# A Scalable Turbo Decoding Algorithm for High-Throughput Network-on-Chip Implementation

Ra'ed Al-Dujaily, An Li, Robert G. Maunder, Terrence Mak, Bashir M. Al-Hashimi, and Lajos Hanzo

*Southampton Wireless, ECS, University of Southampton, Southampton SO17 1BJ, U.K.*

***Abstract*—Wireless communication at near-capacity transmission throughputs is facilitated by employing sophisticated Error Correction Codes (ECCs), such as turbo codes. However, real-time communication at high transmission throughputs is only possible if the challenge of implementing turbo decoders having equally high processing throughputs can be overcome. Furthermore, in many applications, turbo decoders are required to have the flexibility of supporting a wide variety of turbo code parametrizations. This motivates the implementation of turbo decoders using Networks-on-Chip (NoCs), which facilitate flexible and high-throughput parallel processing. However, turbo decoders conventionally operate on the basis of the Logarithmic Bahl-Cocke-Jelinek-Raviv (Log-BCJR) algorithm, which has an inherently-serial nature, owing to its data dependencies. This limits the exploitation of the NoC's computing resources, particularly as the size of the NoC is scaled up. Motivated by this, we propose a novel turbo decoder algorithm, which eliminates the data dependencies of the Log-BCJR algorithm and therefore has an inherently-parallel nature. We show that by jointly optimizing the proposed algorithm with the NoC architecture, a significantly improved utility of the available computing resources is achieved. Owing to this, our proposed turbo decoder achieves a factor of up to 2.13 higher processing throughput than a Log-BCJR benchmarker.**

***Index Terms*—Turbo codes, BCJR, network-on-chip, performance evaluation**

## List of Acronyms

| | |
|---|---|
| **3GPP** | 3rd Generation Partnership Project |
| **ASIC** | Application Specific Integrated Circuit |
| **ASIP** | Application-Specific Instruction-set Processor |
| **AWGN** | Additive White Gaussian Noise |
| **BCJR** | Bahl-Cocke-Jelinek-Raviv |
| **BER** | Bit Error Rate |
| **BPSK** | Binary Phase Shift Keying |
| **CC** | Clock Cycle |
| **DOR** | Dimension-Ordered Routing |
| **ECC** | Error Correction Code |
| **FIFO** | First In First Out |
| **FPGA** | Field Programmable Gate Array |
| **GPU** | Graphical Processing Unit |
| **IC** | Integrated Circuit |
| **IP** | Intellectual Property |
| **LDPC** | Low-Density Parity-Check |
| **LLR** | Log-Likelihood Ratio |
| **Log-BCJR** | Logarithmic Bahl-Cocke-Jelinek-Raviv |
| **LTE** | Long-Term Evolution |
| **NI** | Network Interface |
| **NoC** | Network on Chip |
| **RA** | Repeat Accumulate |
| **WiMAX** | Worldwide Interoperability for Microwave Access |

## List of Symbols

| | |
|---|---|
| $N$ | Number of bits in each message frame |
| $M$ | Number of trellis states |
| $P$ | Number of windows in each frame |
| $K$ | Number of bits in each window |
| $I$ | Number of iterations |
| $\Pr$ | Probability |
| $E_b/N_0$ | Normalized signal to noise ratio per bit |
| $E_b$ | Energy per bit |
| $N_0$ | Noise power spectral density |
| $\boldsymbol{b}_1$ | Frame of message bits |
| $\boldsymbol{b}_2$ | Frame of parity bits |
| $\boldsymbol{b}_3$ | Frame of systematic bits |
| $\bar{\boldsymbol{b}}_1$ | Frame of received message soft bits |
| $\bar{\boldsymbol{b}}_2$ | Frame of received parity soft bits |
| $\bar{\boldsymbol{b}}_3$ | Frame of received systematic soft bits |
| $\boldsymbol{\alpha}_k$ | Forward state metrics |
| $\boldsymbol{\beta}_k$ | Backward state metrics |
| $\gamma_k$ | *A priori* branch metrics |
| $\delta_k$ | *A posteriori* branch metrics |
| $\pi$ | Interleaver |
| $\pi^{-1}$ | De-interleaver |

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://eprints.soton.ac.uk/397738/.

| | |
|---|---|
| **a** | Superscript denoting *a priori* information |
| **e** | Superscript denoting extrinsic information |
| **u** | Superscript denoting upper decoder |
| **l** | Superscript denoting lower decoder |
| **p** | Superscript denoting *a posteriori* information |
| $k$ | Subscript denoting the bit index in a frame |

## I. Introduction

During the last two decades, wireless communication has been revolutionized by turbo codes [1] and other iterative ECCs, such as Low Density Parity Check (LDPC) [2] and Repeat Accumulate (RA) codes [3]. Like these other iterative ECCs, turbo codes provide resilience to the transmission errors that are caused by noise, interference and fading during wireless communication. This is achieved by using a turbo encoder to process each frame of message bits before their transmission, and then employing a corresponding turbo decoder in the receiver to detect and correct transmission errors. In contrast to non-iterative ECCs, the error correction capability of turbo codes is so strong that they facilitate reliable communication, even when employing a transmission throughput that closely approaches the theoretical capacity of the wireless channel. However, in order to facilitate real-time communication with a high *transmission* throughput, the turbo encoder and decoder must be designed to have equally high *processing* throughputs. This is a particular challenge when designing the turbo decoder, since its complexity is typically much higher than that of the encoder, which is almost insignificant in comparison. Furthermore, in many wireless communication applications, the design of turbo decoders with high processing throughputs is complicated by the requirement to support numerous different turbo code parametrizations. For example, the turbo decoder employed by the Long Term Evolution (LTE) standard [4] for cellular telephony is required to support 188 different parametrizations, each corresponding to a different message frame length in the range spanning from 40 to 6144 bits. Furthermore, state-of-the-art mobile devices typically support numerous wireless communication standards, each employing different turbo code parametrizations, or different iterative ECCs altogether. This motivates the employment of high-throughput Application-Specific Instruction-set Processors (ASIPs), allowing mobile devices to flexibly employ the same hardware to implement various turbo decoders and other iterative ECC decoders.

In recent years, the continued scaling of integration technologies has enabled the implementation of hundreds or even thousands of ASIPs within a single Integrated Circuit (IC), hence facilitating flexible high-throughput processing. However, to avoid a bottleneck the structure of interconnecting these ASIPs has to be carefully designed. This becomes a particular challenge as the amount of inter-ASIP communication is increased beyond the bandwidth of classical models of on-chip interconnection, such as point-to-point and bus-based connections. This motivates a Network on Chip (NoC) architecture [5], which comprises a network of ASIPs, each having a router that is connected to those of the neighboring ASIPs. Communication among the ASIPs is achieved using packet-switching, whereby the messages are dynamically routed from the source ASIP to the destination ASIP along a path formed of interconnected routers. Recently, NoCs have been proposed [6]–[9] as the basis of multi-core ICs for implementing flexible, high-throughput turbo decoders. These efforts have focused on optimizing the NoC architecture to suit the Log-BCJR algorithm [10], [11], which is the basis of a conventional turbo decoder's operation. However, in this paper we show that the inherently-serial nature of the Log-BCJR algorithm's data dependencies results in a low utility of a NoC's computing resources, as the number of ASIPs is scaled up.

Against this background, this paper proposes a novel turbo decoding algorithm, which eliminates the data dependencies of the Log-BCJR algorithm. This grants the turbo decoder an inherently-parallel nature, which is better suited to NoC-implementations. In particular, we propose a novel technique for self-regulating the exchange of information within the NoC, in order to avoid congestion. More specifically, rather than adhering to rigidly-defined schedules like that of the conventional Log-BCJR turbo decoding algorithm, the operation of each ASIP in the NoC is adapted in response to the delivery of information across the NoC. Each ASIP makes the most beneficial and timely use of the delivered information, hence maximizing the quality of the information that it generates for delivery over the NoC. The delivery of that information to the connected ASIP stimulates its operation, causing the schedule to cascade organically, with the processing stimulating the networking and the networking stimulating the processing. Compared to the classic Log-BCJR benchmarker, the proposed turbo decoder achieves a significantly improved utility of the NoC's computing resources. Owing to this, our proposed turbo decoder achieves a significantly higher processing throughput than the Log-BCJR benchmarker, particularly as the number of ASIPs in the NoC is scaled up.

The rest of this paper is organized as follows. Section II reviews our previous work [12]–[15] on the fully-parallel turbo decoder, which forms the basis of the proposed algorithm. Following this, our novel NoC-optimized turbo decoding algorithm is proposed in Section III of this paper. Furthermore in Section IV of this paper, we propose a NoC architecture, which we jointly optimize with the proposed turbo decoding algorithm. We present our simulation results in Section V, before we offer our conclusions in Section VI.

## II. Background

In this section, we provide a background discussion on the computations performed by the fully-parallel turbo decoding algorithm of our previous work [12]–[15], which serve as the basis of the novel NoC-optimized turbo decoding algorithm of Section III. We commence by introducing our notation in Section II-A. The schematic of a turbo decoder is described in Section II-B. Following this, Section II-C describes the preprocessing invoked for initializing the turbo decoder. The proposed NoC-optimized turbo decoding algorithm of Section III operates on the basis of algorithmic blocks, which are described in Section II-D.

