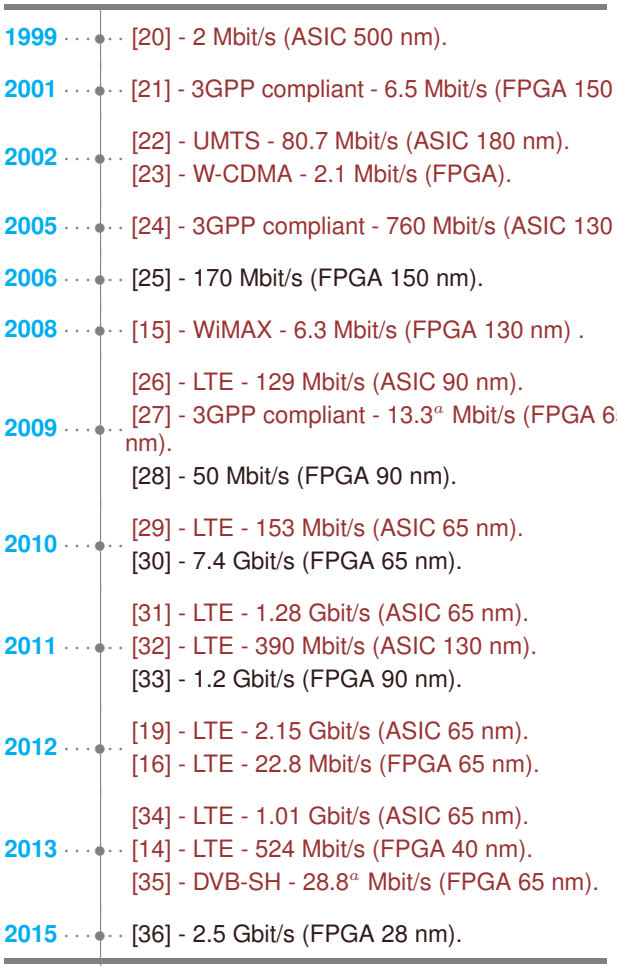
NOMENCLATURE

3GPP 3rd Generation Partnership Project
ALUT Adaptive Look-Up Table
ARA-LDPC Accumulate-Repeat-Accumulate Low-Density Parity-Check
ASIC Application-Specific Integrated Circuit
AWGN Additive White Gaussian Noise
BCJR Bahl-Cocke-Jelinek-Raviv
BER Bit Error Rate
CCMC Continuous-Input Continuous-Output Memoryless Channel
DVB-SH Digital Video Broadcasting - Satellite services to Handhelds
FPGA Field-Programmable Gate Array
LDPC Low-Density Parity-Check
LLR Log-Likelihood Ratio
Log-BCJR Logarithmic Bahl-Cocke-Jelinek-Raviv
LTE Long-Term Evolution
LTE-A Long-Term Evolution Advanced
MCMTC Mission-Critical Machine-Type Communication
PEG-LDPC Progressive-Edge-Growth Low-Density Parity-Check
UMTS Universal Mobile Telecommunications System
VLSI Very-Large-Scale Integration
WiFi Wireless Fidelity
WiMAX Worldwide Interoperability for Microwave Access

I. Introduction

Channel coding plays an important role in the physical layer of wireless communications systems, facilitating the correction of transmission errors imposed by hostile channels. In particular, state-of-the-art iterative channel codes such as Low-Density Parity-Check (LDPC) and turbo codes [1]–[5] are capable of facilitating reliable communication at near-capacity *transmission* throughputs, leading to widespread employment by state-of-the-art mobile telephony standards, such as WiMAX [6] and LTE [7]. However, the *processing* throughputs of the iterative channel decoder often imposes a bottleneck upon the *overall* throughput of real-time wireless communication schemes. Likewise, the processing latency of the iterative channel decoder typically dominates the overall physical layer latency. This is of particular concern in the emerging Mission-Critical Machine-Type Communication (MCMTC) applications of next generation wireless systems [8], [9]. More specifically, these applications will require the reliable transmission of relatively short emergency and control message frames comprising as few as dozens or hundreds of bits with a Gbit/s throughput and an ultra-low latency, which is on the order of microseconds [9]. These short message frames motivate the use of turbo codes in Mission-Critical Machine-Type Communication (MCMTC) applications, since they offer superior error correction capability for short frames than LDPC codes, while additionally performing also well for long frames [10]–[12].

As one may expect, initial systems designed for these emerging MCMTC applications are likely to be employed in relatively small numbers of machines or vehicles, where the cost is of greater concern than size, weight and power, motivating the employment of (Field-Programmable Gate Arrays) FPGAs, rather than (Application-Specific Integrated Circuits) ASICs for the implementation of turbo decoding. However, the throughput and latency requirements of MCMTC applications are particularly challenging to fulfill in FPGA implementations of turbo decoders. More specifically, while Figure 1 shows that the FPGA implementation of LDPC decoders has received a significant amount of attention over the past two decades [13], only [14]–[16] have proposed FPGA implementations of turbo decoders. Owing to their natural suitability to parallel processing, Figure 1 shows that LDPC decoders have previously offered significantly higher processing throughputs than turbo decoders, as well as significantly lower processing latencies. Indeed, the state-of-the-art FPGA-based LTE turbo decoder achieves a peak processing throughput of 524 Mbit/s, when processing the longest 6144-bit message frames. However, this drops to 62 Mbit/s when processing 512-bit frames [14], which are more typical of MCMTC applications. This may be attributed to the state-of-the-art turbo decoder implementations reliance on the iterative operation of two Logarithmic Bahl-Cock-Jelinek-Raviv (Log-BCJR) decoders [17], [18]. 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 [19] and five to eight iterations per message frame.



[a] Throughput that would be achieved by performing six iterations, rather than only a single half-iteration.

Fig. 1: Selected ASIC and FPGA implementations of Log-BCJR turbo decoders (shown in red) and LDPC decoders (shown in black) for different communication standards.

Motivated by achieving Gbit/s turbo decoding processing throughputs and ultra-low processing latencies, we previously proposed a novel floating-point Fully-Parallel Turbo Decoder (FPTD) algorithm [38]. 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 [38]. This facilitates

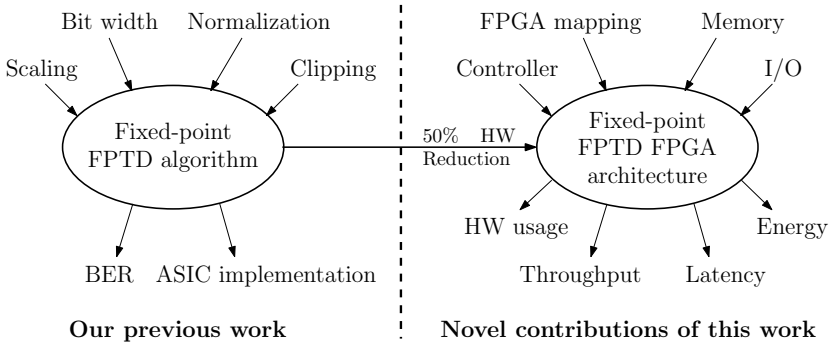


Fig. 2: The novel contributions of this work, compared with our previous work of [37].

fully-parallel processing, allowing each half-iteration to use only a single clock cycle, although this is achieved at the cost of the FPTD typically requiring seven times as many iterations for achieving the same error correction capability as the state-of-the-art turbo decoding algorithm. Despite this, our previous contribution of [37] shows that a fixed-point ASIC implementation of this FPTD algorithm is capable of achieving a processing throughput as high as 21.9 Gbit/s, which is 17.1 times superior to the state-of-the-art Log-BCJR based turbo decoder of [19], when implemented using the same TSMC 65 nm technology and decoding the longest $N = 6144$-bit LTE frames.

Against this background, this paper proposes a novel fixed-point FPTD architecture, which implements the fixed-point FPTD algorithm using 50% less hardware resources per message bit compared to our previous fixed-point FPTD architecture [37], as shown in Figure 2. We also propose a novel FPGA implementation for the proposed FPTD architecture, which is suitable for MCMTC applications. The main experimental results of this work are listed as follows.

- The proposed Stratix IV FPGA implementation of the proposed FPTD architecture achieves an average throughput of 1.5 Gbit/s and an average latency of 0.56 μs, when decoding frames comprising $N = 720$ bits, as may be found in MCMTC applications [8], [9]. These are respectively 13.2 times and 11.1 times superior to those of the state-of-the-art FPGA implementation of the LTE turbo decoder [14] based on Log-BCJR algorithm, when decoding the same frame length of $N = 720$.
- The proposed FPGA implementation of the 720-bit FPTD has a normalized resource usage of 0.42 $\frac{\text{kALUTs}}{\text{Mbit/s}}$, where the Adaptive Look-Up Tables (ALUTs) are the fundamental programmable hardware resources adopted by the FPGA. This is 5.2 times superior to the 22 $\frac{\text{kALUTs}}{\text{Mbit/s}}$ that is obtained for the benchmarker decoder of [14]. Likewise, it is 1.3 times superior to the 0.55 $\frac{\text{kALUTs}}{\text{Mbit/s}}$ recorded for a specific version of the benchmarker optimized for frame lengths N in the range spanning from 512 to 1024 bits.
- When decoding the shortest $N = 40$-bit LTE frame, the proposed FPTD achieves an average throughput of 442 Mbit/s and an average latency of 0.18 μs, which are respectively 21.1 times and 10.6 times superior to the benchmarker decoder of [14]. In this case, the normalized resource usage is 146.4 times lower and 19 times lower than that of the benchmarker decoder of [14] and a specific version optimized for frame lengths N in the range of 40 to 512 bits, respectively.

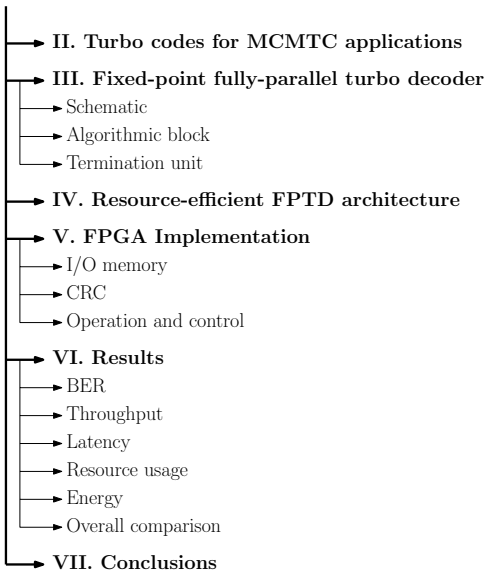


Fig. 3: The paper structure

The rest of the paper of Figure 3 is organized as follows. In Section II, we discuss the motivation for using turbo codes in MCMTC applications. In Section III, we offer background discussions on our previously proposed fixed-point FPTD algorithm [37], which was designed for VLSI applications. In Section IV, we propose a

novel fixed-point FPTD architecture which benefits from a 50% lower hardware resource requirement than our previous architecture [37]. In Section V, we detail the FPGA implementation of our novel resource-efficient FPTD architecture, including its top-level schematic as well as its input/output memory and Cyclic Redundancy Check (CRC) circuit. In Section VI, we characterize the proposed FPGA implementation of our resource-efficient fixed-point FPTD, in terms of its processing throughput, processing latency, hardware resource usage and energy consumption, where the first three items are compared to those of the state-of-the-art Log-BCJR turbo decoder FPGA implementations. Finally, we offer our conclusions in Section VII.

II. Turbo codes for MCMTC applications

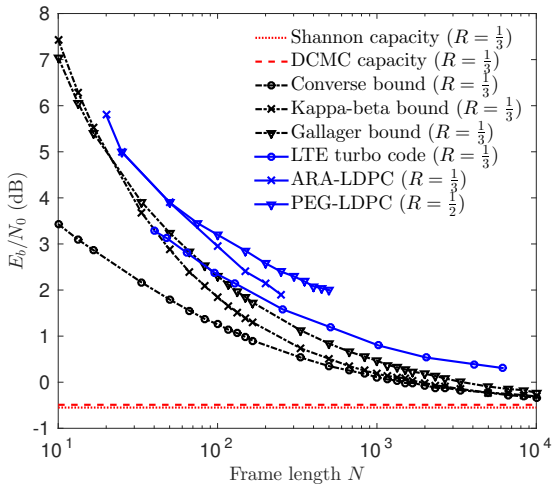


Fig. 4: The E_b/N_0 values where the $R = 1/3$ LTE turbo code achieves a target FER of 10^{-3} as a function of the message frame length N, compared with the channel capacities, several bounds and the LDPC codes of [12], [39], for the case of BPSK transmission over an AWGN channel.

As described in Section I, turbo codes are attractive in applications such as MCMTC, since they offer a strong error correction capability even for short message frames. More specifically, Figure 4 shows the E_b/N_0 values, where a Frame Error Ratio (FER) of 10^{-3} is achieved by the LTE turbo code for different message frame lengths N in the range of 40 to 6144 bits supported by LTE. Note that these results correspond to the case of using an LTE turbo coding rate of $R = 1/3$ combined with Binary Phase Shift Keying (BPSK) modulation for transmission over an AWGN channel, and using a sufficiently high number of turbo decoding iterations for achieving iterative decoding convergence. For comparison with the LTE turbo code, Figure 4 also shows the E_b/N_0 values, where an FER of 10^{-3} is achieved by an $R = 1/3$-rate Accumulate-Repeat-Accumulate (ARA)-LDPC and an $R = 1/2$-rate Progressive-Edge-Growth (PEG)-LDPC, as considered using similar analysis in [39] and [12], respectively. It may be seen that the LTE turbo code outperforms the ARA-LDPC and PEG-LDPC codes for all frame lengths, where the maximum gap of approximately 1 dB is achieved, when the frame length is $N = 40$ bits. Note that according to [12], the performance of the LDPC code used in WiMAX is very similar to that of the PEG-LDPC code shown in Figure 4, but the frame lengths N supported by the WiMAX LDPC are constrained to the range of 288 to 1152 bits for the code rate of $R = 1/2$ [40].

Additionally, Figure 4 shows a selection of capacity bounds, which provide a wider context for the performance achieved by the turbo and LDPC codes considered. More specifically, the Continuous-Input Continuous-Output Memoryless Channel (CCMC) Shannon capacity [41] and the modulation-specific Discrete-input Continuous-output Memoryless Channel (DCMC) capacity bound [42] for the combination of $R = 1/3$ channel coding, BPSK modulation and AWGN channel are represented by the pair of horizontal lines at $E_b/N_0 = -0.55$ dB and $E_b/N_0 = -0.49$ dB, respectively. However, the Shannon capacity and DCMC capacity provide bounds that only apply for infinitely long frame lengths, which therefore do not offer an accurate prediction of the achievable error correction capability for practical channel codes, having short message frame lengths of the order of dozens or hundreds of bits. Motivated by this, the converse bound [43] of Figure 4 represents a lower bound on the achievable error correction capability of practical channel codes as a function of the message frame length N, offering a better estimation for short message frames than the DCMC and Shannon capacity bounds. Furthermore, the Kappa-beta and Gallager bounds [43]–[45] of Figure 4 offer further refinements of the estimated error correction capability that is achievable by practical channel codes. Compared to these refined bounds, the LTE turbo code can be seen in Figure 4 to offer near-optimal error correction capability for both short message frames and long message frames, which motivates the employment of turbo codes in MCMTC applications.