### A. Preliminaries

In this section, we introduce the notation used throughout this paper when referring to the operation of the LTE turbo encoder and the turbo decoder. The LTE turbo encoder [4] may be employed to encode a frame $[b^{\mathrm{u}}_{1,k}]^{N}_{k=1}$ comprising $N$ message bits, each having a binary value $b^{\mathrm{u}}_{1,k} \in \{0, 1\}$. Here, 188 different values in the range spanning from 40 to 6144 are supported for the frame length $N$. The message frame $[b^{\mathrm{u}}_{1,k}]^{N}_{k=1}$ is provided to an upper convolutional encoder. In response, this produces a frame $[b^{\mathrm{u}}_{2,k}]^{N}_{k=1}$ comprising $N$ parity bits, as well as a frame $[b^{\mathrm{u}}_{3,k}]^{N}_{k=1}$ comprising $N$ systematic bits. In addition to this, the upper convolutional encoder produces three termination message bits $[b^{\mathrm{u}}_{1,k}]^{N+3}_{k=N+1}$, as well as three termination parity bits $[b^{\mathrm{u}}_{2,k}]^{N+3}_{k=N+1}$. Meanwhile, the message frame $[b^{\mathrm{u}}_{1,k}]^{N}_{k=1}$ is interleaved, in order to obtain the interleaved message frame $[b^{\mathrm{l}}_{1,k}]^{N}_{k=1}$. This is provided to a lower convolutional encoder, which produces a frame $[b^{\mathrm{l}}_{2,k}]^{N}_{k=1}$ comprising $N$ parity bits, as well as three termination message bits $[b^{\mathrm{l}}_{1,k}]^{N+3}_{k=N+1}$ and three termination parity bits $[b^{\mathrm{l}}_{2,k}]^{N+3}_{k=N+1}$. Note that the lower convolutional encoder does not produce any systematic bits. The LTE turbo encoder has a coding rate of $R = N/(3N + 12)$ since it outputs a total of $(3N + 12)$ bits, namely $[b^{\mathrm{u}}_{2,k}]^{N}_{k=1}$, $[b^{\mathrm{u}}_{3,k}]^{N}_{k=1}$, $[b^{\mathrm{l}}_{2,k}]^{N}_{k=1}$, $[b^{\mathrm{u}}_{1,k}]^{N+3}_{k=N+1}$, $[b^{\mathrm{u}}_{2,k}]^{N+3}_{k=N+1}$, $[b^{\mathrm{l}}_{1,k}]^{N+3}_{k=N+1}$ and $[b^{\mathrm{l}}_{2,k}]^{N+3}_{k=N+1}$. Throughout the remainder of this paper, the superscripts 'u' and 'l' are used only when necessary to explicitly distinguish the upper and lower components of the turbo code, but they are omitted when the discussion applies equally to both.

Following their modulation and transmission over a wireless channel, the $(3N + 12)$ turbo-encoded bits may be demodulated and provided to the turbo decoder. Owing to the effect of noise in the wireless channel however, the demodulator will be uncertain of the correct values for these turbo-encoded bits. Therefore, instead of providing a hard-valued decision for each bit, the demodulator provides soft-valued decisions, in the form of *a priori* Logarithmic Likelihood Ratios (LLRs). More specifically, the *a priori* LLR pertaining to the bit $b_{j,k}$ is defined by

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

In this notation, the diacritical bar indicates a soft-valued decision, while the superscript 'a' corresponds to *a priori* information. In the remainder of this paper, the alternative superscripts 'e' and 'p' are used to indicate extrinsic and *a posteriori* information, respectively.

### B. Schematic

The algorithm proposed for the NoC implementation of the LTE turbo decoder may be characterized by the schematic of Figure 1, which is provided with $(3N+12)$ *a priori* LLRs, namely $[\bar{b}^{\mathrm{u,a}}_{2,k}]^{N}_{k=1}$, $[\bar{b}^{\mathrm{u,a}}_{3,k}]^{N}_{k=1}$, $[\bar{b}^{\mathrm{l,a}}_{2,k}]^{N}_{k=1}$, $[\bar{b}^{\mathrm{u,a}}_{1,k}]^{N+3}_{k=N+1}$, $[\bar{b}^{\mathrm{u,a}}_{2,k}]^{N+3}_{k=N+1}$, $[\bar{b}^{\mathrm{l,a}}_{1,k}]^{N+3}_{k=N+1}$ and $[\bar{b}^{\mathrm{l,a}}_{2,k}]^{N+3}_{k=N+1}$. As shown in Figure 1, these *a priori* LLRs are input to the algorithmic blocks arranged in two rows, each comprising $(N + 3)$ algorithmic blocks. Here, each algorithmic block operates on the basis of the LTE turbo code's state transition diagram, which is depicted in Figure 2.

Note that the first $N$ algorithmic blocks in each row of Figure 1 are interconnected through the interleaver. During the turbo decoding process, these algorithmic blocks operate as described in Section II-D. They exchange LLRs in an iterative manner, which is suited to the implementation of the NoC, as described in Section III. However, before commencing this iterative decoding process, it must initialized using a small amount of preprocessing. More specifically, the last three algorithmic blocks in each row are isolated and only have to be operated once, before the iterative decoding process commences. This may be performed externally to the NoC, as described in Section II-C.

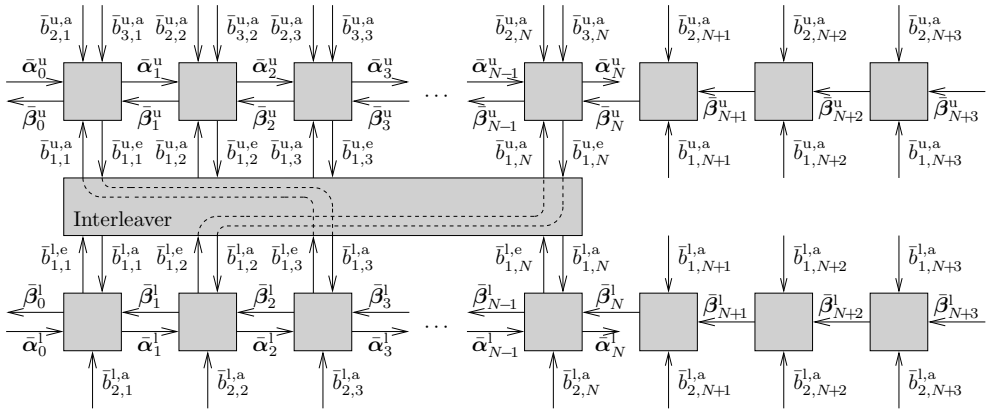



**Fig. 1:** Schematic characterizing the proposed turbo decoding algorithm for NoC implementation.

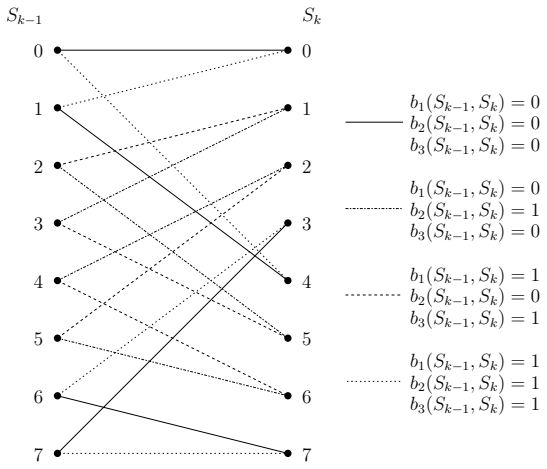



**Fig. 2:** State transition diagram of the LTE turbo code, depicting all legitimate transitions between the $M = 8$ possible values of the previous state $S_{k-1} \in \{0, 1, 2, \ldots, M-1\}$ and the subsequent state $S_k \in \{0, 1, 2, \ldots, M-1\}$. The binary value $c(S_{k-1}, S_k) \in \{0, 1\}$ indicates whether a transition between the states $S_{k-1}$ and $S_k$ is legitimate. Each legitimate transition implies message, parity and systematic bit values of $b_1(S_{k-1}, S_k) \in \{0, 1\}$, $b_2(S_{k-1}, S_k) \in \{0, 1\}$ and $b_3(S_{k-1}, S_k) \in \{0, 1\}$, respectively.

### C. Preprocessing

Before iterative decoding is commenced, the last three algorithmic blocks in each row of Figure 1 apply some preprocessing to the LLRs $[\bar{b}^{\mathrm{a}}_{1,k}]_{k=N+1}^{N+3}$ and $[\bar{b}^{\mathrm{a}}_{2,k}]_{k=N+1}^{N+3}$, which pertain to the termination bits. These LLRs are converted into the state metrics $[\bar{\boldsymbol{\beta}}_k]_{k=N}^{N+2}$, where $\bar{\boldsymbol{\beta}}_k = [\bar{\beta}_k(S_k)]_{S_k=0}^{M-1}$ pertains to the $M = 8$ possible values of the state $S_k$, as shown in Figure 2. This is achieved using the conventional Log-BCJR's backwards recursion, which commences with the operation of the algorithmic block having the index $k = N + 3$, before activating the block with index $k = N + 2$ and then concluding with the operation of the block having the index $k = N + 1$. When operating each of these algorithmic blocks, an *a priori* metric is computed for each of the transitions in Figure 2, according to

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

where the notation $b_j(S_{k-1}, S_k)$ is defined in the caption of Figure 2. Following this, an extrinsic backwards metric is computed for each of the $M = 8$ possible states in Figure 2, according to

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

where the notation $c(S_{k-1}, S_k)$ is defined in the caption of Figure 2. This backwards recursion is initialized using $\bar{\boldsymbol{\beta}}_{N+3} = [0, -\infty, -\infty, \ldots, -\infty]$, since termination guarantees having a final state of $S_{N+3} = 0$. Note that (3) employs the Jacobian logarithm of [16], which is defined for two operands as

$$\max^{*}(\bar{\delta}_1, \bar{\delta}_2) = \max(\bar{\delta}_1, \bar{\delta}_2) + \ln\left(1 + e^{-|\bar{\delta}_1 - \bar{\delta}_2|}\right) \qquad (4)$$

and may be extended to more operands by exploiting its associative property. The above process generates the state metrics $\bar{\boldsymbol{\beta}}_N$ of Figure 1, which are used throughout the iterative decoding process of Section III. Note that the state metrics $\bar{\boldsymbol{\alpha}}_0 = [0, -\infty, -\infty, \ldots, -\infty]$ are also used throughout the iterative decoding process, since the initial state of $S_0 = 0$ is guaranteed.

### D. Algorithmic block operation

As described in Section II-B, the proposed turbo decoder algorithm operates the first $N$ algorithmic blocks in each row according to a novel iterative decoding schedule, as will be described in Section III. Regardless of when an algorithmic block is activated during this schedule, its operation follows the same process, as detailed in the following discussion.

Whenever the block in the upper row having the index $k \in \{1, 2, 3, \ldots, N\}$ is operated during this iterative process, it accepts $L = 3$ *a priori* LLRs as inputs, as shown in Figure 1. More specifically, the block accepts the *a priori* message LLR $\bar{b}_{1,k}^{\mathrm{u,a}}$ that has been most recently provided by the interleaver, while the *a priori* parity and systematic LLRs $\bar{b}_{2,k}^{\mathrm{u,a}}$ and $\bar{b}_{3,k}^{\mathrm{u,a}}$ are accepted from the demodulator. By contrast, the block in the lower row having the index $k \in \{1, 2, 3, \ldots, N\}$ accepts only $L = 2$ *a priori* LLRs when it is operated, namely the *a priori* message and the *a priori* parity LLRs $\bar{b}_{1,k}^{\mathrm{l,a}}$ and $\bar{b}_{2,k}^{\mathrm{l,a}}$. Furthermore, each block having the index $k \in \{1, 2, 3, \ldots, N\}$ accepts the vector of *a priori* forward state metrics $\bar{\boldsymbol{\alpha}}_{k-1} = [\bar{\alpha}_{k-1}(S_{k-1})]_{S_{k-1}=0}^{M-1}$, as well as the vector of *a priori* backward state metrics $\bar{\boldsymbol{\beta}}_k = [\bar{\beta}_k(S_k)]_{S_k=0}^{M-1}$ that have been most recently provided by the neighboring algorithmic blocks. Note that at the start of the iterative decoding process, zero values are employed for the above-mentioned inputs if the interleaver or the neighboring blocks have not yet provided any updated values. The operation of each block having the index $k \in \{1, 2, 3, \ldots, N\}$ is completed using the equations provided in (5) – (8). More specifically, (5) is employed for combining the inputs, in order to obtain an *a posteriori* metric $\bar{\delta}(S_{k-1}, S_k)$ for each transition in the state transition diagram of Figure 2. These *a posteriori* transition metrics are then combined by (6), (7) and (8), in order to produce the vector of extrinsic forward state metrics $\bar{\boldsymbol{\alpha}}_k = [\bar{\alpha}_k(S_k)]_{S_k=0}^{M-1}$, the vector of extrinsic backward state metrics $\bar{\boldsymbol{\beta}}_{k-1} = [\bar{\beta}_{k-1}(S_{k-1})]_{S_{k-1}=0}^{M-1}$ and the extrinsic message LLR $\bar{b}_{1,k}^{\mathrm{e}}$, respectively. These extrinsic state metrics are provided for the neighboring algorithmic blocks, while the extrinsic message LLR $\bar{b}_{1,k}^{\mathrm{e}}$ is provided for the interleaver $\pi$, so that it can be delivered to a block in the other row and used as an *a priori* message LLR, where $\bar{b}_{1,k}^{\mathrm{l,a}} = \bar{b}_{1,\pi(k)}^{\mathrm{u,e}}$.

## III. Turbo decoding algorithm proposed for NoC implementation

In this section, we propose our novel NoC-optimized algorithm for the implementation of the LTE turbo decoder [4]. This is achieved by scheduling the computations of the algorithmic blocks described in Section II-D in a manner that is particularly suited to NoC operation. This is possible, since the algorithmic blocks of Section II-D are not bound by a strict scheduling that requires their operation according to forward and backward recursions, as in the conventional Log-BCJR turbo decoding algorithm [17]. In our previous work [12]–[15], this property was exploited to achieve fully-parallel operation, in which all algorithmic blocks are operated concurrently, in every clock cycle. More specifically, while [12] introduced the fully-parallel turbo decoding algorithm, [13]–[15] considered the implementation of that algorithm in Application Specific Integrated Circuit (ASIC), Field Programmable Gate Array (FPGA) and Graphical Processing Unit (GPU) applications, respectively. In contrast to this previous work, the novel scheduling proposed in this section is specifically designed for mapping onto an NoC, in which each ASIP performs the operations of Section II-D for different algorithmic blocks in different Clock Cycles (CCs). In particular, the proposed scheduling self-regulates the exchange of information within the NoC, in order to avoid congestion, where the operation of each ASIP in the NoC is adapted in response to the arrival of information delivered by the NoC. More specifically, rather than scheduling the operation of the algorithmic blocks according to strict forward and backward recursions, the proposed turbo decoder algorithm schedules their operation according to the exchange of information within the NoC. Here, the operation of the algorithmic blocks generates information for delivery over the NoC, while the delivery of that information to the connected algorithmic blocks stimulates their operation and so on. In this way, the processing stimulates the networking and the networking stimulates the processing, causing the schedule to grow organically.

As described in Section II-B, the iterative decoding process of the proposed turbo-decoding algorithm may be completed using an NoC, comprising an interconnected network of Intellectual Property (IP) cores. Here, each IP core is responsible for the operation of a different subset of the first $N$ algorithmic blocks in each row of Figure 1. We refer to these subsets as windows, with each window comprising $K$ number of adjacent algorithmic blocks, as shown in Figure 3. We assume that the IP cores are sufficiently powerful or are specifically designed for requiring only a single clock cycle for completing the processing of (5) – (8) for a single algorithmic block, as detailed in Section II-D. Note however that this implies that multiple algorithmic blocks within the same window cannot be operated concurrently. During the first $(2K-1)$ CCs, the iterative decoding process is initialized by performing a forward recursion and then a backward recursion within each window of the upper row. More specifically, as shown in Figure 3(a), the forward recursion invokes the operations of Section II-D for the $j^{\text{th}}$ algorithmic block in each window of the upper row during the $j^{\text{th}}$ clock cycle, where $j \in \{1, 2, 3, \ldots, K-1\}$. Following this, the backward recursion invokes the operations of Section II-D for the $(2K-j)^{\text{th}}$ algorithmic block during the $j^{\text{th}}$ clock

The turbo decoding algorithm proposed for NoC implementation.

$$\bar{\delta}_k(S_{k-1}, S_k) = \left[\sum_{j=1}^{L} \left[b_j(S_{k-1}, S_k) \cdot \bar{b}^{\mathrm{a}}_{j,k}\right]\right] + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k) \tag{5}$$

$$\bar{\alpha}_k(S_k) = \left[\max^{*}_{\{S_{k-1}|c(S_{k-1},S_k)=1\}} \left[\bar{\delta}_k(S_{k-1}, S_k)\right]\right] - \bar{\beta}_k(S_k) \tag{6}$$

$$\bar{\beta}_{k-1}(S_{k-1}) = \left[\max^{*}_{\{S_k|c(S_{k-1},S_k)=1\}} \left[\bar{\delta}_k(S_{k-1}, S_k)\right]\right] - \bar{\alpha}_{k-1}(S_{k-1}) \tag{7}$$

$$\bar{b}^{\mathrm{e}}_{j,k} = \left[\max^{*}_{\{(S_{k-1},S_k)|b_j(S_{k-1},S_k)=1\}} \left[\bar{\delta}_k(S_{k-1}, S_k)\right]\right] - \left[\max^{*}_{\{(S_{k-1},S_k)|b_j(S_{k-1},S_k)=0\}} \left[\bar{\delta}_k(S_{k-1}, S_k)\right]\right] - \bar{b}^{\mathrm{a}}_{j,k} \tag{8}$$

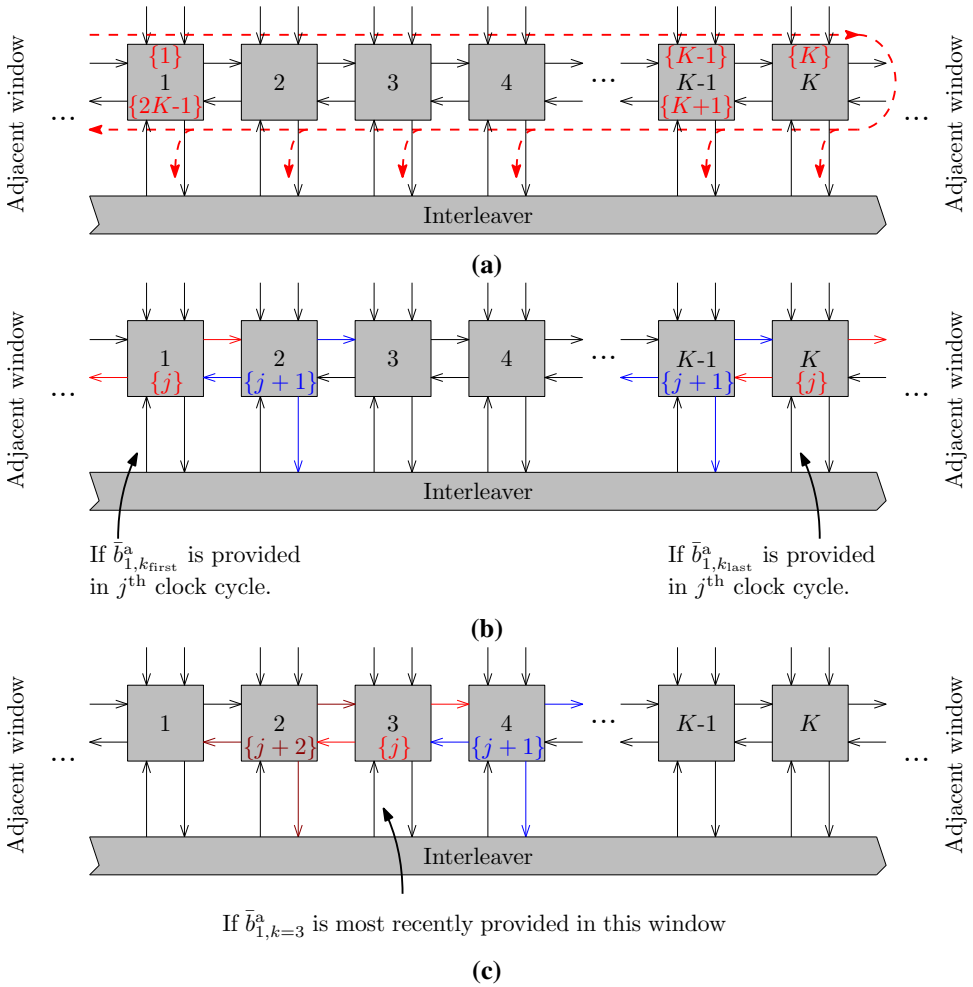



**Fig. 3:** Data flow within a single window of the proposed NoC implementation for the cases of (a) performing the first half-iteration, (b) participating in the dynamically regulated iterative decoding process and (c) remaining active when the window would otherwise be idle. The clock cycle index $j$ is shown in the curly brackets.

cycle, where $j \in \{K, K+1, K+2, \ldots, 2K-1\}$. Note that the extrinsic LLRs $\bar{b}^{\mathrm{u,e}}_{1,k}$ that are generated during the forward recursion are discarded and that only those obtained during the backward recursion are provided to the interleaver. In the NoC, interleaving is achieved by appropriately routing the extrinsic LLRs between IP

cores, which may take several CCs, depending both on the path length and on the level of congestion.