III. Fixed-point fully-parallel turbo decoder

The floating-point FPTD algorithm was originally proposed in [38]. Following this, in [37] we proposed a fixed-point version of the FPTD algorithm, which was optimized for the LTE turbo code and was implemented as an ASIC. In this section, we briefly summarize our previously proposed fixed-point LTE FPTD algorithm

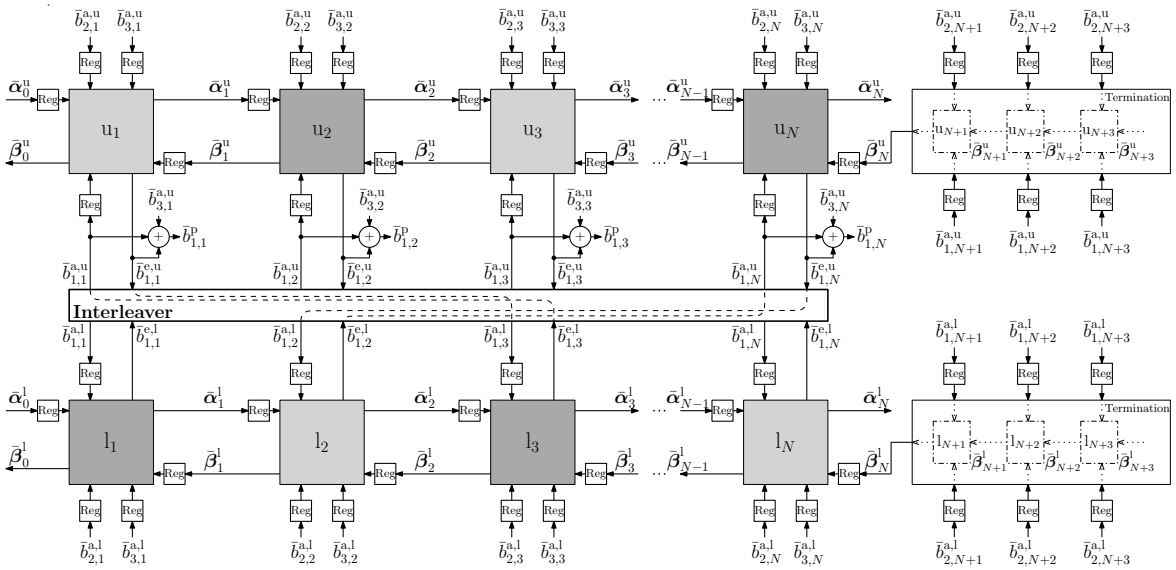


Fig. 5: Schematic of the FPTD algorithm of [38].

as follows. In Section III-A, we discuss the top-level operation of the FPTD algorithm, using the schematics of Figures 5 and 6. In Sections III-B and III-C respectively, we summarize the fixed-point algorithmic blocks of Figure 8 and the termination unit of Figure 11, which may be employed for implementing the FPTD algorithm of Figure 5.

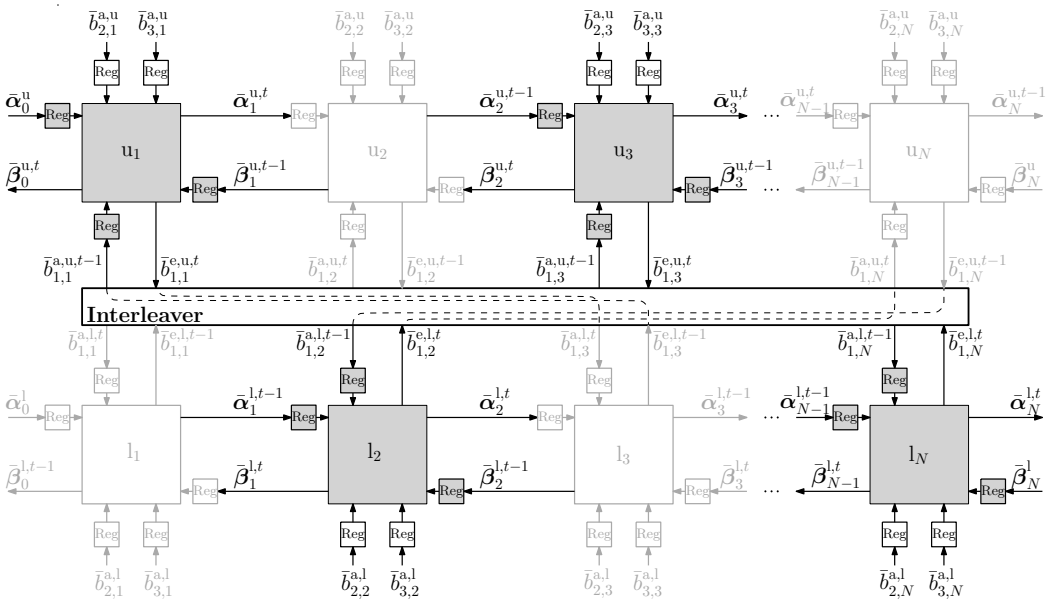
A. Schematic

Figure 5 shows the schematic of the FPTD algorithm proposed in [38]. When decoding N-bit message frames, the FPTD algorithm comprises two rows of N identical algorithmic blocks, where the blocks of the upper and lower rows are labeled as $\{\mathrm{u}_1, \mathrm{u}_2 \ldots, \mathrm{u}_N\}$ and $\{\mathrm{l}_1, \mathrm{l}_2 \ldots, \mathrm{l}_N\}$, respectively. 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. A termination unit comprising unshaded algorithmic blocks is appended to the tail of each row, in order to comply with the LTE termination mechanism [7]. As in the Log-BCJR algorithm, the FPTD algorithm operates on the basis of Logarithmic Likelihood Ratios (LLRs) [46], where each LLR of

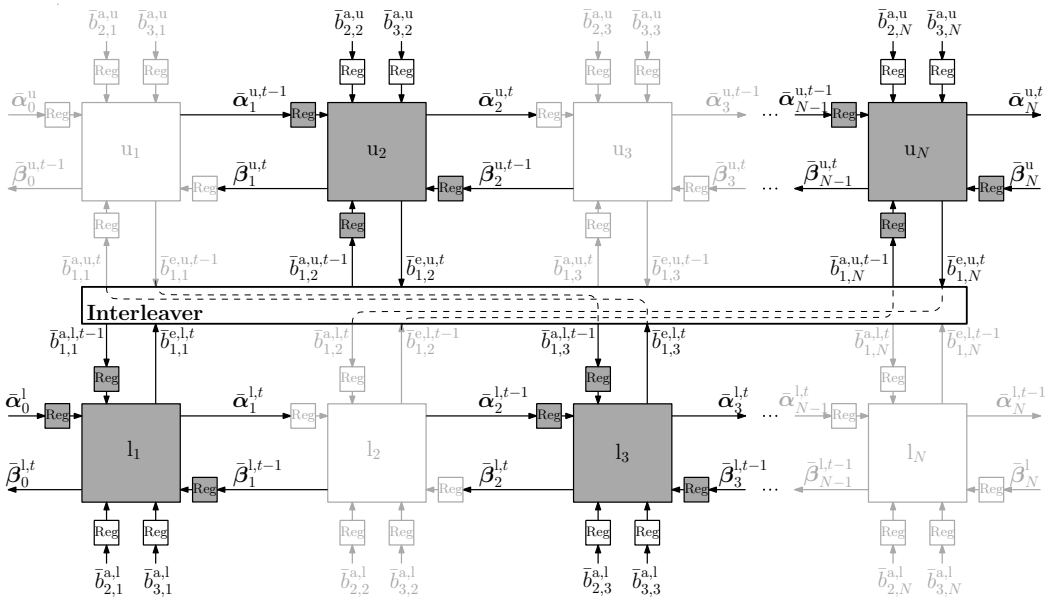
$$\bar{b} = \ln \frac{\Pr(b=1)}{\Pr(b=0)} \tag{1}$$

conveys soft information pertaining to the corresponding bit b within the turbo encoder. Note that in the rest of this paper, the superscripts 'u' and 'l' seen in the notation of Figure 5 are used only when necessary for explicitly distinguishing the upper and lower components of the turbo code, but they are omitted in discussions that apply equally to both. When decoding message 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 may adopt one of 188 values in the range of [40, 6144] in the LTE turbo code. These *a priori* LLRs are provided by the demodulator and are stored in the corresponding registers of Figure 5 throughout the decoding processing of the corresponding frame. Note that the set of lower systematic LLRs $[\bar{b}^{\mathrm{a,l}}_{3,k}]^{N}_{k=1}$ is 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 and as in the conventional Log-BCJR turbo decoder.

Like the conventional Log-BCJR turbo decoder, the FPTD algorithm relies on iterative operation. However, rather than requiring 64 to 192 clock cycles per iteration, each iteration of the FPTD algorithm comprises only two clock cycles, which are referred to as half-iterations. More specifically, Figure 6(a) shows that the first half-iteration of the FPTD algorithm corresponds to the simultaneous operation of the lightly-shaded algorithmic blocks shown in Figure 5 within a single clock cycle. These lightly-shaded blocks comprise the algorithmic blocks in the upper row having odd indices $\{\mathrm{u}_1, \mathrm{u}_3, \mathrm{u}_5, \ldots\}$ and the even-indexed algorithmic blocks in the lower row $\{\mathrm{l}_2, \mathrm{l}_4, \mathrm{l}_6, \ldots\}$. By contrast, Figure 6(b)



(a) Lightly-shaded algorithmic blocks are operated concurrently in every odd clock cycle, corresponding to every first half-iteration.



(b) Darkly-shaded algorithmic blocks are operated concurrently in every even clock cycle, corresponding to every second half-iteration.

Fig. 6: Schematics of the FPTD algorithm, in which the lightly-shaded algorithmic blocks shown in (a) and the darkly-shaded algorithmic blocks shown in (b) are operated alternately.

shows that 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 5. During the t^{th} clock cycles of the decoding process, the $k^{\text{th}} \in [1, N]$ algorithmic block processes the *a priori* LLRs $\bar{b}^{\text{a},t-1}_{1,k}$, $\bar{b}^{\text{a}}_{2,k}$ and $\bar{b}^{\text{a}}_{3,k}$. Here, $\bar{b}^{\text{a},t-1}_{1,k}$ was generated in the $(t-1)^{\text{st}}$ clock cycle by interleaving the appropriate extrinsic message LLR provided by an algorithmic block in the other row, where $\bar{b}^{\text{a,l},t-1}_{1,k} = \bar{b}^{\text{e,u},t-1}_{1,\pi(k)}$ and $\bar{b}^{\text{a,u},t-1}_{1,\pi(k)} = \bar{b}^{\text{e,l},t-1}_{1,k}$. In addition to the *a priori* LLRs $\bar{b}^{\text{a},t-1}_{1,k}$, $\bar{b}^{\text{a}}_{2,k}$ and $\bar{b}^{\text{a}}_{3,k}$, the algorithmic block also consumes a set of M forward-oriented state metrics $\bar{\boldsymbol{\alpha}}^{t-1}_{k-1} = [\bar{\alpha}^{t-1}_{k-1}(S_{k-1})]^{M-1}_{S_{k-1}=0}$ and a set of M backward-oriented state metrics $\bar{\boldsymbol{\beta}}^{t-1}_{k} = [\bar{\beta}^{t-1}_{k}(S_k)]^{M-1}_{S_k=0}$, where the LTE turbo code employs $M = 8$ states. For algorithmic blocks having an index of $k \in [2, N]$, $\bar{\alpha}^{t-1}_{k-1}$ is generated in the previous $(t-1)^{\text{st}}$ clock cycle by the preceding $(k-1)^{\text{st}}$ algorithmic block in the same row. Likewise, for algorithmic blocks having an index of $k \in [1, N-1]$, $\bar{\beta}^{t-1}_{k}$ is generated in the

$$\bar{\gamma}_k^t(S_{k-1}, S_k) = b_1(S_{k-1}, S_k) \cdot \bar{b}_{1,k}^{\mathrm{a},t-1} + b_2(S_{k-1}, S_k) \cdot \bar{b}_{2,k}^{\mathrm{a}} + b_3(S_{k-1}, S_k) \cdot \bar{b}_{3,k}^{\mathrm{a}} \tag{2}$$

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

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

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

previous $(t-1)^{\mathrm{st}}$ clock cycle by the following $(k+1)^{\mathrm{st}}$ algorithmic block in the same row. As shown in Figure 5, registers are required for storing $[\bar{b}_{1,k}^{\mathrm{a},t-1}]_{k=1}^{N}$, $[\bar{\boldsymbol{\alpha}}_{k-1}^{t-1}]_{k=2}^{N}$ and $[\bar{\boldsymbol{\beta}}_k^{t-1}]_{k=1}^{N-1}$ between the consecutive clock cycles, since they are generated by connected algorithmic blocks in the clock cycle before they are used.

Since the *a priori* message LLRs $[\bar{b}_{1,k}^{\mathrm{a},t-1}]_{k=1}^{N}$ are unavailable in the initial first half-iteration, they are initialized as $\bar{b}_{1,k}^{\mathrm{a},t-1} = 0$, 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 in the initial first half-iteration, hence these are also initialized as $\bar{\boldsymbol{\alpha}}_{k-1}^{t-1} = [0, 0, 0, \ldots, 0]$ for the algorithmic blocks having indices of $k \in [2, N]$ and as $\bar{\boldsymbol{\beta}}_k^{t-1} = [0, 0, 0, \ldots, 0]$ 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 $\bar{\boldsymbol{\alpha}}_0 = [0, -\infty, -\infty, ..., -\infty]$ 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, constant values are also used throughout all decoding iterations for the backward state metrics $\bar{\boldsymbol{\beta}}_N$, but these values are obtained using a termination unit, which is detailed in Section III-C. Following the completion of each half-iteration during the t^{th} clock cycle, a set of N *a posteriori* LLRs $[\bar{b}_{1,k}^{\mathrm{p},t}]_{k=1}^{N}$ can be obtained as $\bar{b}_{1,k}^{\mathrm{p},t} = \bar{b}_{1,k}^{\mathrm{e,u},t} + \bar{b}_{1,k}^{\mathrm{a,u},t-1} + \bar{b}_{3,k}^{\mathrm{a,u}}$ for the algorithmic blocks having odd indices k and as $\bar{b}_{1,k}^{\mathrm{p},t} = \bar{b}_{1,k}^{\mathrm{e,u},t-1} + \bar{b}_{1,k}^{\mathrm{a,u},t} + \bar{b}_{3,k}^{\mathrm{a,u}}$ for the blocks having even indices k. Likewise, the hard decision value for each bit may be obtained according to the binary test $\bar{b}_{1,k}^{\mathrm{p},t} > 0$.

B. Algorithmic block

Within each of the clock cycles during which the k^{th} algorithmic block in either row of Figure 5 is activated, it accepts inputs and generates outputs according to (2), (3), (4) and (5). Here, (2) is used to obtain a metric $\bar{\gamma}_k^t(S_{k-1}, S_k)$ for each possible transition between a pair of states S_{k-1} and S_k, as shown in the LTE state transition diagram of Figure 7. Note that each transition implies a particular binary value for the corresponding message bit $b_1(S_{k-1}, S_k)$, parity bit $b_2(S_{k-1}, S_k)$ and systematic bit $b_3(S_{k-1}, S_k)$, where 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) is employed to obtain the vectors of state metrics $\bar{\boldsymbol{\alpha}}_k^t$ and $\bar{\boldsymbol{\beta}}_{k-1}^t$, respectively. Here, $c(S_{k-1}, S_k)$ adopts a binary value of 1, if a transition is possible between the states S_{k-1} and S_k in the state transition diagram of Figure 7. Furthermore, the Jacobian logarithm [18], [47] is defined as

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

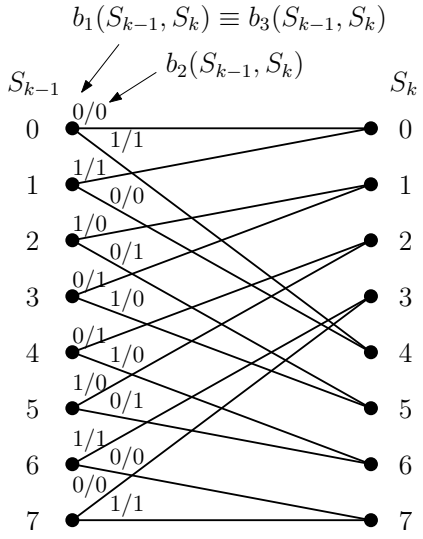


Fig. 7: State transition diagram of the LTE turbo code.