As shown in Figure 3(b), whenever the interleaving of an extrinsic LLR is completed throughout the remainder of the iterative decoding process, the resultant *a priori* LLR $\bar{b}^{\mathrm{a}}_{1,k}$ is immediately processed in that $j^{\mathrm{th}}$ clock cycle by the corresponding algorithmic block, as described in Section II-D. However, the resultant extrinsic LLR $\bar{b}^{\mathrm{e}}_{1,k}$ is not provided to the interleaver, since it is extrinsic to $\bar{b}^{\mathrm{a}}_{1,k}$ and does not benefit from the corresponding *a priori* information. Instead, the resultant extrinsic state metric vectors $\bar{\boldsymbol{\alpha}}_k$ and $\bar{\boldsymbol{\beta}}_{k-1}$ are provided for the adjacent algorithmic blocks, having the indices of $k+1$ and $k-1$, respectively. In the following $(j+1)^{\mathrm{st}}$ clock cycle of Figure 3(b), one of these extrinsic state metric vectors is processed as described in Section II-D by the corresponding adjacent algorithmic block, provided that it resides within the same window as the block with index $k$. Here, a random selection is employed, if both adjacent blocks reside within the same window as the block with index $k$. Finally, it is the extrinsic LLR $\bar{b}^{\mathrm{e}}_{1,k+1}$ or $\bar{b}^{\mathrm{e}}_{1,k-1}$ produced by this adjacent algorithmic block that is provided for the interleaver. In this way, the delivery of each LLR through the interleaver stimulates the generation of another LLR, dynamically regulating the level of congestion within the NoC.

Owing to the delays imposed by the interleaver, there may be CCs in which none of the algorithmic blocks within a particular window is engaged in the dynamically-regulated process described above. During these CCs, it is still beneficial to perform the operations of Section II-D for the algorithmic blocks within the window, so that information may be propagated using the extrinsic state metrics. In this way, the IP core dedicated to each window is able to perform useful computations in every clock cycle, achieving a hardware resource utility of 100%. Note however that the interleaver is not provided with the extrinsic LLRs that are produced when operating algorithmic blocks in this way, since this would cause excessive congestion in the NoC. As shown in Figure 3(c), during these otherwise-unutilized CCs, priority is given to propagating the information within the *a priori* LLR $\bar{b}^{\mathrm{a}}_{1,k}$ that has been most recently provided by the interleaver. This is achieved by performing the operations of Section II-D for the algorithmic blocks within the window in order of decreasing proximity to the $k^{\mathrm{th}}$ block, alternating between the blocks in the forward and backward directions. This propagation is halted once all of the blocks in the window have been operated, or if the interleaver provides the window with another *a priori* LLR, whereupon the above-described process is restarted. When not employing otherwise-unutilized CCs for propagating the information supplied by *a priori* LLRs, these CCs may be used to propagate information supplied by the adjacent windows. More specifically, when the algorithmic blocks at the adjoining ends of the adjacent windows are operated as described in Section II-D, they will provide updated *a priori* state metrics $\bar{\boldsymbol{\alpha}}_{k_{\mathrm{first}}-1}$ or $\bar{\boldsymbol{\beta}}_{k_{\mathrm{last}}}$. Here, $k_{\mathrm{first}}$ and $k_{\mathrm{last}}$ are the indices of the first and last algorithmic blocks in the window considered. The information provided by $\bar{\boldsymbol{\alpha}}_{k_{\mathrm{first}}-1}$ may be propagated by using a forward recursion to perform the operations of Section II-D for the algorithmic blocks in the window in ascending order of index $k$. Likewise, a backward recursion may be employed to operate the algorithmic blocks in decreasing order of index $k$ and propagate the information provided by $\bar{\boldsymbol{\beta}}_{k_{\mathrm{last}}}$. These recursions are halted once all of the blocks in the window have been operated, or if the window is provided with an updated *a priori* LLR or state metric $\bar{\boldsymbol{\alpha}}_{k_{\mathrm{first}}-1}$ or $\bar{\boldsymbol{\beta}}_{k_{\mathrm{last}}}$, whereupon the corresponding above-mentioned process is restarted.

Following the completion of the iterative decoding process, the *a posteriori* LLRs $\{\bar{b}^{\mathrm{u,p}}_{1,k}\}^{N}_{k=1}$ may be obtained according to $\bar{b}^{\mathrm{u,p}}_{1,k} = \bar{b}^{\mathrm{u,a}}_{1,k} + \bar{b}^{\mathrm{u,e}}_{1,k}$. A hard-decision may be imposed upon these *a posteriori* LLRs in order to obtain the bits $\{\hat{b}^{\mathrm{u}}_{1,k}\}^{N}_{k=1}$, where each bit value is obtained as the result of a binary test $\hat{b}^{\mathrm{u}}_{1,k} = \bar{b}^{\mathrm{u,p}}_{1,k} > 0$.

## IV. The NoC proposed for turbo decoder implementation

In this section, we propose an NoC platform for implementing the proposed scalable turbo decoding algorithm of Section III. We commence in Section IV-A by providing a brief introduction to the architecture of a typical NoC platform. Following this, Section IV-B describes the topology of the tiles in our proposed NoC platform and the mapping of the windows described in Section III onto these tiles. Finally, the NoC routing employed for implementing the interleaver is described in Section IV-C.

### A. NoC architecture

An NoC comprises an on-chip interconnection network between many identical nodes, each of which is so-called a tile. A tile comprises four main components, namely an IP core, a router, Network Interfaces (NIs) and links. The IP cores refer to a block of reusable components, which may be ASIPs, ASICs, FPGAs, CPUs, DSPs or memory, for example. In order to perform inter-tile communications, a router is employed and connected to the local IP core via an NI in each tile, while the

routers of different tiles are connected via links, which are typically on-chip wires.

The layout pattern of the interconnections between the on-chip components is referred to as the network topology. Many topologies exist in the literature, ranging from the simple shared bus [18] and ring [19] topologies, to more complicated application-specific topologies, such as those of [20], [21]. The selection of an appropriate topology depends on design constraints such as performance, area, power, locality of traffic and quality of service. Given a particular network topology, routing is responsible for choosing a path between any two communicating nodes, which are referred to as a source-destination pair. The careful selection of an appropriate routing algorithm is crucial for achieving good network performance. An effective routing algorithm has to carefully consider load balancing, throughput, latency, power consumption, fault-tolerance, deadlocks, livelocks and the limited hardware resources available on chip [22]. Although an abundance of routing algorithms have been proposed in the literature [23], [24], they are typically classified into three kinds, namely deterministic, oblivious and adaptive [23].

### *B. Topology and Mapping onto IP Cores*

As shown in Figure 4(a), we employ a two-dimensional (2D) mesh topology for implementing our proposed turbo decoder NoC, since its regular grid-like structure is particularly suited to fabrication [25], [26]. Furthermore, the proposed turbo decoder schematic of Figure 1 is also a regular 2D structure, which can therefore be readily mapped to an NoC employing a 2D mesh topology. As shown in Figure 4(a), a 2D mesh NoC having the dimensions of $X$ by $2Y$ may be invoked for decoding frames comprising $N$ bits that are decomposed into $P$ number of windows, where $XY = P$. Accordingly, the unshaded tiles $\in \{1, 2, 3, \ldots, XY\}$ and the shaded tiles $\in \{XY + 1, XY + 2, XY + 3, \ldots, 2XY\}$ respectively correspond to the upper and the lower decoders of Figure 1, wherein each tile processes one of the $P$ windows. Note that adjacent windows of algorithmic blocks are mapped to adjacent tiles in Figure 4(a), according to a pattern that meanders from side to side in the 2D mesh topology.

The 2D mesh is employed to interleave the extrinsic LLRs $\bar{b}^{\mathrm{e}}_{1,k}$ that are produced by the tiles of Figure 4(a). These are routed according to the approach of Section IV-C and delivered to the destination tile, where they are used as the *a priori* LLRs $\bar{b}^{\mathrm{a}}_{1,k}$. As described in Section IV-A, each tile includes an IP core, which comprises an ASIP processor and some memory blocks,

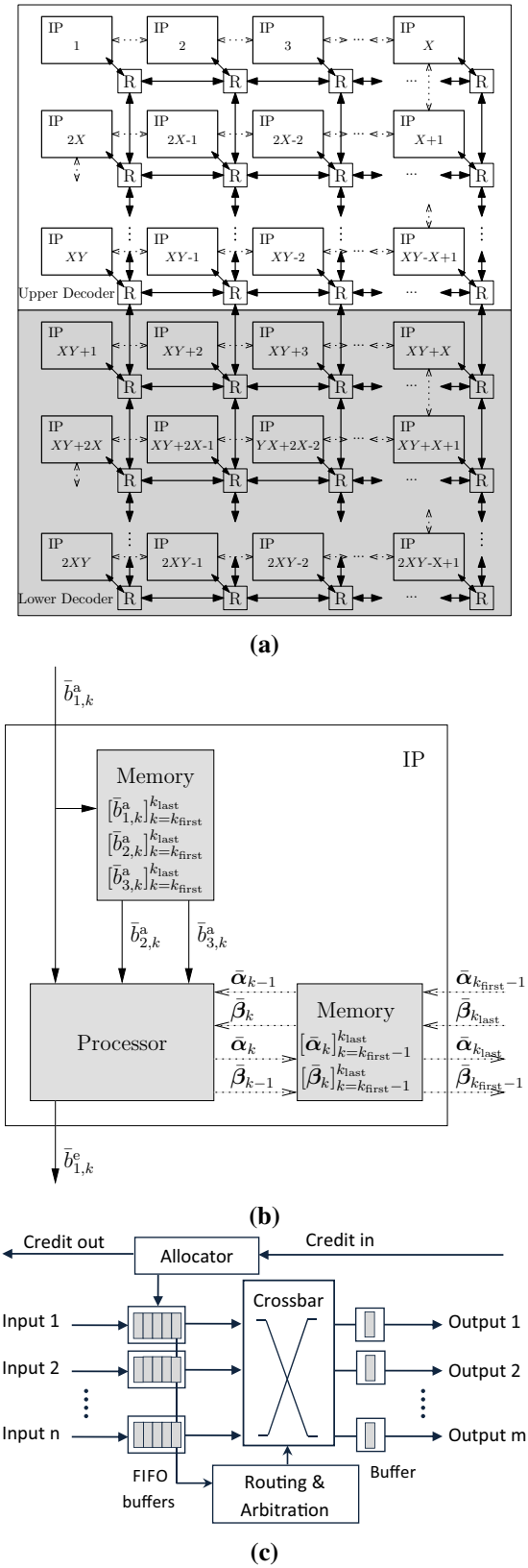



**Fig. 4:** (a) Schematic of the proposed 2D mesh NoC, having the dimensions of $X$ by $2Y$, (b) schematic of the IP core in each tile, (c) schematic of the router in each tile.

as shown in Figure 4(b). In each clock cycle, the ASIP performs the operations of Section II-D for a particular one of the algorithmic blocks in the corresponding window, depending on the delivery of the *a priori* LLRs $\bar{b}^{\mathrm{a}}_{1,k}$, according to the novel schedule described in Section III. More specifically, in each unshaded tile corresponding to the upper decoder, the ASIP processor is designed to process (5) – (8) using $L = 3$, as detailed in Section II-D. As demonstrated in [13], the ASIP may complete these calculations for a single algorithmic block within a single clock cycle using 75 adders (some of which are used for maximum or subtraction operations), where the critical path is formed by a chain of 6 adders. Furthermore, a block of memory is employed for storing all *a priori* message LLRs $\bar{b}^{\mathrm{u,a}}_{1,k}$, all parity LLRs $\bar{b}^{\mathrm{u,a}}_{2,k}$ and all *a priori* systematic LLRs $\bar{b}^{\mathrm{u,a}}_{3,k}$ for all algorithmic blocks in the corresponding window. By contrast, the ASIP processors of the shaded tiles corresponding to the lower decoder process (5) – (8) with only $L = 2$, requiring 74 adders with a critical path comprising 6 adders. Accordingly, these tiles do not require memory for storing *a priori* systematic LLRs $\bar{b}^{\mathrm{l,a}}_{3,k}$, as discussed in Section II-D. In addition to these LLRs, each IP core employs another block of memory for storing both the branch metrics $\bar{\delta}_k$, as well as the forward state metrics $\bar{\boldsymbol{\alpha}}_k$ and backward state metrics $\bar{\boldsymbol{\beta}}_{k-1}$, as shown in Figure 4(b). Most of these intermediate variables are produced and consumed within the same window of algorithmic blocks and hence within the same tile. However, as described in Section III, when the $k^{\mathrm{th}}$ = first and $k^{\mathrm{th}}$ = last algorithmic blocks in a particular window are operated, the resultant sets of $M$ backward state metrics $\bar{\boldsymbol{\beta}}_{k_{\mathrm{first}}-1}$ and $M$ forward state metrics $\bar{\boldsymbol{\alpha}}_{k_{\mathrm{last}}}$ must be provided to the preceding window and the following window, respectively. Compared to the single extrinsic LLRs that are produced at a time by each tile, these sets of state metrics comprise $M$ times as much data, where we have $M = 8$ in the case of the LTE turbo code. However, this data is only exchanged with a neighboring window and hence with a neighboring tile. Therefore, in order to avoid congestion on the 2D mesh, these sets of state metrics are exchanged using one-hop links between adjacent tiles in our proposed NoC implementation, which are shown as the dotted lines in Figure 4(a) and 4(b). These single-hop links may be formed using shared registers and links, where the size of the registers will be specified according to the size of the forward and backward state metrics.

### C. Routing

The router of each tile in the proposed NoC has five bi-directional ports, which are connected to the IP core of that tile and to the routers of its four neighboring tiles, as shown in Figure 4. These connections form the paths that are used for conveying the extrinsic message LLRs $\bar{b}^{\mathrm{e}}_{1,k}$ between the tiles associated with the upper and the lower decoders of the proposed turbo decoder algorithm. Note that the tiles residing at the edges or the corners of the NoC mesh have fewer than five ports, as shown in Figure 4. In each clock cycle, an input may be provided to some or all of a router's ports, and it may provide an output to one of its ports. More specifically, an input will be provided on the port connected to the associated IP core, if it generates an extrinsic extrinsic message LLR $\bar{b}^{\mathrm{e}}_{1,k}$ during the clock cycle, as described in Section III. Likewise, an input will be provided on a port connected to a neighboring router, if it provides an output on its corresponding port, as will be detailed below. As shown in Figure 4(c), an input provided on a particular port is passed to a corresponding First In First Out (FIFO) buffer, containing four memory elements, where this number was found to offer a desirable tradeoff between NoC performance and hardware requirement [22], [27]. If the FIFO buffer is currently empty, then this input will become immediately available at its output. By contrast, if the FIFO buffer is partially full, then the input is stored in the specific memory element representing the end of the queue and the value stored at the start of the queue is made available at the output of the FIFO buffer. Finally, if the FIFO buffer is totally full, then the input is rejected and the credit out signal of Figure 4(c) is asserted, in order to inform the source of the input. Following this, the crossbar switch of Figure 4(c) is directed by the routing algorithm to be discussed below for selecting the value at the front of the queue in a particular one of the FIFO buffers. The routing algorithm also directs the crossbar switch to clock the selected value into the output buffer of a particular one of the ports, ready to be provided in the next clock to the associated IP core or a neighboring router, as appropriate. However, if the credit in signal of Figure 4(c) is asserted by that IP core or neighboring router, then the output buffer is not updated, so that another attempt can be made to pass the rejected value in the next clock cycle.

The router of Figure 4(c) employs round-robin arbitration, to direct the crossbar switch to select which input FIFO buffer should provide the output of the router in each clock cycle. In successive CCs, round-robin arbitration rotates through each input FIFO buffer in turn, skipping over any that are empty. Furthermore, the router of Figure 4(c) employs the deterministic Dimension-Ordered Routing (DOR) routing algorithm of [27], which is also known as *XY* routing for the 2D mesh topology. As suggested by the terminology, the DOR algorithm

transfers the LLRs from the source tile to the destination tile in a dimension-by-dimension manner, which readily guarantees freedom from deadlock. More specifically, if the destination tile of an LLR has a different X coordinate to that of its current location, then it is routed in the appropriate X direction. Otherwise, it is routed in the appropriate Y direction, until it reaches its destination. Since the proposed scalable turbo decoder has the feature of having self-regulated network traffic, [27], [28] showed that the DOR routing algorithm can outperform the most sophisticated-adaptive routing algorithms, such as negative-first [29], odd-even [30] and DyAD [31], [32].

## V. Results and Evaluation

In this section, we compare the proposed NoC turbo decoder with the application of the conventional Log-BCJR turbo decoder to an NoC. We commence in Section V-A by discussing the differences between the NoC implementation of the proposed turbo decoding algorithm and the conventional Log-BCJR decoding algorithm. Following this, we describe the simulation tools and methodology used for their comparison in Section V-B. Finally, we present the results of our experiments in Section V-C.

### A. Implementation of the Benchmarker

In order to serve as a benchmarker for the proposed NoC turbo decoder, a conventional LTE turbo decoder employing the Log-BCJR algorithm with the PIVI windowing technique [33] was also applied to the proposed NoC of Figure 4. More specifically, this conventional turbo decoder was implemented using the same topology and mapping as the proposed parallel turbo decoding algorithm, where $P$ tiles are used for each of the upper and lower decoder, with each tile allocated to the processing of one of the $P$ windows in a frame. This approach facilitates a direct and fair comparison, since this topology and mapping does not particularly benefit or impede either algorithm. In the Log-BCJR benchmarker, the IP core in each tile is capable of processing one of the $K = N/P$ bits in the corresponding window per clock cycle. However, in contrast to the proposed turbo decoding algorithm, here the ASIP processor in the IP core is designed to process the conventional Log-BCJR algorithm of (9) – (13), where we have $L = 3$ for the upper decoder and $L = 2$ for the lower decoder. Accordingly, each IP core schedules the processing of the bits in the corresponding window across several consecutive CCs, using a conventional Log-BCJR's forward recursion, followed by a backward recursion. More explicitly, during the first $(K - 1)$ CCs of the turbo decoding process, each tile corresponding to the upper decoder performs a forward recursion, in which (9) and (10) are performed for the $j^{\text{th}}$ bit in the window during the $j^{\text{th}}$ clock cycle, where $j \in \{1, 2, 3, \ldots, K - 1\}$. During the next $K$ CCs, each tile corresponding to the upper decoder performs a backward recursion, in which (11) – (13) are performed for the $(2K - j)^{\text{th}}$ bit in the $j^{\text{th}}$ clock cycle, where $j \in \{K, K + 1, K + 2, \ldots, 2K - 1\}$. As and when the extrinsic LLRs $\bar{b}^{\text{u,e}}_{1,k}$ are obtained during the backward recursion, they are routed across the NoC to the tiles of the lower decoder, which may take several CCs. At the corresponding tiles of the lower decoder, these LLRs become the *a priori* message LLRs $\bar{b}^{\text{l,a}}_{1,k}$ according to the interleaver $\pi$, where $\bar{b}^{\text{l,a}}_{1,k} = \bar{b}^{\text{u,e}}_{1,\pi(k)}$. A tile of the lower decoder may begin its forward recursion as soon as the *a priori* message LLR corresponding to the first bit in the window has been received. However, once the forward recursion has started, it may stall while waiting for each successive *a priori* message LLR to be received, owing to the data dependencies of the Log-BCJR algorithm. Once a tile of the lower decoder has completed its forward recursion, it may perform the backward recursion during the next $K$ successive CCs, in order to generate extrinsic LLRs to be routed across the NoC. When these LLRs arrive at the corresponding tiles of the upper decoder of Figure 4(a), they become *a priori* message LLRs. The rest of the iterative decoding process continues in this manner, where stalls may be incurred during the forward recursions of the tiles corresponding to both the upper and lower decoder.