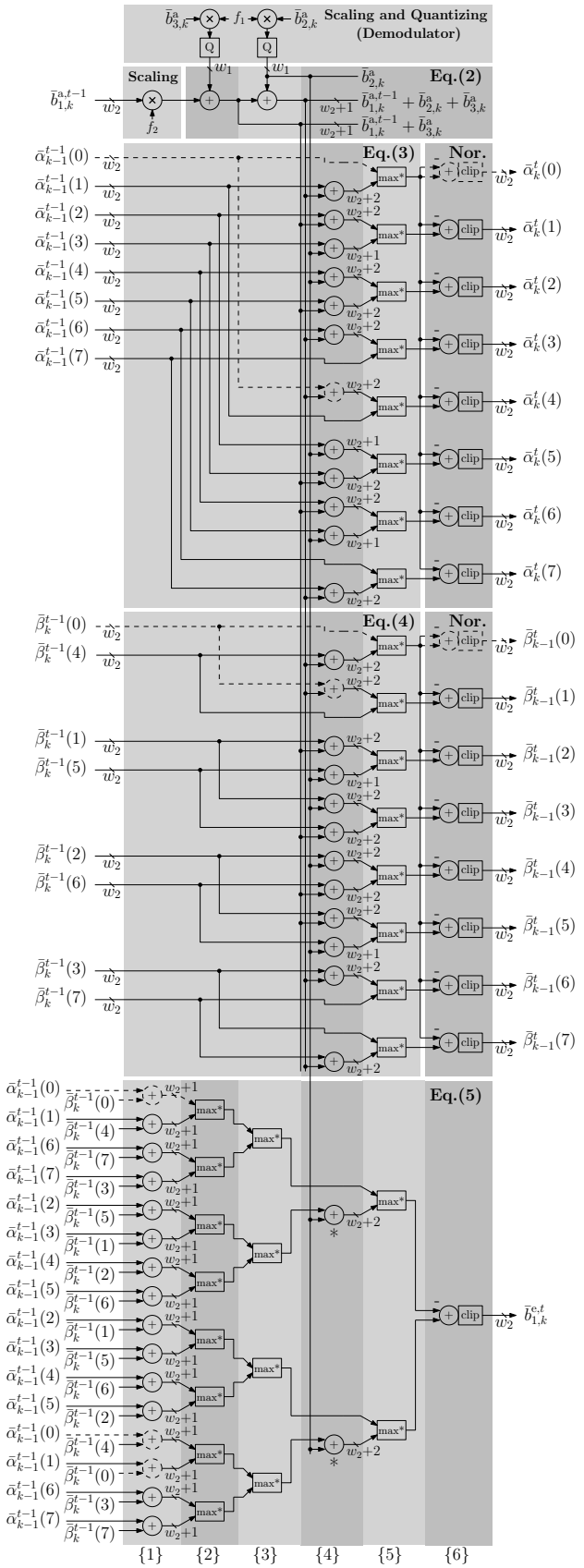


Fig. 8: The datapath for the k^{th} processing element of the fixed-point FPTD algorithm for the case of the LTE turbo code. The six datapath stages are distinguished by the dark/light shading and are indexed as shown in the curly brackets.

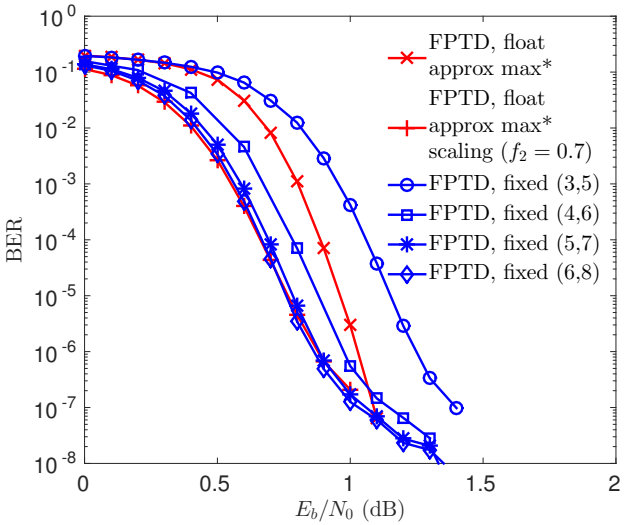


Fig. 9: BER performance of the fixed-point FPTD using the approximate max* operation of (7), message LLR scaling ($f_2 = 0.75$), *state-zero* state metric normalization and various bit-widths (w_1, w_2). The BER performance is compared to that of the floating-point FPTD using the approximate max* operation 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.

However, its approximated version of

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

may be employed, for reducing the computational complexity of the FPTD algorithm, in analogy with the Max-Log BCJR algorithm [18], [47]. Finally, (5) is employed for obtaining the extrinsic LLR $\bar{b}^{\text{e},t}_{1,k}$, where the associative property of the max* operator may be involved for extending (6) and (7) to more than two operands.

The processing element of Figure 8 is designed for performing all operations of an algorithm block, within a single clock cycle, as required by the FPTD algorithm. During this single clock cycle, the signals propagate through six datapath stages, which impose similar propagation delays. More explicitly, these datapath stages perform addition, subtraction and maximum calculations, which can all be efficiently implemented at similar complexities using two's complement arithmetic. In particular, the variables of (2) to (5) are represented using two's complement fixed point numbers, having the bit-widths of (w_1, w_2), where the bit-widths of $(w_1, w_2) = (4, 6)$ offer an attractive trade off between strong BER performance and low computational complexity, as shown in Figure 9. More specifically, the bit-width of $w_1 = 4$ is employed for the *a priori* parity LLR $\bar{b}^{\text{a}}_{2,k}$ and systematic LLR $\bar{b}^{\text{a}}_{3,k}$, as recommended in [37]. As shown at the top

of Figure 8, 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, where it is assumed that a quantizer is employed for converting the real-valued LLRs to fixed-point LLRs. In order to prevent a significant BER performance degradation owing to quantization distortion, it is assumed that the demodulator applies noise-dependent scaling [48] to both the *a priori* LLRs $\bar{b}^{\mathrm{a}}_{2,k}$ and $\bar{b}^{\mathrm{a}}_{3,k}$. 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 the E_b/N_0 is expressed in dB, $v = 2^{w_1-1}$ 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 [48]. In contrast to the channel LLRs, the bit-width of $w_2 = w_1 + 2 = 6$ is employed for the *a priori* and extrinsic message LLRs $\bar{b}^{\mathrm{a},t-1}_{1,k}$ and $\bar{b}^{\mathrm{e},t}_{1,k}$, as well as for the *a priori* and extrinsic state metrics $\bar{\boldsymbol{\alpha}}^{t-1}_{k-1}$, $\bar{\boldsymbol{\alpha}}^{t}_{k}$, $\bar{\boldsymbol{\beta}}^{t-1}_{k}$ and $\bar{\boldsymbol{\beta}}^{t}_{k-1}$. A higher bit-width of $w_2 > w_1$ is required, because the magnitudes of $\bar{b}^{\mathrm{a},t-1}_{1,k}$, $\bar{b}^{\mathrm{e},t}_{1,k}$, $\bar{\boldsymbol{\alpha}}^{t-1}_{k-1}$, $\bar{\boldsymbol{\alpha}}^{t}_{k}$, $\bar{\boldsymbol{\beta}}^{t-1}_{k}$ and $\bar{\boldsymbol{\beta}}^{t}_{k-1}$ tend to grow in successive decoding iterations, 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. Furthermore, in order to avoid overflow, up to $w_2 + 2 = 8$ bits are used for the intermediate variables within the processing element of Figure 8. However, the output variables $\bar{b}^{\mathrm{e},t}_{1,k}$, $\bar{\boldsymbol{\alpha}}^{t}_{k}$ and $\bar{\boldsymbol{\beta}}^{t}_{k-1}$ are clipped to bit-widths of w_2 before they are output, as shown in Figure 8.

In addition to the noise-dependent scaling applied to $\bar{b}^{\mathrm{a}}_{2,k}$ and $\bar{b}^{\mathrm{a}}_{3,k}$ by the demodulator, the BER performance of the FPTD algorithm can be improved by scaling the *a priori* message LLR $\bar{b}^{\mathrm{a},t-1}_{1,k}$, in order to counteract the degradation imposed by the approximate max* operation of (7) [37]. While a scaling factor of $f_2 = 0.7$ is beneficial for the floating-point FPTD, our results of Figure 9 show that the fixed-point FPTD benefits from applying a scaling factor of $f_2 = 0.75$, which also facilitates a low-complexity hardware implementation. More specifically, by exploiting the two's complement multiplication arithmetic illustrated in Figure 10, the message LLR scaling factor of 0.75 may be applied to the *a priori* LLR $\bar{b}^{\mathrm{a},t-1}_{1,k}$ using two steps. In the first step, a 2-bit sign-extended version of $\bar{b}^{\mathrm{a},t-1}_{1,k}$ is added to a replica of itself that has been shifted to the left by one bit position, according to $\bar{b}^{\mathrm{a},t-1}_{1,k} + (\bar{b}^{\mathrm{a},t-1}_{1,k} \ll 1)$. Then in a second step, a *floor* truncation [49] is applied to the two least significant bits of the result, which maintains the same bit-width of w_1 as that employed before the message LLR scaling. Here, the sign extension, bit shifting and *floor* operations can be carried out by hard-wiring, since the scaling factor of $f_2 = 0.75$ is fixed throughout the iterative decoding process. Therefore, the only hardware required for message LLR scaling is an adder, which occupies only the first datapath stage of Figure 8.

			1	0	0	1	0	1 $_\triangle$	-27
$\times$						0 $_\triangle$	1	1	0.75
Sign Extension	(1	1)	1	0	0	1	0	1	
$+$	(1)	1	0	0	1	0	1	0	$<<1$
Truncate	1	0	1	0	1	1 $_\triangle$	1	1	-20.25
	1	0	1	0	1	1 $_\triangle$			-21

Fig. 10: An example of two's complement multiplication, where the multiplicand is an integer and the multiplier is 0.75. The *floor* truncation is applied to the product after the decimal point ($\triangle$).

As the iterative decoding process proceeds, the values of the extrinsic state metrics $\bar{\boldsymbol{\alpha}}^{t}_{k}$ and $\bar{\boldsymbol{\beta}}^{t}_{k-1}$ can grow without upper bound [50]. In order to prevent any potential BER error floors that may be caused by saturation or overflow, state metric normalization may be employed for reducing the magnitudes of $\bar{\boldsymbol{\alpha}}^{t}_{k}$ and $\bar{\boldsymbol{\beta}}^{t}_{k-1}$, in order to ensure that they remain within the range that is supported by their bit-width of w_2. As shown in Figure 8, *state-zero* normalization [37] is performed in the sixth datapath stage within each processing element. This is achieved by subtracting $\bar{\alpha}^{t}_{k}(0)$ and $\bar{\beta}^{t}_{k-1}(0)$ from all extrinsic forward state metrics $[\bar{\alpha}^{t}_{k}(S_k)]^{M-1}_{S_k=0}$ and all extrinsic backward state metrics $[\bar{\beta}^{t}_{k-1}(S_{k-1})]^{M-1}_{S_{k-1}=0}$, respectively [50], [51]. Note that this subtraction does not change the information conveyed by the extrinsic state metrics, since this is carried by their differences, rather than by their absolute values. After *state-zero* normalization, zero-values are guaranteed for the first extrinsic state metrics $\bar{\alpha}^{t}_{k}(0) = 0$ and $\bar{\beta}^{t}_{k-1}(0) = 0$. In our fixed-point FPTD algorithm, this allows the registers and additions involving $\bar{\alpha}^{t-1}_{k-1}(0)$ and $\bar{\beta}^{t-1}_{k}(0)$ to be simply removed, saving two w_2-bit registers and seven additions per processing element, as shown by the dotted lines in Figure 8. Furthermore, this approach guarantees a constant value of zero for one of the operands input to 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.

C. Termination unit

Each row of algorithmic blocks shown in Figure 5 is appended with a termination unit, comprising three termination blocks having indices of $(N + 1)$, $(N + 2)$ and $(N + 3)$. These termination blocks employ only

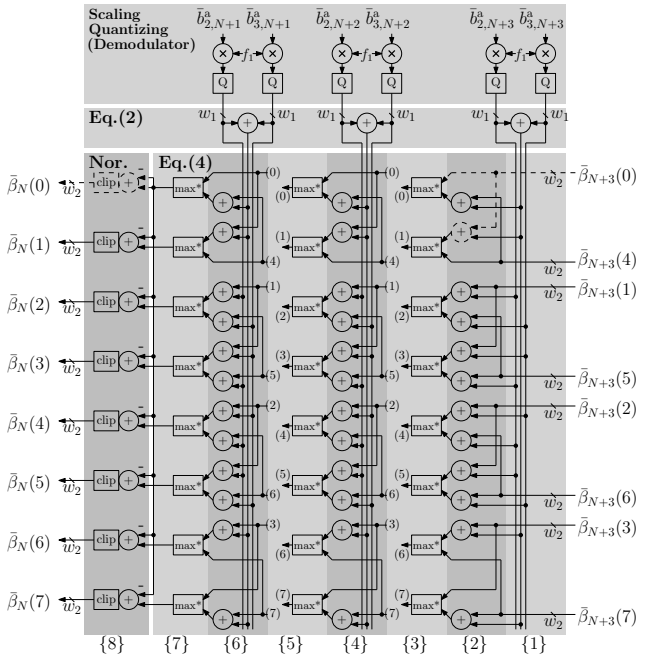


Fig. 11: The datapath for the termination unit of the proposed fixed-point FPTD for the case of the LTE turbo code. The eight datapath stages are distinguished by the dark/light shading and are indexed as shown in the curly brackets.

(2) without the term of $b_3(S_{k-1}, S_k) \cdot \bar{b}^{\mathrm{a}}_{3,k}$ and (4), operating in a backward-oriented recursion fashion for successively calculating $\bar{\boldsymbol{\beta}}_{N+2}$, $\bar{\boldsymbol{\beta}}_{N+1}$ and $\bar{\boldsymbol{\beta}}_N$. Here, we employ $\bar{\boldsymbol{\beta}}_{N+3} = [0, -\infty, -\infty, ..., -\infty]$, since the LTE termination technique guarantees $S_{N+3} = 0$. As described in Section III-A, here $-\infty$ is replaced by a negative constant having a suitably high magnitude in the fixed-point FPTD algorithm. 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 $\bar{\boldsymbol{\beta}}_N$ value can be used throughout the iterative decoding process, with no need to operate the termination unit again, as described in Section III-A.

In contrast to the processing element of Figure 8, the termination unit of Figure 11 requires eight datapath stages for implementing the three consecutive algorithmic blocks, 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-oriented state metrics $\bar{\boldsymbol{\beta}}_N$. As shown in Figure 11, 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 termination blocks in a backward recursive manner, where calculating (4) for each termination block requires two datapath stages. The final datapath stage is occupied by the above-mentioned *state-zero* normalization. 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 8, the termination unit does not dictate the critical path length of the fixed-point FPTD algorithm, which remains six datapath stages. This is because the termination units only have to be operated once before the iterative decoding process commences. Intuitively, this would imply that the termination units would impose a delay of two clock cycles before the iterative decoding process would be begun. However, in the fixed-point FPTD algorithm, we prefer to start the operation of the termination units at the same time as the iterative decoding process. In this way, the termination units do not impose a delay of two clock cycles 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 cycles. However, our experimental results demonstrate that this does not impose any BER degradation.