Note that since the tiles process the windows independently of each other, the processing of one window may begin earlier than another. However, the *a priori* message LLRs corresponding to a particular window in either the upper or the lower decoder are typically provided from many different windows in the other decoder, owing to the action of the interleaver. This tends to approximately synchronize the processing of the windows in the benchmarker. Owing to this, the throughput of the benchmarker tends to be limited by the longest routes through the NoC, which cause stalls in the forward recursion processing performed throughout the NoC. Note that this bottleneck affecting the benchmarker's processing efficiency is imposed by the data dependencies of the Log-BCJR algorithm, which are eliminated in the proposed NoC-optimized algorithm of Section III.

The Log-BCJR turbo decoding algorithm.

$$\bar{\gamma}_k(S_{k-1}, S_k) = \sum_{j=1}^{L} \left[ b_j(S_{k-1}, S_k) \cdot \bar{b}^{\mathrm{a}}_{j,k} \right] \tag{9}$$

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

$$\bar{\beta}_{k-1}(S_{k-1}) = \max^{*}_{\{S_k|c(S_{k-1},S_k)=1\}} \left[ \bar{\gamma}_k(S_{k-1}, S_k) + \bar{\beta}_k(S_k) \right] \tag{11}$$

$$\bar{\delta}_k(S_{k-1}, S_k) = \bar{\gamma}_k(S_{k-1}, S_k) + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k) \tag{12}$$

$$\bar{b}^{\mathrm{e}}_{j,k} = \left[ \max^{*}_{\{(S_{k-1},S_k)|b_j(S_{k-1},S_k)=1\}} \left[ \bar{\delta}_k(S_{k-1}, S_k) \right] \right] - \left[ \max^{*}_{\{(S_{k-1},S_k)|b_j(S_{k-1},S_k)=0\}} \left[ \bar{\delta}_k(S_{k-1}, S_k) \right] \right] - \bar{b}^{\mathrm{a}}_{j,k} \tag{13}$$

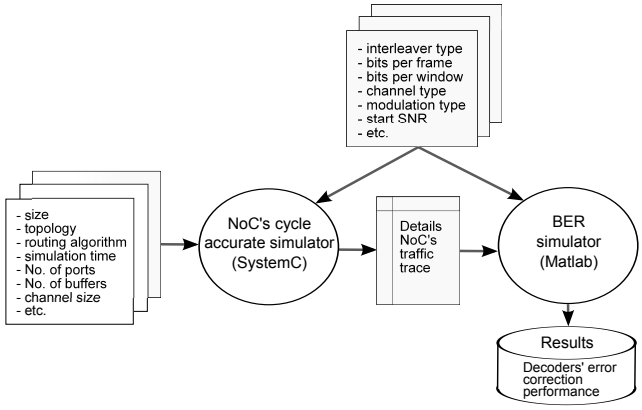



**Fig. 5:** The tool-chain used to evaluate the proposed NoC-optimized turbo decoding algorithm.

### *B. Evaluation Methodology*

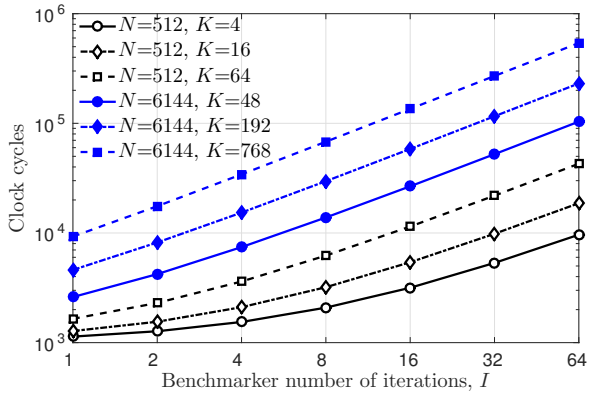
The performance comparison of Section V-C was performed using the tool-chain of Figure 5. Here, Noxim [34] simulator was modified to simulate the traffic that is produced by the proposed NoC. More specifically, Noxim was configured to simulate the schedules used by the proposed NoC-optimized turbo decoding algorithm of Section III and by the benchmarker Log-BCJR algorithm of Section V-A. In this way, Noxim became able to identify, which specific extrinsic LLRs are generated by each tile in each clock cycle. More specifically, by simulating the routing of these LLRs across the NoC, Noxim could also identify which *a priori* LLRs are delivered to each tile in each clock cycle and then apply the turbo decoding schedule for determining which extrinsic LLRs are produced in response and when. Note that while Noxim has awareness of when the LLRs are generated and delivered, it does not have awareness of the specific value of these LLRs, since it does not simulate the computations of the turbo decoding algorithms of Sections III and V-A. Instead, the traffic traces generated by Noxim were provided for Matlab simulations of the algorithms, which were programmed to simulate the computations of the algorithms with consideration of when the *a priori* LLRs become available at each tile. These Matlab simulations were run repeatedly in a Monte Carlo manner, in order to characterize the Bit Error Ratio (BER) performance of the proposed NoC-optimized turbo decoding algorithm, as well as of the benchmarker Log-BCJR algorithm. More specifically, after each iteration of each run of each algorithm, our simulation recorded the number of decoding errors observed and the number of CCs that have elapsed so far. By averaging across the different runs of the algorithms, the BER may be characterized as functions of the number of iterations performed, or equivalently of the number of CCs performed. During the operation of the proposed NoC-optimized turbo decoding algorithm of Section III, the number of 'equivalent iterations' is given by dividing the total number of generated extrinsic LLRs by $2N$, noting that some tiles may have generated more extrinsic LLRs than others. In the case of the benchmarker Log-BCJR algorithm of Section V-A, an iteration is completed whenever a complete set of $2N$ extrinsic LLRs is generated.

As shown in Figure 5, our Noxim simulations are parametrized by the NoC topology, size, and methods of arbitration and routing, which were configured as

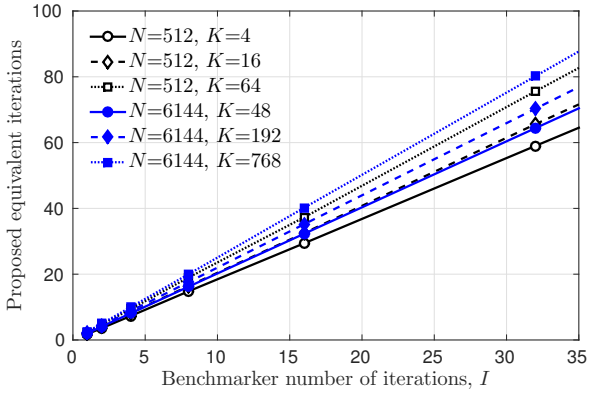
described in Section IV. Furthermore, our Noxim and Matlab simulations are both parametrized by the frame length $N$, the window size $K$ and the interleaver design $\pi$, as shown in Figure 5. Here, the standard LTE interleaver designs were employed for all simulations, while the effects of the parameters $N$ and $K$ are investigated in Section V-C.

### C. Results and Discussions

In this section, we discuss the results of the comparison between the proposed NoC turbo decoder and the application of the conventional Log-BCJR turbo decoder to an NoC. The hardware resource utility achieved by both implementations is discussed in Section V-C1, while their error correction capability is discussed in Section V-C2.




**(a)**




**(b)**

**Fig. 6:** Cycle-accurate Noxim simulation results of routing in the proposed and benchmarker NoC implementations of the LTE turbo decoder. (a) Number of CCs required to complete different numbers of decoding iterations $I$ by the benchmarker employing different frame lengths $N$ and window sizes $K$. (b) Comparison between the number of decoding iterations completed by the benchmarker and the number of equivalent decoding iterations completed by the proposed implementation, in the same number of CCs.

*1) Hardware resource utility:* Figure 6(a) characterizes the number of CCs required by the benchmarker NoC implementation of the LTE turbo decoder for performing different numbers of decoding iterations $I$, when decoding frames having different frame lengths $N$ and window sizes $K$. As may be expected, the number of CCs required for the benchmarker to complete each iteration grows with the window size $K$. However, the relationship is not entirely linear, since a larger NoC size typically results in heavier inter-tile traffic, leading to a greater prevalence of stalls, requiring more CCs to complete each iteration than might otherwise be expected. Note that the NoC size employed is in accordance with the ratio of the frame length $N$ and the window size $K$, as shown in Table I. Owing to this, Table I shows that a small-size NoC facilitates a higher hardware resource utility for the benchmarker than a large-size NoC for a given frame length $N$. Here, the utility is quantified as $\frac{\text{CCs required per } I}{\text{CCs used per } I} \times 100\%$, where the number of CCs used per $I$ is obtained using the experimental results of Figure 6(a), while the number of CCs required per $I$ may be obtained as $4K$, since a single iteration comprises two half-iterations, which includes a $K$-clock-cycle forward recursion and a $K$-clock-cycle backward recursion. Note that although a small-size NoC implementation achieves a better hardware resource utility than a large-size NoC implementation, the required overall CCs per iteration of Figure 6(a) is typically worse, owing to the employment of a large window size $K$.

By contrast, our proposed NoC implementation achieves a constant hardware resource utility of 100% all the time, regardless of the frame length $N$, the window size $K$ and the NoC size. This is due to its elimination of the Log-BCJR data dependencies, allowing it to avoid the stalls that are incurred by the benchmarker. As shown in Figure 6(b), this advantage of the proposed NoC implementation allows it to complete 1.8 to 2.5 times as many equivalent iterations as the benchmarker in the same number of CCs, when implementing the LTE turbo decoder. Note that this performance gain for the proposed NoC implementation is related more to the NoC size rather than to the window size $K$, as in the benchmarker implementation. This is because the data dependencies within a window have been eliminated in the proposed NoC implementation.

*2) Error correction capability:* Figures 7 and 8 characterize the BER performance of the proposed and benchmarker NoC implementations of the LTE turbo decoder, when employing frame lengths of $N = 512$ and $N = 6144$, respectively. In each case, the BER performance is plotted for various window sizes $K$ and for the numbers of CCs that are required for the

**TABLE I:** A comparison between the proposed and benchmarker NoC implementations for different configurations, in terms of hardware resource utility, error correction capability and throughput. Here, the hardware resource utility of the benchmarker is defined as $\frac{\text{CCs required per } I}{\text{CCs used per } I} \times 100\%$, the error correction capability is represented by the $E_b/N_0$ value where a BER of $10^{-4}$ is achieved and the throughput is indicated by the CCs used in order to achieve the BER of $10^{-4}$. In each case, BPSK modulation is employed for communication over an AWGN channel.

| Configurations | | | | Benchmarker resource utility | | | BER and throughput | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Frame length $N$ | Window size $K$ | No. of cores $2P$ | NoC size | CCs required per $I$ | CCs used per $I$ | Resource utility (%) | $E_b/N_0$ (dB) (BER=$10^{-4}$) | CCs for benchmarker | CCs for proposed | Throughput gain (%) |
| 512 | 4 | 256 | 16×16 | 16 | 150 | 10.7 | 2.61 | 3981 | 1271 | 213 |
| | 16 | 64 | 8×8 | 64 | 292 | 21.9 | 2.61 | 3744 | 1825 | 105 |
| | 64 | 16 | 4×4 | 256 | 669 | 38.3 | 2.61 | 5567 | 4261 | 30.7 |
| 2048 | 16 | 256 | 16×16 | 64 | 466 | 13.7 | 1.78 | 3647 | 2442 | 49.3 |
| | 64 | 64 | 8×8 | 256 | 1146 | 22.3 | 1.78 | 5049 | 3528 | 43.1 |
| | 256 | 16 | 4×4 | 1024 | 2694 | 38.0 | 1.78 | 20862 | 18022 | 15.8 |
| 6144 | 48 | 256 | 16×16 | 192 | 1620 | 11.9 | 1.47 | 21829 | 15421 | 41.6 |
| | 192 | 64 | 8×8 | 768 | 3611 | 21.3 | 1.47 | 36824 | 26084 | 41.2 |
| | 768 | 16 | 4×4 | 3072 | 8438 | 36.4 | 1.47 | 85143 | 76713 | 11.0 |

benchmarker to complete $I \in \{2, 4, 8, 16, 32\}$ iterations, when using BPSK modulation for communication over an AWGN channel. After employing a sufficiently high number of CCs in each case, the NoC implementations can be seen to converge towards the BER performance of a conventional Log-BCJR LTE turbo decoder employing only a single window of length $K = N$ and employing $I = 50$ decoding iterations, hence validating the operation of the NoC.

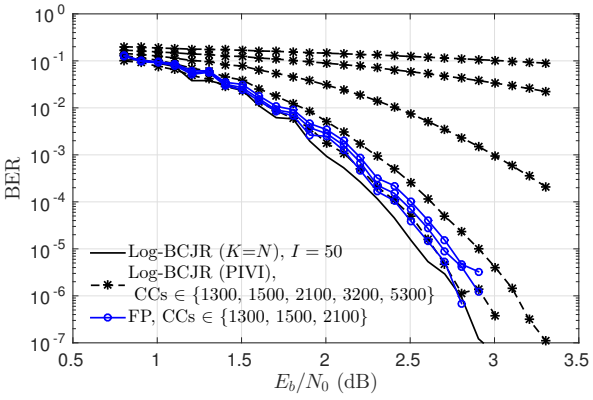
In all cases, Figures 7 and 8 show that within any particular number of CCs, the proposed implementation achieves a superior BER. In the case where $N = 512$ and $K = 4$, the proposed implementation using 1300 CCs achieves a superior BER to the benchmarker using 2100 CCs, as shown in Figure 7(a). This very significant performance gain may be attributed to the avoidance of stalls in the proposed NoC implementation, which allows it to complete around 4 equivalent iterations within the number of CCs required for the benchmarker to complete $I = 2$ iterations, as shown in Figure 6(b). Furthermore, as described in Section III, the proposed implementation achieves a higher hardware resource utility than the benchmarker, with the result that each of its equivalent decoding iteration has a greater benefit to the BER than each iteration of the benchmarker. Note however that the gain demonstrated by the proposed implementation for $N = 512$ and $K = 4$ diminishes as $N$ or $K$ is increased. For example, in the case where $N = 6144$ and $K = 192$, the proposed implementation using 30000 CCs achieves a similar BER to the benchmarker using 58000 CCs, as shown in Figure 8(b).

Note that in conventional PIVI-based Log-BCJR LTE turbo decoders, it is necessary to synchronize the processing within the algorithmic blocks in order to avoid error correction performance degradation. However, in a NoC implementation, the requirement to synchronize the processing of the algorithmic blocks causes stalls, which temporarily prevent decoding progress from being made and which lead to a poor resource utility. By contrast, the novel scheduling of the proposed NoC implementation eliminates stalls, by removing the requirement to synchronize the processing of the algorithmic blocks. This allows the proposed implementation to maintain 100% resource utility and to make more decoding progress within a particular amount of time, leading to the BER gains shown in Figures 7 and 8.

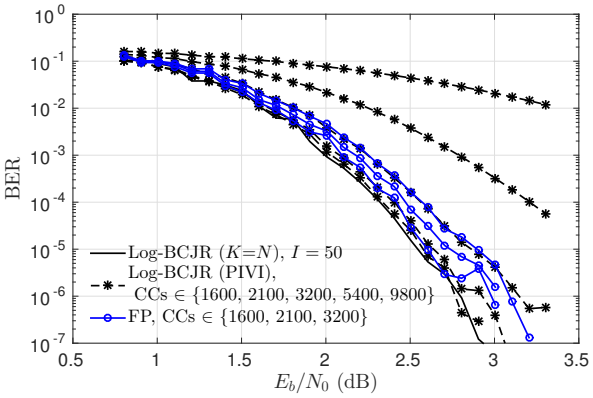
Table I quantifies the average number of CCs required by the proposed and benchmarker NoC implementations of the LTE turbo decoder to achieve a BER of $10^{-4}$ at the $E_b/N_0$ value, where this achieved by the Log-BCJR turbo decoder employing $I = 8$ decoding iterations. For all combinations of the frame length $N$ and the window size $K$ considered, the proposed NoC implementation can be seen to require significantly fewer CCs than the benchmarkers, for the reasons discussed above. The throughput of these NoC implementations is proportional to the reciprocal of the number of CCs that they employ. In accordance with this, Table I quantifies the throughput gain that is offered by the proposed NoC implementation over the benchmarker. This throughput gain is as high as 213% for the case where $N = 512$ and $K = 4$. For most other combinations of $N$ and $K$, the throughput gains offered by the proposed NoC implementation is several tens of percent.

## VI. Conclusions

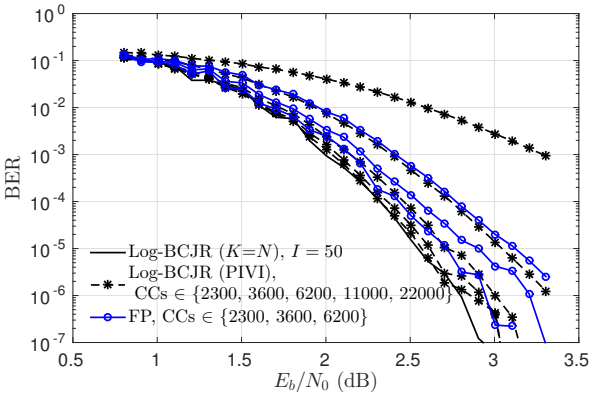
Owing to their programmable arrangement of high-performance IP cores, NoCs hold the promise of facilitating turbo decoders that can flexibly support different frame lengths with high throughputs. In this paper,




**(a)**

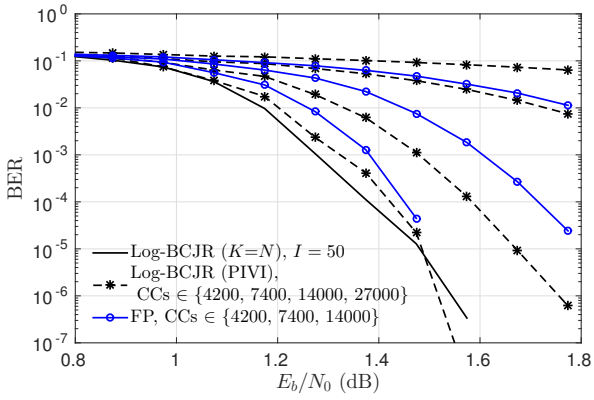



**(b)**

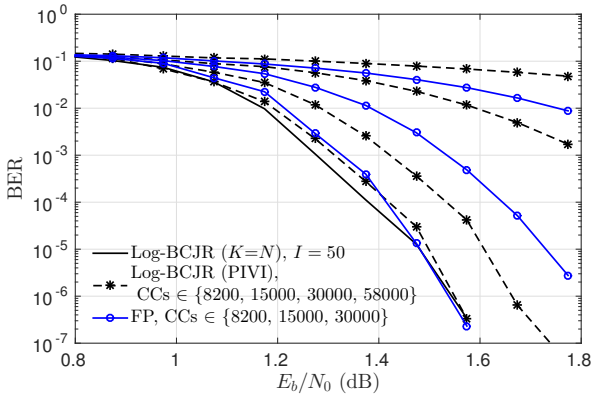



**(c)**

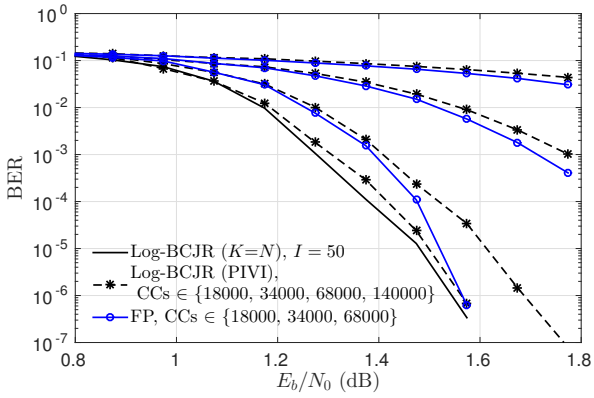
**Fig. 7:** BER performance achieved by the proposed Fully-Parallel (FP) and benchmarker Log-BCJR (PIVI) NoC implementations of the LTE turbo decoder, when employing a frame length of $N = 512$, window sizes of (a) $K = 4$, (b) $K = 16$ and (c) $K = 64$, as well as the numbers of CCs that are required for the benchmarker to complete various numbers of decoding iterations $I \in \{2, 4, 8, 16, 32\}$. The BER performance of a non-windowing Log-BCJR ($K = N$) LTE turbo decoder is also provided for the case of employing the same frame length of $N = 512$ and $I = 50$ iterations. In each case, BPSK modulation over an AWGN channel is assumed.