IV. Resource-efficient FPTD architecture

In this section, we propose a novel resource-efficient architecture for implementing the fixed-point FPTD algorithm of Section III. In contrast to the FPTD architecture of [37], the proposed design requires only N processing elements instead of $2N$ for decoding N-bit frames, therefore achieving 50% reduction in hardware resource usage. This is achieved by exploiting the odd-even operation of the FPTD algorithm, which results in only half of the algorithmic blocks being operated simultaneously, as described in Section III. The proposed area-efficient FPTD architecture employs the schematic of Figure 12, which uses the same N processing elements for alternately operating the algorithmic blocks of the first half-iteration of Figure 6(a) and those of the second half-iteration of Figure 6(b), in each pair of consecutive clock cycles. More specifically, each processing element having an odd index k performs the operation of the k^{th} algorithmic block from the upper row of Figure 5 in odd clock cycles and the k^{th} algorithmic block from the lower row in even clock cycles. By contrast, each processing element having an even index k performs the operation of the k^{th} algorithmic block from the lower row of Figure 5 in odd clock cycles and the k^{th} algorithmic block from the upper row in even clock cycles.

As shown in Figure 12, multiplexers are employed for each processing element in order to alternately select the corresponding upper *a priori* LLRs $\bar{b}^{\mathrm{a,u}}_{1,k}$, $\bar{b}^{\mathrm{a,u}}_{2,k}$ and $\bar{b}^{\mathrm{a,u}}_{3,k}$ or the corresponding lower *a priori* LLRs $\bar{b}^{\mathrm{a,l}}_{1,k}$,

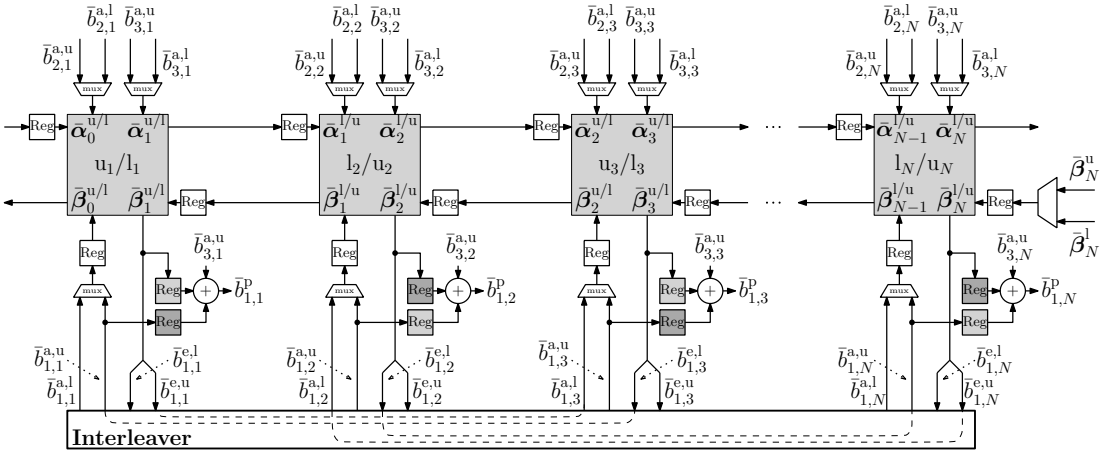


Fig. 12: Schematic of the proposed resource-efficient FPTD architecture.

$\bar{b}^{\mathrm{a,l}}_{2,k}$ and $\bar{b}^{\mathrm{a,l}}_{3,k}$, in accordance with the odd and even clock cycles. Note that the vectors of upper systematic *a priori* LLRs $\bar{\boldsymbol{b}}^{\mathrm{a,u}}_{3}$, upper parity *a priori* LLRs $\bar{\boldsymbol{b}}^{\mathrm{a,u}}_{2}$ and lower parity *a priori* LLRs $\bar{\boldsymbol{b}}^{\mathrm{a,l}}_{2}$ are stored in an input memory, while the lower systematic *a priori* LLRs $\bar{\boldsymbol{b}}^{\mathrm{a,l}}_{3}$ are obtained by interleaving the upper systematic *a priori* LLRs $\bar{\boldsymbol{b}}^{\mathrm{a,u}}_{3}$, as shown in Figure 5. Similarly, a multiplexer is required for the N^{th} processing element for alternately selecting the corresponding sets of upper backward-oriented state metrics $\bar{\boldsymbol{\beta}}^{\mathrm{u}}_{N}$ and lower backward-oriented state metrics $\bar{\boldsymbol{\beta}}^{\mathrm{l}}_{N}$, where $\bar{\boldsymbol{\beta}}^{\mathrm{u}}_{N}$ and $\bar{\boldsymbol{\beta}}^{\mathrm{l}}_{N}$ are generated using two termination units, as shown in Figure 5. Apart from this however, multiplexers are not required for the forward-oriented state metrics $\bar{\boldsymbol{\alpha}}_k$ and the backward-oriented state metrics $\bar{\boldsymbol{\beta}}_k$, since the state metrics generated by a particular processing element in a particular clock cycle will be directly processed by the neighboring processing elements in the next clock cycle, as a natural consequence of the odd-even operation of the FPTD algorithm. Furthermore, each processing element is associated with two separate routings through the interleaver. More specifically, the interleaver connects the k^{th} processing element with both the $\pi(k)^{\mathrm{th}}$ and $\pi^{-1}(k)^{\mathrm{th}}$ processing elements, as required when interleaving LLRs from the upper row to the lower row, as well as when deinterleaving from the lower row to the upper row, respectively. Note that the extrinsic message LLR $\bar{b}^{\mathrm{e}}_{1,k}$ is routed through both the interleaving and the deinterleaving paths in every clock cycle, which implies that each processing element receives two *a priori* message LLRs $\bar{b}^{\mathrm{a}}_{1,k}$ at a time. However, only the desired one is selected by the corresponding multiplexer, in accordance with the odd and even clock cycle scheduling. At the end of each clock cycle, the unshaded registers shown in Figure 5 are employed to cache $\bar{b}^{\mathrm{a,u}}_{1,k}$, $\bar{\boldsymbol{\alpha}}_k$ and $\bar{\boldsymbol{\beta}}_k$, ready for use in the next clock cycle. Note that the interleaver may be implemented using hard wires, implying that only a single fixed frame length N is supported at run time, although this particular frame length may be selected during synthesis. Our future work will consider the replacement at the hard-wired interleaver with a Beneš network, which will enable the run-time support of different frame lengths, having different interleaver designs.

In addition to the unshaded registers shown in Figure 12, each processing element employs a pair of registers for storing the extrinsic message LLR $\bar{b}^{\mathrm{e,u}}_{1,k}$ and the *a priori* message LLR $\bar{b}^{\mathrm{a,u}}_{1,k}$, as indicated using light and dark shading in Figure 5, respectively. Here, the lightly-shaded registers are updated in odd clock cycles, while the darkly-shaded registers are updated in even clock cycles. More specifically, in each odd clock cycle, each processing element having an odd index k generates the upper extrinsic message LLR $\bar{b}^{\mathrm{e,u}}_{1,k}$, which is cached in the lightly-shaded register of that processing element. Meanwhile, each processing element having an even index k generates the lower extrinsic message LLR $\bar{b}^{\mathrm{e,l}}_{1,k}$ during each odd clock cycle. This is deinterleaved in order to obtain the upper *a priori* message LLR $\bar{b}^{\mathrm{a,u}}_{1,\pi^{-1}(k)}$, where $\pi^{-1}(k)$ is guaranteed to be even owing to the odd-even nature of the LTE interleaver. Following this, $\bar{b}^{\mathrm{a,u}}_{1,\pi^{-1}(k)}$ is cached in the lightly-shaded register of the processing element having the even index $\pi^{-1}(k)$. By contrast, in each even clock cycle, each processing element having an even index k generates the upper extrinsic message LLR $\bar{b}^{\mathrm{e,u}}_{1,k}$, which is cached in the darkly-shaded register of that processing element. Meanwhile, each processing element having an odd index k generates the lower extrinsic message LLR $\bar{b}^{\mathrm{e,l}}_{1,k}$ during each even clock cycle. This is deinterleaved in order to obtain the upper *a priori* message LLR $\bar{b}^{\mathrm{a,u}}_{1,\pi^{-1}(k)}$, where $\pi^{-1}(k)$ is guaranteed to be odd owing to

the odd-even interleaver design. Following this, $\bar{b}^{\mathrm{a,u}}_{1,\pi^{-1}(k)}$ is cached in the darkly-shaded register of the processing element having the odd index $\pi^{-1}(k)$. During each clock cycle, the *a posteriori* LLRs $[\bar{b}^{\mathrm{p}}_{1,k}]^{N}_{k=1}$ can be updated by adding the contents of the lightly-shaded and darkly-shaded registers, together with the *a priori* systematic LLRs $[\bar{b}^{\mathrm{a,u}}_{3,k}]^{N}_{k-1}$. The iterative decoding process continues until a fixed clock cycle limit is reached or until the *a posteriori* LLRs satisfy a CRC check, which is computed within a single clock cycle, as it will be described in Section V.

V. FPGA Implementation

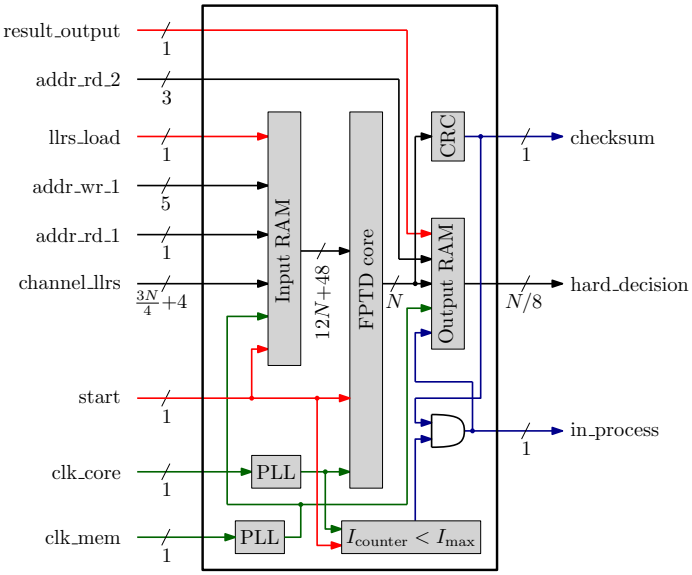


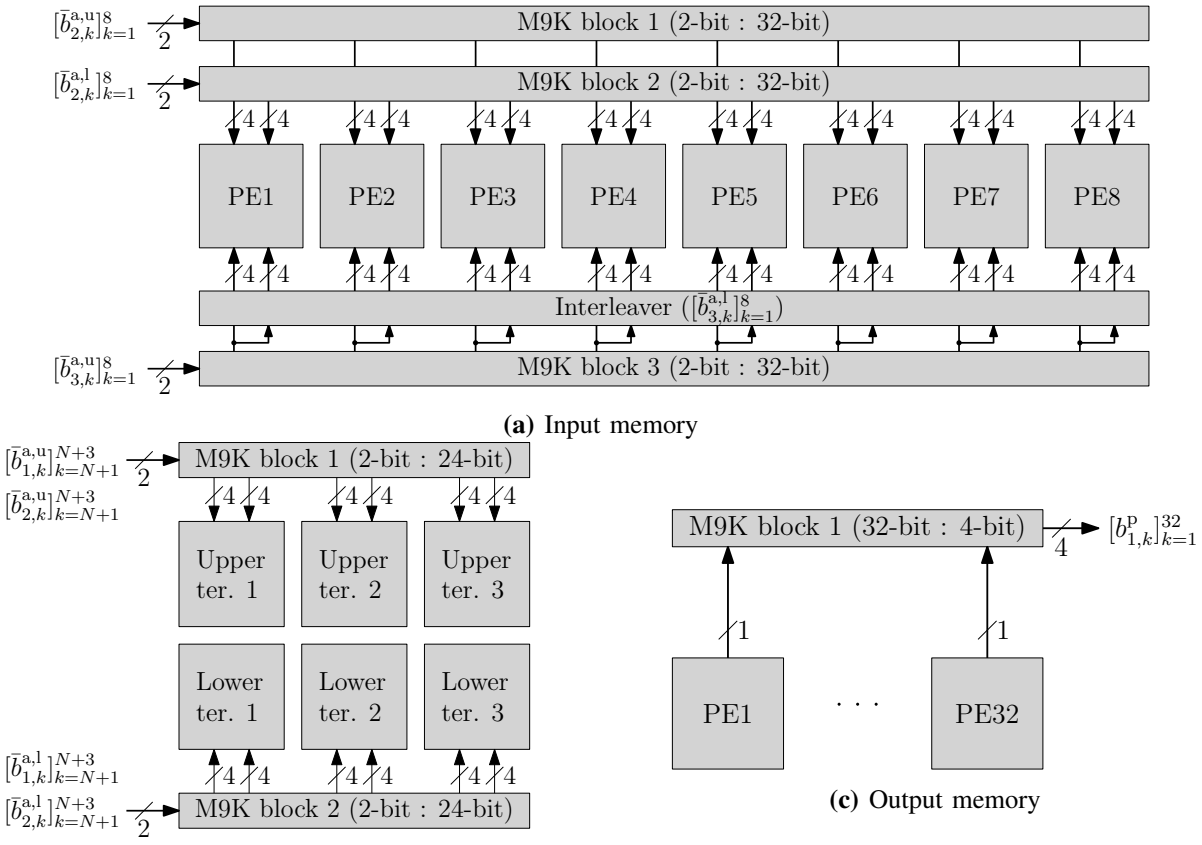
Fig. 13: Top-level schematic of the proposed FPGA implementation of the fixed-point FPTD architecture, where datapaths, input controls, output controls and clocks are cataloged and shown by black lines, red lines, blue lines and green lines, respectively.

In this section, we propose an FPGA implementation which integrates the FPTD architecture of Section IV with the I/O mechanisms and a fully-parallel CRC, which can operate in a single clock cycle. More specifically, Figure 13 provides the top-level schematic of the proposed FPGA implementation of the resource-efficient fixed-point FPTD architecture. This schematic comprises several functional components, including the FPTD core of Section IV, the input/output memory, CRC circuit, Phase-Locked Loops (PLLs) [52] and clock cycle counter. Note that two PLLs are employed, since the input and output RAMs are operated using a different higher frequency clock than the FPTD core and CRC circuit. Each of the interconnections among these functional components are classified as one of the input control signals, output control signals, clocks and datapaths, according to the color-coding shown in Figure 13. Furthermore, the input pins and output pins are shown on the left-hand side and the right-hand side of Figure 13, respectively. The input/output memory, CRC circuit and the control mechanism are detailed in Sections V-A, V-B and V-C, respectively.