**(a)**




**(b)**




**(c)**

**Fig. 8:** BER performance achieved by the proposed FP and benchmarker Log-BCJR (PIVI) NoC implementations of the LTE turbo decoder, when employing a frame length of $N = 6144$, window sizes of (a) $K = 48$, (b) $K = 192$ and (c) $K = 768$, as well as the numbers of CCs that are required for the benchmarker to complete various numbers of decoding iterations $I \in \{2, 4, 8, 16\}$. The BER performance of a non-windowing Log-BCJR ($K = N$) LTE turbo decoder is also provided for the case of employing the same frame length of $N = 6144$ and $I = 50$ iterations. In each case, BPSK modulation over an AWGN channel is assumed.

we have proposed a novel NoC implementation of the LTE turbo decoder. This implementation is based upon a novel algorithm that dispenses with the serial data-dependencies of the Log-BCJR algorithm, which other-

wise cause frequent stalls in NoC implementations. By avoiding these stalls, our proposed NoC implementation achieves a significantly higher hardware resource utility than a benchmarker NoC implementation based on the Log-BCJR algorithm. This facilitates significant throughput improvements for a wide variety of frame lengths $N$ and window sizes $K$, including a throughput gain of 213% for the case of decoding an $N = 512$-bit frame using a NoC comprising 16×16 IP cores. Our future work will consider the implementation of the proposed NoC in silicon.

## REFERENCES


[1] 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, no. 99, pp. 1–1, jun 2015. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7131434

[2] 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, 2015. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7360870

[3] A. Abbasfar, D. Divsalar, and Kung Yao, "Accumulate repeat accumulate coded modulation," in *IEEE MILCOM 2004. Mil. Commun. Conf. 2004.*, vol. 1. IEEE, 2004, pp. 169–174. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1493264

[4] ETSI, *LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and Channel Coding*, ETSI Std. v11.1.0, feb 2013.

[5] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70 –78, 2002.

[6] H. Moussa, "On-chip communication network for flexible multiprocessor turbo decoding," in *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, 2008, pp. 1–6.

[7] M. Martina and G. Masera, "Turbo noc: A framework for the design of network-on-chip-based turbo decoder architectures," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 57, no. 10, pp. 2776–2789, 2010.

[8] C. Condo, M. Martina, and G. Masera, "A network-on-chip-based turbo/LDPC decoder architecture," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, 2012, pp. 1525–1530.

[9] C. Condo, M. Martina, and G. Masera, "VLSI implementation of a multi-mode turbo/LDPC decoder architecture," in *IEEE Trans. Circuits Syst. I*, vol. 20, no. 6, pp. 1441–1454, June 2013.

[10] 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, no. 1, pp. 14–22, jan 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6112200

[11] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and Sub-optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding," *Eur. Trans. Telecommun.*, vol. 8, no. 2, pp. 119–125, mar 1997. [Online]. Available: http://doi.wiley.com/10.1002/ett.4460080202

[12] R. G. Maunder, "A Fully-Parallel Turbo Decoding Algorithm," *IEEE Trans. Commun.*, vol. 63, no. 8, pp. 2762–2775, aug 2015. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7137638

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

[14] A. Li, P. Hailes, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "1.5 Gbit/s FPGA implementation of a fully-parallel turbo decoder designed for mission-critical machine-type communication applications," *IEEE Access*, vol. 4, pp. 5452–5473, aug 2016. [Online]. Available: http://eprints.soton.ac.uk/399185/

[15] A. Li, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Implementation of a fully-parallel turbo decoder on a general-purpose graphics processing unit," *IEEE Access*, vol. 4, pp. 5624–5639, june 2016. [Online]. Available: http://eprints.soton.ac.uk/397525/

[16] 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: IEEE, jun 1995, pp. 1009–1013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=524253

[17] M. Wu, Y. Sun, G. Wang, and J. R. Cavallaro, "Implementation of a High Throughput 3GPP Turbo Decoder on GPU," *J. Signal Process. Syst.*, vol. 65, no. 2, pp. 171–183, nov 2011. [Online]. Available: http://link.springer.com/10.1007/s11265-011-0617-7

[18] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proc. Des. Autom. Test Eur. Conf. Exhib. 2000 (Cat. No. PR00537)*. IEEE Comput. Soc, 2000, pp. 250–256. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=840047

[19] S. Khawam, S. Baloch, A. Pai, I. Ahmed, N. Aydin, T. Arslan, and F. Westall, "Efficient implementations of mobile video computations on domain-specific reconfigurable arrays," in *Proc. Des. Autom. Test Eur. Conf. Exhib. DATE-04*, vol. 2. IEEE, 2004, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1269064

[20] K. Srinivasan, K. Chatha, and G. Konjevod, "An automated technique for topology and route generation of application specific on-chip interconnection networks," in *ICCAD-2005. IEEE/ACM Int. Conf. Comput. Des. 2005.*, vol. 2005. IEEE, 2005, pp. 231–237. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1560070

[21] T. Ahonen, D. A. Sigüenza-Tortosa, H. Bin, and J. Nurmi, "Topology optimization for application-specific networks-on-chip," in *Proc. 2004 Int. Work. Syst. Lev. interconnect Predict. - SLIP '04*. New York, New York, USA: ACM Press, 2004, p. 53. [Online]. Available: http://dl.acm.org/citation.cfm?id=966747.966758http://portal.acm.org/citation.cfm?doid=966747.966758

[22] R. Al-Dujaily, T. Mak, F. Xia, A. Yakovlev, and M. Palesi, "Embedded transitive closure network for runtime deadlock detection in networks-on-chip," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 7, pp. 1205–1215, 2012.

[23] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. USA: Morgan Kaufmann Publishers, 2004.

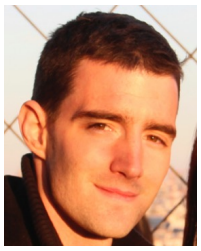
[24] N. Jerger and L. Peh, *On-Chip Networks*, ser. Synthesis lectures in computer architecture. Morgan & Claypool Publishers, 2009. [Online]. Available: http://books.google.co.uk/books?id=Nf9q6goSpssC

[25] Intel, "The 80-core Tera-scale Research Chip," [March.

2, 2012]. [Online]. Available: http://techresearch.intel.com/ProjectDetails.aspx?Id=151
[26] Tilera, "Tilera company products briefs," [Dec. 10, 2011]. [Online]. Available: www.tilera.com/products/processors
[27] G. Ascia, V. Catania, M. Palesi, and D. Patti, "Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip," *Computers, IEEE Transactions on*, vol. 57, no. 6, pp. 809 –820, june 2008.
[28] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, ser. The Morgan Kaufmann Series in Computer Architecture and Design Series. Morgan Kaufmann Publishers, 2004. [Online]. Available: https://books.google.ca/books?id=oOqpcB5191sC
[29] C. Glass and L. Ni, "The turn model for adaptive routing," in *Computer Architecture, 1992. Proceedings., The 19th Annual International Symposium on*, 1992, pp. 278 –287.
[30] G.-M. Chiu, "The odd-even turn model for adaptive routing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 11, no. 7, pp. 729 –738, jul 2000.
[31] J. Hu and R. Marculescu, "DyAD - smart routing for networks-on-chip," in *In ACM/IEEE Design Automation Conference*, 2004, pp. 260–263.
[32] T. Mak, P. Y. K. Cheung, K.-P. Lam, and W. Luk, "Adaptive Routing in Network-on-Chips Using a Dynamic-Programming Network," *IEEE Trans. Ind. Electron.*, vol. 58, no. 8, pp. 3701–3716, aug 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5590290
[33] M. May, C. Neeb, and N. Wehn, "Evaluation of High Throughput Turbo-Decoder Architectures," in *2007 IEEE Int. Symp. Circuits Syst.* IEEE, may 2007, pp. 2770–2773. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4253252
[34] F. Fazzino, M. Palesi, and D. Patti, "Noxim: Network-on-chip simulator," http://noxim.sourceforge.net/.

**Ra'ed Al-Dujaily** received both B.Sc. and M.Sc. degrees in control and computer engineering from the University of Technology of Baghdad/Iraq in 1998 and 2001, respectively. He received the Ph.D. degree in the School of Electrical, Electronic and Computer Engineering, Newcastle University, UK. His research interests include Network-on-Chip, reconfigurable computing, VLSI circuits design and fault tolerance in VLSI.

**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.



**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.

**Terrence Mak** (S'05-M'09) received both B.Eng. and M.Phil. degrees in Systems Engineering from the Chinese University of Hong Kong in 2003 and 2005, respectively, and the Ph.D. degree from Imperial College London in 2009. He joined the School of Electrical, Electronic and Computer Engineering at Newcastle University as a lecturer in 2010. During his Ph.D., he worked as a Research Engineer Intern in the VLSI group at Sun Microsystems Laboratories in Menlo Park, California. He also worked as a Visiting Research Scientist in the Poons Neuroengineering Laboratory at MIT. He was the recipient of both the Croucher Foundation Scholarship and the US Navel Research Excellence in Neuroengineering in 2005. In 2008, he served as the Co-Chair of the UK Asynchronous Forum, and in March 2008 he was the Local Arrangement Chair of the Fourth International Workshop on Applied Reconfigurable Computing. His research interests include FPGA architecture design, Network-on-Chip, reconfigurable computing and VLSI design for biomedical applications.

**Bashir M. Al-Hashimi** (M'99-SM'01-F'09) 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.