A. I/O memory

As discussed in Section III-A, the input to the FPTD core comprises $(3N+12)$ *a priori* channel LLRs, namely N upper systematic LLRs $[\bar{b}^{\mathrm{a,u}}_{3,k}]^{N}_{k=1}$, N upper parity LLRs $[\bar{b}^{\mathrm{a,u}}_{2,k}]^{N}_{k=1}$, N lower parity LLRs $[\bar{b}^{\mathrm{a,l}}_{2,k}]^{N}_{k=1}$, three upper message termination LLRs $[\bar{b}^{\mathrm{a,u}}_{1,k}]^{N+3}_{k=N+1}$, three lower message termination LLRs $[\bar{b}^{\mathrm{a,l}}_{1,k}]^{N+3}_{k=N+1}$, three upper parity termination LLRs $[\bar{b}^{\mathrm{a,u}}_{1,k}]^{N+3}_{k=N+1}$ and three lower parity termination LLRs $[\bar{b}^{\mathrm{a,u}}_{1,k}]^{N+3}_{k=N+1}$. A bit-width of $w_1 = 4$ is employed for each of these *a priori* LLRs, corresponding to a total of $12N + 48$ bits. However, it is not possible to feed the FPTD core using a one-to-one mapping of the FPGA's general I/O pins, owing to the limited number of pins, especially when the frame length N is large. Owing to this, the input memory shown in Figure 13 is employed to provide a serial-to-parallel conversion and the storage of these bits. More specifically, the input memory comprises a number of the FPGA's M9K Static Random Access Memory (SRAM) blocks, which are all configured in the simple dual-port mode [53]. In this mode, one port is configured as an input port with a fixed bit-width of 2, while the other port is configured as an output port with a fixed bit-width of 32. Note that 32 bits is the maximum bit-width that an M9K block can support, in addition to its four parity bits. This implementation requires a total of $\lceil \frac{12N+48}{32} \rceil$ M9K memory blocks for the input memory, but only necessitates $2 \times \lceil \frac{12N+48}{32} \rceil = \frac{3N}{4} + 4$ input pins, as shown in Figure 13. Accordingly, the input memory requires 16 consecutive memory clock cycles to load the *a priori* LLRs, but necessitates only a single clock cycle to feed them to the FPTD core. This serial-to-parallel conversion ratio of 1:16 is motivated by the trade-off between the I/O pin usage and the number of clock cycles required to load a frame, since a larger ratio requires fewer clock cycles to transfer the data, but occupies more I/O pins. More specifically, our experimental results show that the largest FPTD that can be accommodated on our testbench EP4SE820F43C3 FPGA has a frame length of $N = 720$ bits, which is limited by the computational resource capacity. In this case, the input memory occupies $\frac{3N}{4} + 4 = 544$ of this FPGA's 1104 general I/O pins [54].

Furthermore, each M9K memory block has a capacity of 8192 bits besides the parity bits, which is sufficient to



(a) Input memory

(b) Input memory for the termination bits

(c) Output memory

Fig. 14: The mapping between M9K memory blocks and the processing elements, where **(a)** shows that three 2:32 memory blocks may be used to feed eight consecutive processing elements having indices of $k \leqslant N$, **(b)** shows that two 2:24 memory blocks may be used to feed the upper and the lower termination units, **(c)** shows that a 32:4 memory block may be used to output the hard-decision bits from 32 consecutive processing elements.

store upto 256 frames, since each frame occupies only 32 bits per M9K memory block. However, we employ only 64 bits per M9K memory block in this work for the sake of simplicity, which allows two frames to be stored at the same time. As shown in Figure 13, the input memory accordingly accepts a 5-bit address **addr_wr_1** for the 2-bit wide write port and a 1-bit address **addr_rd_1** for the 32-bit wide read port, in order to allow switching between the two independent frames. This allows the iterative decoding of one frame to be pipelined with the loading of the next frame, as it will be described in Section V-C. Note that the channel LLRs provided by the input memory read port are fed directly to the FPTD core, without registers in between as a cache.

The mapping between the M9K memory blocks and the processing elements is illustrated in Figure 14. As shown in Figure 14(a), each memory block is used for providing channel LLRs from one of the sets of $[\bar{b}^{\mathrm{a,u}}_{2,k}]^{N}_{k=1}$, $[\bar{b}^{\mathrm{a,l}}_{2,k}]^{N}_{k=1}$ and $[\bar{b}^{\mathrm{a,u}}_{3,k}]^{N}_{k=1}$ for eight neighboring processing elements, in the case where each *a priori* channel LLR uses $w_1 = 4$ bits. Therefore, three memory blocks are required for each set of eight processing elements, necessitating a total of $3N/8$ memory blocks for the input memory, when excluding the LLRs pertaining to the termination bits. These termination LLRs correspond to a total of 48 bits, which can be provided using the first 24 bits from each of the two above-mentioned 1:16 M9K memory blocks, as shown in Figure 14(b).

In addition to the input memory, an output memory is employed for storing and outputting the N hard-decision bits obtained by the FPTD core of Section IV, when the iterative decoding process is completed. Similarly to the input memory, the output memory is also implemented using dedicated M9K SRAM blocks, configured in simple two-port mode. However, the output memory is used for supporting a parallel-to-serial conversion, in contrast to the serial-to-parallel conversion required for the input memory. More specifically, the input port and the output port are configured to have a 32-bit and 4-bit width, respectively, giving a parallel-to-serial ratio of 8:1. In analogy to the implementation of the input memory, $\lceil N/32 \rceil$ M9K memory blocks are required for

the output memory, allowing the N hard decision bits $[b^{\mathrm{p}}_{1,k}]^{N}_{k=1}$ to be cached into the output memory in a single clock cycle, which are then output using $N/8$ output pins during eight consecutive clock cycles. For the sake of simplicity, we employ only 32 of the 8192 bits that can be stored in each M9K memory block, allowing the storage of a single decoded frame at a time. As shown in Figure 13, the output memory accordingly accepts only a 3-bit address **addr_rd_2** for its read port, whereas the address for the 32-bit wide write port can be fixed to zero internally. Note that outputting these hard decision bits can also be pipelined with the iterative decoding and loading of the next frames, as it will be detailed in Section V-C.

B. CRC

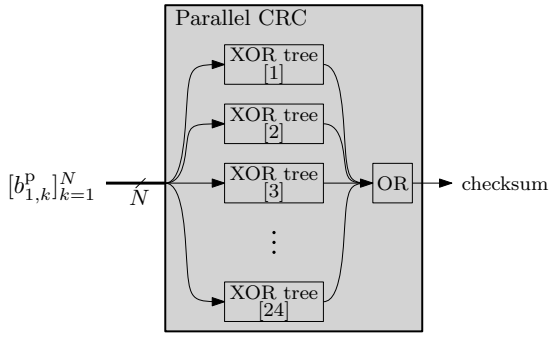


Fig. 15: Schematic of the single clock cycle LTE CRC.

As described in Section IV, the FPTD core performs iterative decoding in accordance with the FPTD algorithm of Figure 5, obtaining N hard-decision bits $[b^{\mathrm{p}}_{1,k}]^{N}_{k=1}$ in every clock cycle while it is active. In each clock cycle, these hard-decision bits are provided to an LTE CRC circuit, which detects if and when the FPTD core successfully decodes a frame before a predetermined clock cycle limit has been reached, hence enabling early stopping. However, this requires completing the computation of the CRC in a single clock cycle, which is not feasible, when adopting the conventional LFSR approach to implement the CRC [55]. Motivated by this, our FPGA implementation employs a fully-parallel CRC circuit, which is capable of computing the CRC in a single clock cycle and without an excessive critical path length, according to the design guideline of [56]. More specifically, the LTE turbo code employs a 24-bit CRC, having the generator polynomial of $g(D) = [D^{24} + D^{23} + D^6 + D^5 + D + 1]$. In our fully-parallel CRC circuit of Figure 15, the 24 CRC bits are computed simultaneously using separate predefined XOR trees within a single clock cycle. Each XOR tree is constructed by unfolding the corresponding LFSR operations of a conventional CRC circuit, accepting a particular selection of N hard-decision bits $[b^{\mathrm{p}}_{1,k}]^{N}_{k=1}$ as its input. Note that many of the XOR operations can be shared between XOR trees that consider overlapping sets of bits. Each of the XOR trees takes no more than N bits as input, therefore requiring no more than $(N-1)$ 2-input XOR gates, which may be structured as a tree comprising no more than $\lceil \log_2(N) \rceil$ layers. Hence, $\lceil \log_2(N) \rceil$ represents an upper bound on the number of gates in the critical path length of the parallel CRC. Following the XOR trees, the CRC result is obtained by performing the OR logic combination of all 24 CRC bits, using a tree comprising 23 2-input OR gates, arranged in $\lceil \log_2(24) \rceil = 5$ layers. As shown in Figure 13, the resultant **checksum** is output by the FPGA to indicate the correctness of the decoded results, where a zero value indicates successful decoding. Note that in the proposed FPGA implementation of Figure 13, the CRC is operated simultaneously with the FPTD core, but operates on the hard-decision bits that is provided in the previous clock cycle. Therefore, the CRC circuit does not affect the 6-stage critical path length of the FPTD core, as discussed in Section III-B.

C. Operation and control

As shown in Figure 13, the input and output memory is clocked by an external clock **clk_mem** having a clock frequency $f_{\mathrm{mem}} = 333$ MHz, while the FPTD core and CRC circuit are clocked by another external clock **clk_core** having a different clock frequency f_{core}, which depends on the frame length N. Both clocks are compensated using different ones of the FPGA's dedicated PLL circuits. Figure 16 illustrates a time diagram for an example operation of the proposed FPTD, including all three loading, processing and output stages. The loading stage comprises the operation of the input memory, which requires 16 memory clock cycles, as discussed in Section V-A. This loading process is controlled by the control signal **llrs_load**, as shown in Figure 16. Following these 16 memory clock cycles, there is a delay of approximately two memory clock cycles before the iterative decoding process begins. This delay is required by the memory read and for overcoming the phase difference between the FPTD core clock **clk_core** and the memory clock **clk_mem**. Here, the memory read is triggered at the rising edge of the **start** signal, which therefore feeds the FPTD core with the *a priori* channel LLRs, in accordance with the selected memory read address **adr_rd_1**. Considering these 18 memory clock cycles, the overall time used for loading is $18/f_{\mathrm{mem}} = 54.1$ ns.

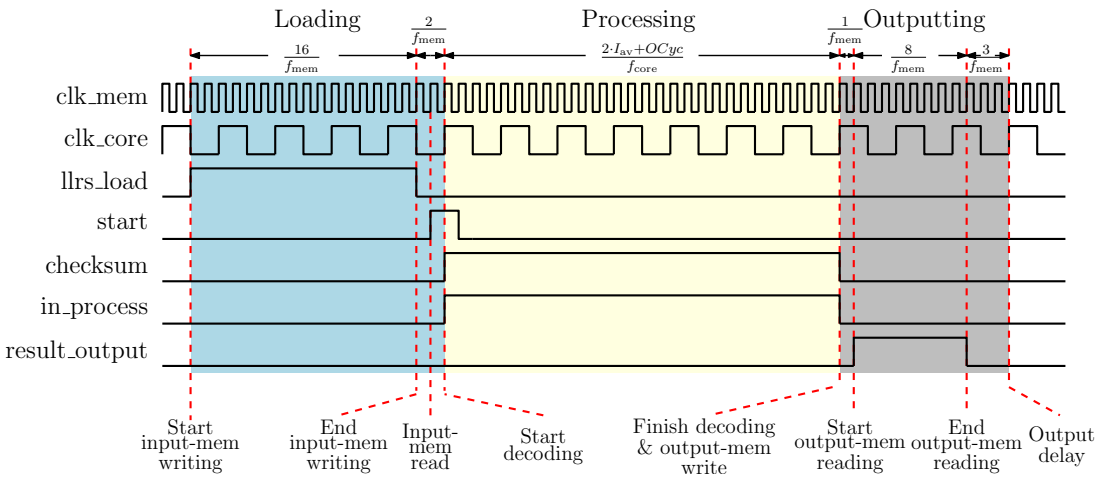


Fig. 16: An example time diagram of the proposed FPTD operation.

The pulse of the **start** control signal is also used for resetting the FPTD core synchronously with **clk_core**, as well as to activate its iterative decoding process. The **in_process** output signal of Figure 13 is asserted throughout this iterative decoding processing, which continues until the **checksum** becomes zero or until a maximum number $I_{\max}$ of decoding iterations has been reached. As shown in Figure 13, the **in_process** signal is implemented using a single AND gate, accepting the inputs of **checksum** from the CRC of Section V-B and the condition check of $I_{\text{counter}} < I_{\max}$. Here, I_{counter} may be accumulated using a generic ripple counter, which is reset to zero when the **start** control signal is pulsed. The average duration of the iterative decoding process is given by $\tau_1 = \frac{2 \cdot I_{\text{av}} + OCyc}{f_{\text{core}}}$, where I_{av} is the average number of iterations performed, $OCyc$ is the clock cycle overhead and f_{core} is the clock frequency for the FPTD core. The worst-case duration $\tau_1^{\max} = \frac{2 \cdot I_{\max} + OCyc}{f_{\text{core}}}$ is incurred for frames, where the iterative decoding process is terminated, when the counter I_{counter} reaches the maximum iteration limit $I_{\max}$. In the proposed FPTD implementation, the overhead is $OCyc = 2$ clock cycles, which comprises one clock cycle for resetting the FPTD core at the beginning of the iterative decoding process and one clock cycle delay for the CRC calculation of Figure 15 at the end.

As shown in Figure 16, the write operation for the output memory is triggered by a failing edge of the **in_process** signal. This causes the N hard decision bits $[b^{\text{p}}_{1,k}]_{k=1}^{N}$ obtained for the present frame to be stored in the output memory within a single memory clock cycle. Following this, the control signal **result_output** is asserted for eight memory clock cycles, in order to signal that the hard-decision bits are being output, as discussed in Section V-A. In addition to this, there is an output delay of approximately three memory clock cycles. Considering all of these 12 memory clock cycles, the time required for outputting the results is $12/f_{\text{mem}} = 36$ ns.

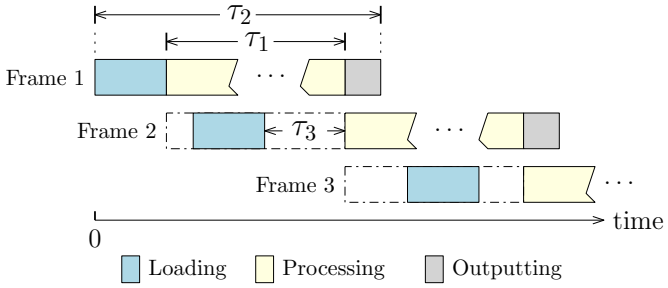


Fig. 17: Timeline for pipelining the turbo decoding of three consecutive frames, where τ_2 is the time required for completely decoding a frame, in which the iterative turbo decoding process occupies τ_1 time. In addition, τ_3 is the time delay which may be incurred between completing the loading of a frame and beginning its iterative decoding.

As illustrated in Figure 17, the loading, processing and outputting operations may be pipelined, in order to maximize the processing throughput when decoding several successive frames. More specifically, the proposed FPTD decoder is implemented for such that as soon as the processing operation for the present frame is started, the loading of the next frame can begin. This allows the FPTD core to start the iterative decoding processing of the next frame immediately after completing the decoding process for the present frame. In this way, the average decoding throughput can be improved from $\frac{N}{\tau_2}$ to $\frac{N}{\tau_1}$, where $\tau_1 = \frac{2 \cdot I_{\text{av}} + OCyc}{f_{\text{core}}}$ is the average delay incurred by the FPTD core, while $\tau_2 = \tau_1 + 30/f_{\text{mem}}$ is the overall delay for completely decoding a frame, including loading, processing and outputting. Note that this throughput improvement can only be achieved for the specific case, where the next frame becomes available before the iterative decoding process of the current frame has

been completed. Furthermore, this pipelining technique does not improve the latency of the proposed FPTD implementation, which has the value $\tau_2 = \tau_1 + 30/f_{\text{mem}}$ on average and $\tau_2^{\max} = \tau_1^{\max} + 30/f_{\text{mem}}$ in the worst case. Note that Figure 17 illustrates a time delay τ_3, which may be incurred between completing the loading of a frame and beginning its iterative decoding. However, our characterization of the proposed FPTD implementation's latency does not include τ_3, since it may vary from frame to frame and since its value depends on the timing of the delivery of frames by the demodulator, which is outside the scope of this work.

VI. Results

In this section, we characterize the proposed FPGA implementation of our fixed-point LTE FPTD, using the Altera FPGA EP4SE820F43C3, which comprises 813k Logic Elements (LEs), 650k registers, 1.6k M9K memory blocks and 12 PLLs. These results are compared to the state-of-the-art FPGA implementation of the LTE turbo decoder of [14], which applies the conventional Log-BCJR algorithm to the same FPGA. More specifically, we compare our proposed FPGA implementation to the benchmarker of [14] in terms of its BER, throughput, latency and resource usage, in Sections VI-A, VI-B, VI-C and VI-D, respectively. Following this, we characterize the energy consumption of our proposed FPGA implementation in Section VI-E, although we are unable to compare this to that of the benchmarker. This is because the energy consumption of the benchmarker implementation is not discussed in [14] nor is it discussed for other existing FPGA implementations of the Log-BCJR based LTE turbo decoder. Finally, we perform an overall comparison between the proposed LTE FPTD FPGA implementation and other state-of-the-art FPGA implementations of the Log-BCJR turbo decoder and various LDPC decoders in Section VI-F.

A. BER

The BER performance of the proposed FPGA implementation of the FPTD is compared to that of the benchmarker FPGA implementation of the Log-BCJR turbo decoder of [14] in Figure 18. Here, the benchmarker employs the exact max* operation of (6), while performing $I = 5$ iterations, for the case where $N = \{512, 1024\}$. By contrast, the same BER performance can be achieved for our proposed fixed-point FPTD, when employing the approximate max* operation of (7) and performing $I \leqslant 28$ iterations. More specifically, the iterative decoding is curtailed, once a sufficient number of iterations have been performed to achieve successful decoding or

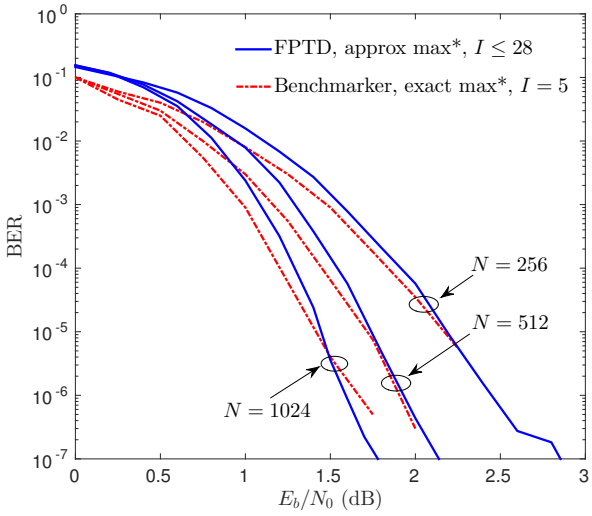


Fig. 18: BER comparison between the proposed fixed-point FPTD and the benchmarker of [14]. The FPTD employs the approximate max* operation of (7) and performs $I \leqslant 28$ iterations for frame lengths of $N \in \{256, 512, 1024\}$. The benchmarker decoder of [14] employs the exact max* operation of (6) and performs $I = 5$ iterations for frame lengths of $N \in \{256, 512, 1024\}$.

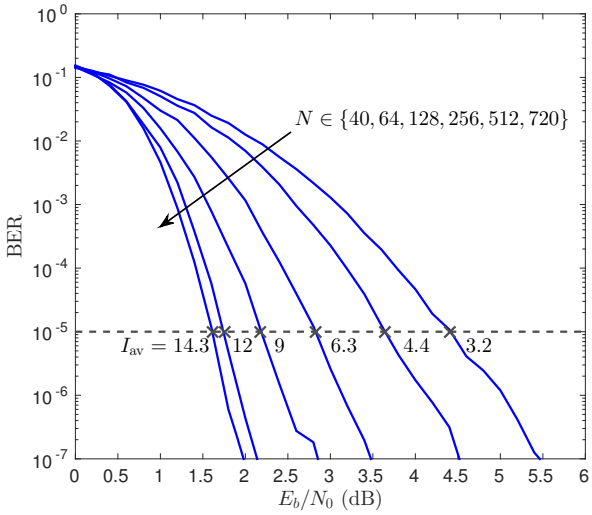


Fig. 19: BER performance of the proposed FPTD algorithm, performing $I \leqslant 28$ iterations for different frame $N \in \{40, 64, 128, 256, 512, 720\}$, where I_{av} is the average number of iterations used to achieve the target BER of 10^{-5}.

when a limit of $I_{\text{maxt}} = 28$ iterations is reached. Furthermore, Figure 19 characterizes the BER performance of the proposed fixed-point FPTD performing $I \leqslant 28$ iterations for decoding frames having different lengths of $N \in \{40, 64, 128, 256, 512, 720\}$, which can all be accommodated within the hardware of the target FPGA. Note that the BER performance is not provided in [14] for the frame lengths of $N \in \{40, 64, 128, 720\}$ for the benchmarker, hence we are unable to compare it

to our proposed FPTD. Note that Figure 19 shows the average number I_{av} of iterations used by the proposed implementation for each frame length at the specific E_b/N_0 value, where a BER of 10^{-5} is reached.

B. Throughput

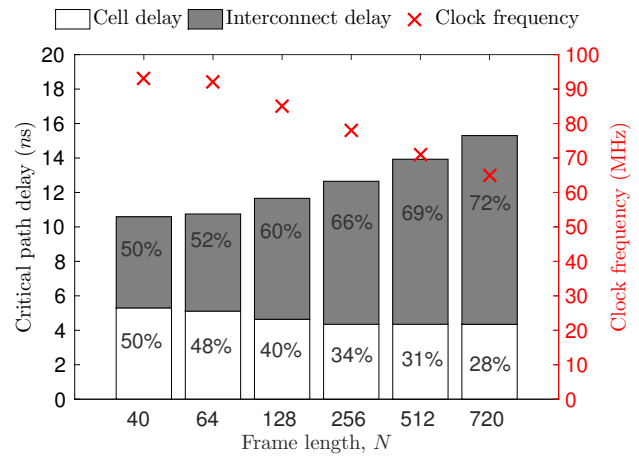


Fig. 20: Critical path delay and maximum clock frequency for different frame lengths $N \in \{40, 64, 128, 256, 512, 720\}$.

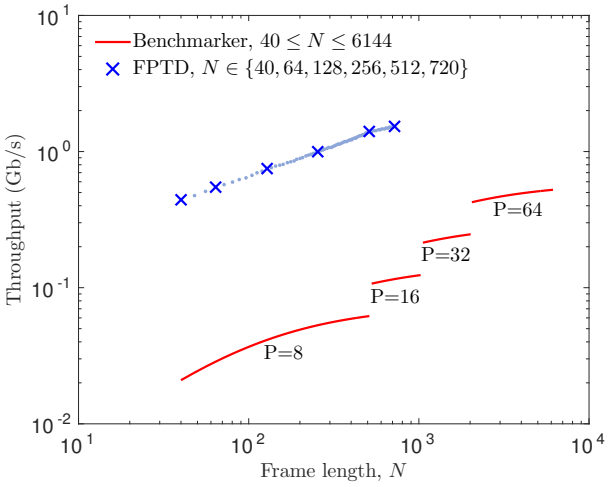


Fig. 21: Comparison of throughput for the proposed fixed-point FPTD FPGA implementation and for the benchmarker FPGA decoder of [14], where $I \leqslant 28$ iterations are compared for the proposed FPTD, while the benchmarker decode employs $I = 5$ iterations.

Figure 20 characterizes the critical path delay of the proposed FPTD core for the frame lengths of $N \in \{40, 64, 128, 256, 512, 720\}$, as well as their resultant maximum clock frequency, as reported by the Quartus design tool. Here, the critical path delay comprises two parts, namely the cell delay and interconnect delay. The cell delay is the sum of the time occupied by all the combinational components residing on the critical path. By contrast, the interconnect delay is the sum of the time occupied the interconnections between those combinational components. As shown in Figure 20, when implementing the FPTD for the shortest LTE frame length of $N = 40$ bits, the critical path delay is 10.6 ns, in which the cell delay and the interconnect delay are evenly distributed, achieving a maximum clock frequency of $f_{\text{core}} = 93$ MHz. Note that this maximum clock frequency depends also on the delay associated with the clock tree, although this is negligible compared to the cell delay and interconnect delay shown in Figure 20. When the frame length N is increased from $N = 40$ bits to $N = 720$ bits, the critical path delay increases gradually to 15.3 ns and the maximum clock frequency of f_{core} decreases accordingly to $f_{\text{core}} = 65$ MHz. As shown in Figure 20, this increased critical path delay versus N is increased and it is mainly imposed by the interconnects, while the cell delay is reduced slightly. This may be because a greater fraction of the FPGA's logic elements are employed for implementing the FPTD, when the frame length N is increased, which increases the difficulty of optimizing the placing and routing. In particular, the interleaver may be required to route information between processing elements that are implemented near the opposite corners of the FPGA. By contrast, the reduced cell delay may be attributed to the deeper optimization performed by the Quartus design tool, when the resources become limited.

As described in Section V-C, the average throughput of the proposed FPTD is given by $\frac{N}{\tau_1}$, where $\tau_1 = \frac{2 \cdot I_{\text{av}} + OCyc}{f_{\text{core}}}$. Note that the throughput is a function of the frame length N, as well as the clock frequency f_{core} and the average number of iterations performed I_{av}, which also both depend on N, as characterized in Figures 19 and 20. Owing to this, Figure 21 compares the throughput of the proposed FPTD with that of the benchmarker FPGA implementation of [14], as a function of N. More specifically, the resultant throughput of the proposed FPTD ranges from 442 Mbit/s for $N = 40$ to 1.53 Gbit/s for $N = 720$. By contrast, the throughput of the benchmarker of [14] is given by $\frac{f_{\text{clk}} \cdot N}{2 \cdot I \cdot (\frac{N}{P} + OCyc) + \frac{N}{P}}$, where the clock frequency is $f_{\text{clk}} = 102$ MHz, the number of iterations performed is $I = 5$ and the overhead is $OCyc = 14$ clock cycles per half-iteration. Furthermore, the benchmarker decoder of [14] comprises 64 sub-decoders, each of which processes one or zero partitions of the frame, depending on the frame length N. More specifically, a frame having the length N is decomposed into P partitions, where

$$P = \begin{cases} 8, & \text{if } 40 \leqslant N \leqslant 512 \\ 16, & \text{if } 528 \leqslant N \leqslant 1024 \\ 32, & \text{if } 1056 \leqslant N \leqslant 2016 \\ 64, & \text{if } 2048 \leqslant N \leqslant 6144. \end{cases}$$

Considering these configurations, the resultant throughput of the benchmarker FPGA decoder of [14] is in the range from 21 Mbit/s for $N = 40$ to 524 Mbit/s for $N = 6144$, as shown in Figure 21. Note that these throughputs are 21 and 13.2 times lower than those of the proposed FPTD decoder for the cases of $N = 40$ and $N = 720$, respectively. Furthermore, the maximum throughput gain is achieved by the proposed FPTD decoder for $N = 512$, where it has a throughput of 1.4 Gbit/s, which is 22.6 times higher than the 62 Mbit/s achieved by the benchmarker decoder of [14].

C. Latency

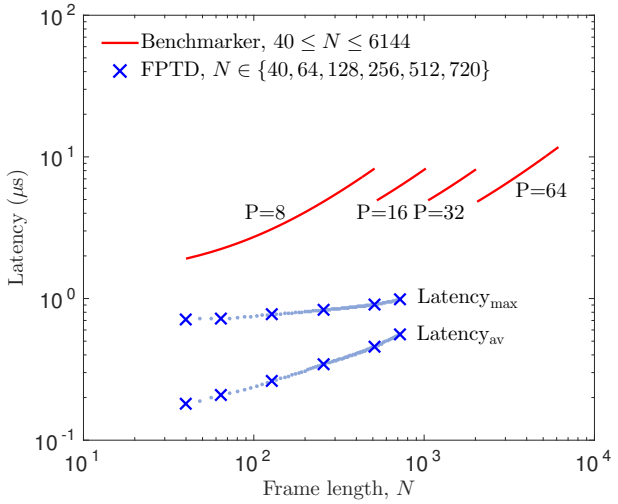


Fig. 22: Comparison of average and maximum latency for the proposed fixed-point FPTD FPGA implementation and for the benchmarker FPGA decoder of [14], where $I \leqslant 28$ iterations are compared for the proposed FPTD, while the benchmarker decode employs $I = 5$ iterations.

As described in Section V, the average latency imposed by loading, processing and outputting a frame is given by $\frac{2 \cdot I_{\text{av}} + OCyc}{f_{\text{core}}} + \frac{30}{f_{\text{mem}}}$, while the worst-case latency is given by $\frac{2 \cdot I_{\text{max}} + OCyc}{f_{\text{core}}} + \frac{30}{f_{\text{mem}}}$, which is incurred when decoding is unsuccessful. By contrast, the latency of the benchmarker decoder is not quantified in [14] but may be optimistically estimated as latency $= N/$throughput, which ignores the latency for loading and outputting the data. As shown in Figure 22, the latency of the benchmarker decoder ranges between 1.9 μs when $N = 40$ and 6.2 μs when $N = 720$. By contrast, our proposed FPTD FPGA implementation achieves a worst-case latency of 0.72 μs when $N = 40$ and 0.98 μs when $N = 720$, which are 2.6 times and 6.3 times less than those of the benchmarker decoder. Meanwhile the average latency of our proposed FPTD FPGA implementation reduces to 0.18 μs when $N = 40$ and 0.56 μs when $N = 720$, which are 10.6 times and 11.1 times less than those of the benchmarker decoder. Here, the maximum latency improvement is obtained when $N = 512$, where the latency of 0.46 μs for the proposed FPTD is 18 times less than the 8.3 μs, obtained by the benchmarker of [14].

D. Resource usage

The resource usage of the proposed $N = 720$ FPTD FPGA implementation is compared in Table I, in terms of combinational Adaptive Look-Up-Tables (ALUTs), as well as memory ALUTs, dedicated logic registers and total block memory bits. Note that the EP4SE820F43C3 FPGA has a capacity of 650,440 ALUTs, half of which can be configured to implement combinational logic, while the other half can be configured as either combinational logic or as memory. Here, we compare three versions for the benchmarker decoder of [14], namely $P = 8$, $P = 16$ and $P = 64$ versions. The $P = 64$ version is the original implementation presented in [14], which comprises 64 sub-decoders and is capable of supporting all LTE frame lengths at run time. However, our proposed FPTD implementation supports only a single frame length of up to $N = 720$ bits at run time. In order to facilitate fairer resource usage comparisons with our FPTD, the $P = 8$ and $P = 16$ versions of the benchmarker decoder comprise only 8 and 16 sub-decoders, respectively. As described in Section VI-B, this is motivated, since the benchmarker of [14] only uses $P = 8$ and $P = 16$ sub-decoders for frame lengths N in the ranges 40 to 512 and 528 to 1024, respectively. Owing to this, the $P = 8$ and $P = 16$ versions offer the same throughputs as the $P = 64$ version of the benchmarker for frame lengths in the ranges 40 to 512 and 528 to 1024 respectively, but at the cost of lower hardware usage. Note that the resource usage of the $P = 8$ and $P = 16$ versions reported in Table I was estimated by linearly scaling those of the $P = 64$ version. As shown in Table I, the $N = 720$ FPTD occupies 99% of the EP4SE820F43C3 FPGA's ALUTs as combinational logic, while it uses 11% of the FPGA's dedicated registers and 0.04% of its memory bits. Furthermore, the $N = 720$ FPTD employs 650 general I/O pins, in accordance with Figure 13. By contrast, the $P = 16$ version of the benchmarker decoder occupies 8.8% of the ALUTs as combinational logic and 1% of the ALUTs as memory, while it uses 8% of the dedicated registers and 0.2% of the total memory bits.

TABLE I: Resource usage comparison between the proposed FPTD FPGA implementation and the benchmarker FPGA implementation of [14]. The percentages shown in brackets indicate the corresponding fraction of the capacity of the EP4SE820F43C3 FPGA.

Implementations	Benchmarker [14] ($P = 8$)	Benchmarker [14] ($P = 16$)	Benchmarker [14] ($P = 64$)	FPTD
Supported frame lengths N	40, 48, 56, ..., 6144	40, 48, 56, ..., 6144	40, 48, 56, ..., 6144	720
Comb. ALUTs	28480 (4.4%)	56960 (8.8%)	227843 (35%)	644464 (99%)
Memory ALUTs	3308 (0.5%)	6616 (1%)	26465 (4%)	0 (0%)
Dedicated logic registers	26606 (4%)	53213 (8%)	212852 (33%)	73443 (11%)
Total block memory bits	25280 (0.1%)	50560 (0.2%)	202240 (0.9%)	28928 (0.04%)
I/O pins	-	-	-	650 (58.9%)

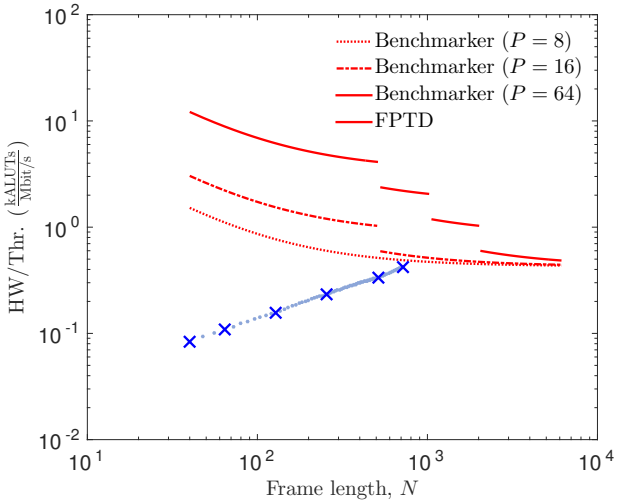


Fig. 23: Normalized resource usage comparison between the proposed FPTD FPGA implementation and the benchmarker of [14] with $P = 8$, $P = 16$ and $P = 64$ sub-decoders.

The resource usage may be normalized as $\frac{\text{ALUTs}}{\text{throughput}}$, since both the proposed FPTD implementation and the benchmarker are limited by ALUT resources, rather than by memory. Figure 23 depicts the normalized resource usage of the proposed FPTD FPGA implementation as a function of the frame length N, compared with those of the $P = 8$, $P = 16$ and $P = 64$ versions of the benchmarker decoder. Note that for frmae lengths N above 512 and 1024 bits respectively, the throughputs of the $P = 8$ and $P = 16$ versions of the benchmarker are estimated by linearly scaling those of the $P = 64$ version, as shown in Figure 21. As shown in Figure 23, the normalized resource usage of the proposed FPTD FPGA implementation is 0.08 $\frac{\text{kALUTs}}{\text{Mbit/s}}$ for the case, where $N = 40$ and 0.42 $\frac{\text{kALUTs}}{\text{Mbit/s}}$ for the case where $N = 720$. These are 19 times lower than the 1.52 $\frac{\text{kALUTs}}{\text{Mbit/s}}$ and 1.3 times lower than the 0.55 $\frac{\text{kALUTs}}{\text{Mbit/s}}$, which are achieved by the $P = 8$ and $P = 16$ versions of the benchmarker FPGA implementation for $N = 40$ and $N = 720$, respectively.

E. Energy

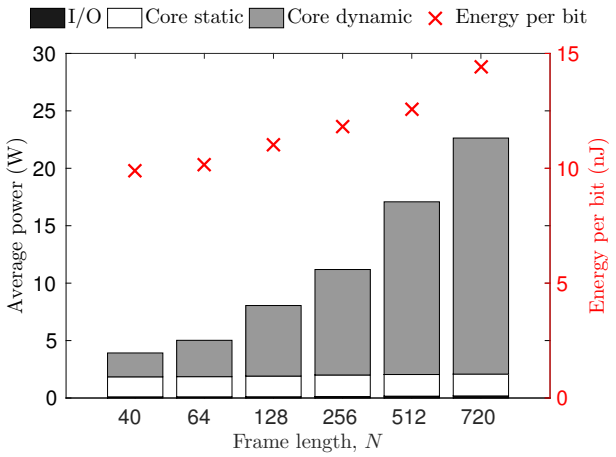


Fig. 24: Power consumption and Energy consumption per bit of the proposed FPTD FPGA implementation with different frame length $N \in \{40, 64, 128, 256, 512, 720\}$.

The power consumption of the proposed FPTD FPGA implementation was estimated using the power analysis tool in the design tool kit of Qaurtus II [57], based on the Value Change Dump (VCD) results obtained from a post-fit dynamic simulation of 100 frames. These frames were recorded during transmission over an AWGN channel using BPSK at the specific E_b/N_0 values, where the BER reaches 10^{-5}. More specifically, $E_b/N_0 \in \{4.41, 3.64, 2.83, 2.18, 1.76, 1.61\}$ dB are respectively used for the frame lengths $N \in \{40, 64, 128, 256, 512, 720\}$, according to Figure 19.

Figure 24 depicts the estimated power consumption of the proposed FPTD FPGA implementation. Here, the power consumption is classified into three components, namely I/O, core static and core dynamic, where the FPGA core includes all functional components shown in Figure 13. The core dynamic power consumption is dominated by the switching activity of all in-use hardware resources, which increases gradually with the frame length N, in correspondence with the associated increase of hardware resource usage. By contrast, the

TABLE II: Comparison between the proposed LTE FPTD FPGA implementation and different FPGA implementations of the conventional Log-BCJR turbo decoder.

Implementations	This work (N=720)	L.F. Gonzalez-Perez 2013 [14]	P. Murugappa 2012 [16]	M. Del Barco 2012 [27]	R. Shrestha 2013 [35]	O. Muller 2008 [15]
Code standard	LTE	LTE	LTE	LTE	DVB-SH	WiMAX
Radix/Parallelism	2/6144	2/64	4/8	2/16	2/20	2/4
Max* operation	Approx. max*	LUT-based exact max*	-	Approx. max*	Approx. max*	Approx. max*
Windowing technique	Fully-parallel	Non-sliding window/ Sliding window	Sliding window	Sliding window	Sliding window	-
Iterations I	$\leqslant$28	5	7	0.5	0.5	5
FPGA	EP4SE820	EP4SE820	XC5VLX330	XC5VLX330	XC5VFX200T	XC2VP30
FPGA's technology	40 nm	40 nm	65 nm	65 nm	65 nm	130 nm
Clock frequency [MHz]	65	102	80	100	346	110
Resource usage [kALUTs]	644	254	21.2 kSlices (169.6[a])	45.8 kSlices (366.4[a])	15.7 kSlices (125.6[a])	9.5 kSlices (19[b])
Peak throughput [Mbit/s]	1530	524	22.8 (52[c])	1600 (260[c])	346 (58[c])	6.3 (14.2[c])
Normalized resource usage [$\frac{\text{kALUTs}}{\text{Mbit/s}}$]	0.42	0.5	3.26[ac]	1.41[ac]	2.17[ac]	1.34[bc]
Energy consumption [nJ/bit]	14.1	-	-	-	-	-

[a] Multiplied by 8, since a slice in Virtex-5 FPGAs contains four 6-input look-up tables [58], which is equivalent to eight Altera ALUTs [13].

[b] Multiplied by 2, since a slice in Virtex-2 FPGAs contains two 4-input look-up tables [58], which is equivalent to two Altera ALUTs [13].

[c] Scaling linearly to $I = 5$ iterations and scaling to 40 nm technology, using a scaling factor of $\sim 1/s$, where s is the FPGA's technology [34].

core static power consumption is relatively consistent for all frame lengths, since this depends more upon the FPGA's technology and size, rather than its application. Furthermore, when implementing the $N = 40$-bit FPTD, the static power consumption comprises approximately 50% the total consumption. By contrast, the static power consumption represents only 8.4% of the total power consumption, when implementing the $N = 720$-bit FPTD, which occupies all of the FPGA's computational resources. Compared to the core dynamic and core static power consumption, the power consumption of the I/O pins is negligible, as shown at the bottom of each bar in Figure 24.

The average energy consumption per bit may be obtained as $\frac{\tau_2 \cdot \text{Power}}{N}$, where $\tau_2 = \frac{2 \cdot I_{\text{av}} + OCyc}{f_{\text{core}}} + \frac{30}{f_{\text{mem}}}$ is the average latency for decoding a frame, as described in Section VI-C. As shown in Figure 24, the average energy consumption per bit ranges from 9.9 nJ to 14.1 nJ, when the frame length is increased from $N = 40$ to $N = 720$. This increased energy consumption per bit is dominated by the increased energy consumption associated with routing, which is incurred by the more complex interconnections and clock trees that are associated with longer frames. Note that the energy consumption is not characterized for the benchmarker decoder in [14], or for any other state-of-the-art FPGA implementations of the Log-BCJR turbo decoder, hence preventing a comparison with our proposed FPTD implementation of the FPTD.

F. Overall comparison

Table II compares the overall characteristics of the proposed LTE FPTD FPGA implementation with several state-of-the-art LTE turbo decoder FPGA implementations based on the Log-BCJR algorithm. In order to facilitate fair comparisons with the other implementations, their characteristics have been scaled to become equivalent to using a 40 nm FPGA technology and using $I = 5$ decoding iterations, as shown in the brackets of Table II. Note that [27], [35] characterizes the throughput and resource usage for only a single MAP decoder,

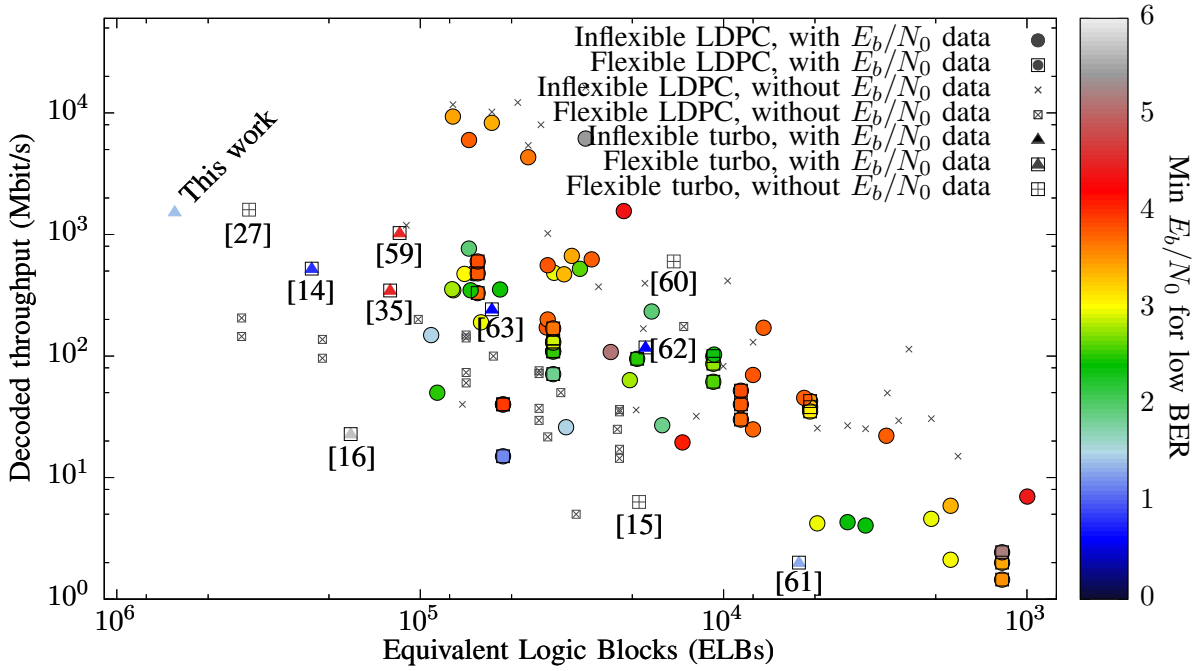


Fig. 25: A comparison between the FPGA implementations for turbo codes and LDPC codes, in terms of processing throughput, hardware resource usage, flexibility and BER performance.

without considering the overhead of implementing the interleaver and CRC circuit. Note also that the FPGAs from different vendors have widely differing architectures, which prevents a precise comparison in terms of resource usage. Nonetheless, we adopt the concept of Equivalent Logic Blocks (ELBs) defined in [13] to offer a fair comparison between the resource usage of implementations using FPGA manufactured by different vendors. More specifically, an ELB corresponds to a pair of 4-input Look-Up Tables (LUTs) and a register, where an ALUT in Altera FPGAs is equivalent to a single ELB, while a 6-input LUT in Xilinx FPGAs is approximately equivalent to two ELBs. As shown in Table II, the proposed LTE FPTD FPGA implementation achieves the highest peak processing throughput, compared to all other implementations considered. Note that the peak throughput of the proposed FPTD is achieved for a frame length of $N = 720$ bits, while the peak throughput of the other LTE turbo decoder implementations is achieved for the case of $N = 6144$-bit frames. Similarly, the normalized resource usage of the proposed LTE FPTD FPGA implementation is better than those of the other implementations, as shown in Table II.

Furthermore, Figure 25 compares the turbo decoder implementations of Table II with many FPGA implementations of LDPC decoders which were characterized in [13]. Here, the comparison considers processing throughput, hardware resource usage, BER performance and flexibility to support different frame lengths and coding rates at run-time. More specifically, Figure 25 plots the resource usage and the throughput of the various FPGA implementations on its x-axis and y-axis, respectively. Here, the resource usage is quantified using the above-mentioned ELB metric, which facilitates a fairer comparison between implementations that employ different FPGAs. Furthermore, in order to be consistent with the comparisons of [13], the throughputs presented in Figure 25 are the unscaled ones of Table II. In addition to throughput and resource usage, the flexibility of each implementation is identified by the shape of the data points, while the BER performance is indicated by their color. Here, the BER performance is characterized by the minimal E_b/N_0 value where a BER of 10^{-4} is achieved, which is related to the code design, coding rate, number of iterations and frame length. As shown in Figure 25, the flexible turbo decoders offer similar normalized resource usage ($\frac{\text{kELBs}}{\text{Mbit/s}}$) to the flexible LDPC decoders, despite the LDPC decoders typically having lower computational complexity. This may be attributed to the significantly further interconnection complexity of LDPC decoders, as well as to the significant challenges associated with implementing high-throughput flexible LDPC decoders. More specifically, all turbo decoder algorithmic blocks are identical and of them is only connected to its neighbors and a single algorithmic block through the interleaver. By contrast, the variable

and check nodes of an LDPC decoder have various degrees, often much greater than one, which quantifies the number of connected nodes through the interleaver. Owing to these complications, the flexible WiFi LDPC decoders support only 12 combinations of frame length and coding rate, while the LTE turbo decoders support 643 million combinations. Furthermore, the flexible turbo decoders of Figure 25 can be seen to offer superior BER performance to the flexible LDPC decoders.

VII. Conclusions

In this paper, we have proposed a novel area-efficient fixed-point LTE FPTD, which achieves 50% hardware resource reduction compared with the FPTD architecture of [37]. We have also proposed a holistic FPGA implementation of this resource-efficient FPTD, which includes schemes for loading each frame, processing it and outputting the results. The proposed FPGA implementation offers a processing throughput gain up to 22.6 times and a processing latency gain of up to 18 times, compared to those of the state-of-the-art FPGA implementation based on the conventional Log-BCJR LTE turbo decoder. The peak processing throughput of 1.53 Gbit/s and the worst case latency of 0.98 μs for the proposed FPTD implementation meet the throughput and latency requirements for state-of-the-art telephony communication standards, such as LTE cat.12 [64]. In particular, its processing latency represents only an insignificant fraction of the 1 ms end-to-end transmission latency budget for MCMTC applications [9]. Furthermore, the normalized resource usage of the proposed FPTD FPGA implementation is up to 19 times better than that of the $P = 8$ version of the benchmarker. Our future work will be motivated by the improvements desired for 5G communications, such as increased flexibility to a wider range of frame lengths, as well as improved hardware efficiency and energy efficiency. More specifically, we will consider techniques that can further reduce the resource usage and facilitate support for all LTE turbo code frame lengths, as well as any that are defined for 5G. We will also consider the employment of a Beneš network [65] in order to implement the LTE interleaver and to support different frame lengths at run time.

References

[1] J. Woodard and L. Hanzo, "Comparative Study Of Turbo Decoding Techniques: An Overview," *IEEE Trans. Veh. Technol.*, vol. 49, 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. Surv. Tutorials*, vol. PP, 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. Syst.*, vol. 14, 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. Syst.*, vol. 17, pp. 92–102, Jan 2009.

[5] H. Chen, R. G. Maunder, and L. Hanzo, "A Survey and Tutorial on Low-Complexity Turbo Coding Techniques and a Holistic Hybrid ARQ Design Example," *IEEE Commun. Surv. Tutorials*, vol. 15, pp. 1546–1566, Jan 2013.

[6] IEEE, "IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Broadband Wireless Access Systems," Mar 2012.

[7] ETSI, "LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and Channel Coding," Feb 2013.

[8] A. Osseiran, F. Boccardi, V. Braun, K. Kusume, P. Marsch, M. Maternia, O. Queseth, M. Schellmann, H. Schotten, H. Taoka, H. Tullberg, M. a. Uusitalo, B. Timus, and M. Fallgren, "Scenarios for 5G Mobile and Wireless Communications: the Vision of the METIS Project," *IEEE Commun. Mag.*, vol. 52, pp. 26–35, May 2014.

[9] O. N. C. Yilmaz, Y.-P. E. Wang, N. A. Johansson, N. Brahmi, S. A. Ashraf, and J. Sachs, "Analysis of Ultra-Reliable and Low-Latency 5G Communication for a Factory Automation Use Case," pp. 1190–1195, Jun 2015.

[10] K. Fagervik and A. Larssen, "Performance and Complexity Comparison of Low Density Parity Check Codes and Turbo Codes," *Proc. Nor. Signal Process. Symp.*, Oct 2003.

[11] T. Lestable, E. Zimmerman, M.-h. Hamon, and S. Stiglmayr, "Block-LDPC Codes vs Duo-Binary Turbo-Codes for European Next Generation Wireless Systems," in *IEEE Veh. Technol. Conf.*, pp. 1–5, IEEE, Sep 2006.

[12] C. Rachinger, R. Muller, and J. B. Huber, "Low Latency-Constrained High Rate Coding: LDPC Codes vs. Convolutional Codes," in *2014 8th Int. Symp. Turbo Codes Iterative Inf. Process.*, no. 2, pp. 218–222, IEEE, Aug 2014.

[13] P. Hailes, L. Xu, R. Maunder, B. Al-Hashimi, and L. Hanzo, "A Survey of FPGA-based LDPC Decoders," *IEEE Commun. Surv. Tutorials*, no. c, pp. 1–1, Jan 2015.

[14] L. F. Gonzalez-Perez, L. C. Yllescas-Calderon, and R. Parra-Michel, "Parallel and Configurable Turbo Decoder Implementation for 3GPP-LTE," in *2013 Int. Conf. Reconfigurable Comput. FPGAs*, pp. 1–6, IEEE, Dec 2013.

[15] O. Muller, A. Baghdadi, and M. J, "From Application to ASIP-based FPGA Prototype: a Case Study on Turbo Decoding," in *2008 19th IEEE/IFIP Int. Symp. Rapid Syst. Prototyp.*, no. Figure 1, pp. 128–134, IEEE, Jun 2008.

[16] P. Murugappa, J.-N. Bazin, A. Baghdadi, and M. Jezequel, "FPGA Prototyping and Performance Evaluation of Multi-standard Turbo/LDPC Encoding and Decoding," *2012 23rd IEEE Int. Symp. Rapid Syst. Prototyp.*, pp. 143–148, Oct 2012.

[17] 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. Syst.*, vol. 21, pp. 14–22, Jan 2013.

[18] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and Sub-optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding," *Eur. Trans. Telecommun.*, vol. 8, pp. 119–125, Mar 1997.

[19] T. Ilnseher, F. Kienle, C. Weis, and N. Wehn, "A 2.15GBit/s Turbo Code Decoder for LTE Advanced Base Station Applications," in *2012 7th Int. Symp. Turbo Codes Iterative Inf. Process.*, (Gothenburg), pp. 21–25, IEEE, Aug 2012.

[20] G. Masera, G. Piccinini, M. Roch, and M. Zamboni, "VLSI

Architectures for Turbo Codes," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 7, pp. 369–379, Sep 1999.

[21] J. Steensma and C. Dick, "FPGA Implementation of a 3GPP Turbo Codec," in *Conf. Rec. Thirty-Fifth Asilomar Conf. Signals, Syst. Comput.*, vol. 1, pp. 61–65 vol.1, IEEE, Nov 2001.

[22] 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 2002 Cust. Integr. Circuits Conf. (Cat. No.02CH37285)*, (Orlando, FL, USA), pp. 389–392, IEEE, May 2002.

[23] X. Zeng and Z. Hong, "Design and Implementation of a Turbo Decoder for 3G W-CDMA Systems," *IEEE Trans. Consum. Electron.*, vol. 48, pp. 284–291, May 2002.

[24] G. Prescher, T. Gemmeke, and T. Noll, "A Parametrizable Low-Power High-Throughput Turbo-Decoder," in *Proceedings. IEEE Int. Conf. Acoust. Speech, Signal Process.*, vol. 5, (Philadelphia), pp. 25–28, IEEE, Mar 2005.

[25] Z. Cui and Z. Wang, "A 170 Mbps (8176, 7156) Quasi-Cyclic LDPC Decoder Implementation with FPGA," in *2006 IEEE Int. Symp. Circuits Syst.*, no. x, p. 4, IEEE, May 2006.

[26] C.-C. Wong, Y.-Y. Lee, and H.-C. Chang, "A 188-size 2.1mm2 Reconfigurable Turbo Decoder Chip with Parallel Architecture for 3GPP LTE System," in *VLSI Circuits, 2009 Symp.*, (Kyoto), pp. 288–289, Jun 2009.

[27] M. I. Del Barco, G. N. Maggio, D. a. Morero, J. Fernández, F. Ramos, H. S. Carrer, and M. R. Hueda, "FPGA Implementation of High-speed Parallel Maximum A Posteriori (MAP) Decoders," in *Proc. Argentine Sch. Micro-Nanoelectronics, Technol. Appl. 2009, EAMTA 2009*, no. 1, pp. 98–102, 2009.

[28] Y. Sun, Y. Zhang, J. Hu, and Z. Zhang, "FPGA Implementation of Nonbinary Quasi-cyclic LDPC Decoder Based on EMS Algorithm," in *2009 Int. Conf. Commun. Circuits Syst.*, pp. 1061–1065, IEEE, Jul 2009.

[29] M. May, T. Ilnseher, N. Wehn, and W. Raab, "A 150Mbit/s 3GPP LTE Turbo Code Decoder," in *Proc. Conf. Des. Autom. Test Eur.*, (Dresden, Germany), pp. 1420–1425, European Design and Automation Association, Mar 2010.

[30] V. A. Chandrasetty and S. M. Aziz, "FPGA Implementation of High Performance LDPC Decoder Using Modified 2-Bit Min-Sum Algorithm," in *2010 Second Int. Conf. Comput. Res. Dev.*, pp. 881–885, IEEE, 2010.

[31] Y. Sun and J. R. Cavallaro, "Efficient Hardware Implementation of a Highly-parallel 3GPP LTE/LTE-advance Turbo Decoder," *Integr. VLSI J.*, vol. 44, pp. 305–315, Sep 2011.

[32] 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, pp. 8–17, Jan 2011.

[33] X. Chen, J. Kang, S. Lin, and V. Akella, "Memory System Optimization for FPGA-Based Implementation of Quasi-Cyclic LDPC Codes Decoders," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 58, pp. 98–111, Jan 2011.

[34] S. Belfanti, C. Roth, M. Gautschi, C. Benkeser, and Q. Huang, "A 1Gbps LTE-advanced Turbo-decoder ASIC in 65nm CMOS," in *VLSI Circuits (VLSIC), 2013 Symp.*, (Kyoto), pp. C284–C285, Jun 2013.

[35] R. Shrestha and R. Paily, "Design and Implementation of a High Speed MAP Decoder Architecture for Turbo Decoding," in *2013 26th Int. Conf. VLSI Des. 2013 12th Int. Conf. Embed. Syst.*, pp. 86–91, IEEE, Jan 2013.

[36] S. Mhaske, D. Uliana, H. Kee, T. Ly, A. Aziz, and P. Spasojevic, "A 2.48Gb/s FPGA-based QC-LDPC Decoder: An Algorithmic Compiler Implementation," in *2015 36th IEEE Sarnoff Symp.*, no. 1, pp. 88–93, IEEE, Sep 2015.

[37] A. Li, L. Xiang, T. Chen, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "VLSI Implementation of Fully Parallel LTE Turbo Decoders," *IEEE Access*, vol. 4, pp. 323–346, Jan 2016.

[38] R. G. Maunder, "A Fully-Parallel Turbo Decoding Algorithm," *IEEE Trans. Commun.*, vol. 63, pp. 2762–2775, Aug 2015.

[39] C. Rachinger, J. B. Huber, and R. R. Muller, "Comparison of Convolutional and Block Codes for Low Structural Delay," *IEEE Trans. Commun.*, vol. 63, pp. 4629–4638, Dec 2015.

[40] "IEEE Standard for Local and Metropolitan Area Networks. Part 16: Air Interface for Fixed Broadband Wireless Access Systems," 2004.

[41] C. E. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. J.*, vol. 27, no. 4, pp. 623–656, 1948.

[42] L. Kong, S. X. Ng, R. Maunder, and L. Hanzo, "Maximum-Throughput Irregular Distributed Space-Time Code for Near-Capacity Cooperative Communications," *IEEE Trans. Veh. Technol.*, vol. 59, pp. 1511–1517, Mar 2010.

[43] Y. Polyanskiy, H. V. Poor, and S. Verdu, "Channel Coding Rate in the Finite Blocklength Regime," *IEEE Trans. Inf. Theory*, vol. 56, pp. 2307–2359, May 2010.

[44] Y. Polyanskiy, H. V. Poor, and S. Verdu, "Minimum Energy to Send k Bits with and without Feedback," in *2010 IEEE Int. Symp. Inf. Theory*, pp. 221–225, IEEE, Jun 2010.

[45] Y. Polyanskiy, H. V. Poor, and S. Verdú, "New Channel coding Achievable Bounds," *Proc. {IEEE} Intl. Symp. Inform. Theory*, vol. 5, pp. 1763–1767, Jul 2008.

[46] C. Berrou and A. Glavieux, "Near Optimum Error Correcting Coding and Decoding: Turbo-codes," *IEEE Trans. Commun.*, vol. 44, pp. 1261–1271, Oct 1996.

[47] 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 '95*, vol. 2, (Seattle, WA, USA), pp. 1009–1013, IEEE, Jun 1995.

[48] Y. Wu and B. Woerner, "The Influence of Quantization and Fixed Point Arithmetic upon the BER Performance of Turbo Codes," in *1999 IEEE 49th Veh. Technol. Conf. (Cat. No.99CH36363)*, vol. 2, (Houston, TX), pp. 1683–1687, May 1999.

[49] P.-Y. Wu and S. Pisuk, "Implementation of a low complexity, low power, integer-based turbo decoder," in *GLOBECOM'01. IEEE Glob. Telecommun. Conf. (Cat. No.01CH37270)*, vol. 2, (San Antonio, TX), pp. 946–951, IEEE, Nov 2001.

[50] Y. Wu, B. Woerner, and T. Blankenship, "Data Width Requirements in SISO Decoding with Module Normalization," *IEEE Trans. Commun.*, vol. 49, pp. 1861–1868, Nov 2001.

[51] 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 Relat. Top.*, (Brest, France), pp. 351–354, Sep 2000.

[52] Altera Corporation, "ALTPLL (Phase-Locked Loop) IP Core User Guide," 2014.

[53] Altera Corporation, "RAM Megafunction User Guide," Dec 2008.

[54] Altera Corporation, "Altera Product Catalog," p. 100, 2014.

[55] C. H. Janakiram and K. N. H. Srinivas, "An Efficient Technique for Parallel CRC Generation," vol. 3, no. 12, pp. 9761–9765, Dec 2014.

[56] E. Stavinov, "A Practical Parallel CRC Generation Method," Jan 2010.

[57] Altera Corp., "Quartus II Handbook Version 13.1," 2013.

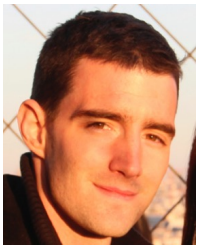
[58] G. Description, "Summary of Virtex-5 FPGA Features," vol. 100, 2009.

[59] R. Shrestha and R. Paily, "A Novel State Metric Normalization Technique for High-throughput Maximum-a-posteriori-probability Decoder," in *2013 Int. Conf. Adv. Comput. Commun. Informatics*, pp. 903–907, IEEE, Aug 2013.

[60] C. Leroux, C. Jego, P. Adde, and M. Jezequel, "Towards Gb/s turbo decoding of product code onto an FPGA device," in *2007 IEEE Int. Symp. Circuits Syst.*, pp. 909–912, IEEE, May 2007.
[61] J. Liang, R. Tessier, and D. Goeckel, "A Dynamically-Reconfigurable, Power-Efficient Turbo Decoder," in *12th Annu. IEEE Symp. Field-Programmable Cust. Comput. Mach.*, pp. 91–100, IEEE, Jan 2004.
[62] Altera Corperation, "Turbo IP Core - User Guide," 2015.
[63] XILINX, "LogiCORE IP 3GPP Mixed Mode Turbo Product Guide," 2014.
[64] E. Universal and M. A. Co, "ETSI TS 1 136 306," *Etsi*, vol. V13.0.0, 2015.
[65] R. Kannan, "The KR-Benes Network: A Control-optimal Rearrangeable Permutation Network," *IEEE Trans. Comput.*, vol. 54, pp. 534–544, May 2005.

An Li received his first class honors BEng degree in Electronic Engineering from the University of Southampton in 2011 and his MPhil degree from the University of Cambridge in 2012. He is currently a PhD student in Wireless Communication research group in the University of Southampton. His research interests include parallel turbo decoding algorithms and their implementations upon VLSI, FPGA and GPGPU.

Peter Hailes studied Electronic Engineering with Mobile and Secure Systems with the department of Electronics and Computer Science at the University of Southampton, and graduated with a first-class masters degree in 2013. He then stayed on to undertake research towards a Ph.D. in advanced hardware implementations of LDPC decoders. His other research interests include field-programmable gate arrays, error correction coding, embedded hardware/software design and high-level software development.



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

Bashir M. Al-Hashimi (M99-SM01-F09) is a Professor of Computer Engineering and Dean of the Faculty of Physical Sciences and Engineering at University of Southampton, UK. He is ARM Professor of Computer Engineering and 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 published over 300 technical papers, authored or co-authored 5 books and has graduated 31 PhD students.

Lajos Hanzo (FREng, FIEEE, FIET, Eurasip Fellow, RS Wolfson Fellow, www-mobile.ecs.soton.ac.uk) holds the Chair of Telecommunications at Southampton University, UK. He co-authored 1500+ IEEE Xplore entries, 20 IEEE Press & Wiley books, graduated 100+ PhD students and has an H-index of 59.