A Flexible Iterative Receiver Architecture for Wireless Sensor Networks: A Joint Source and Channel Coding Design Example

Matthew F. Brejza, Robert G. Maunder, Bashir M. Al-Hashimi and Lajos Hanzo
Department of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK
Email: {mfb2g09, rm, bmah, lh}@ecs.soton.ac.uk

Abstract

Due to their computational complexity, iterative decoder components such as source and channel decoders are usually implemented using specialized dedicated hardware. This leads to the scenario where each different iterative decoder component of the receiver requires its own hardware. Owing to their relatively high complexity, many capacity-approaching techniques proposed in the literature have not yet been invoked in Wireless Sensor Network (WSN) applications, despite their potential benefits of facilitating a reduced transmission power or extended communication range. Against this background, we propose an energy-efficient architecture comprised of multiple Computation Units (CUs), which is sufficiently flexible for accommodating different iterative decoder components using the same hardware. In this work, the flexible architecture is applied to Joint Source and Channel Coding (JSCC), comprising the Unary Error Correction (UEC) code, a turbo code, and an iterative demodulator. We conceive a flexible technique for controlling the hardware, which supports a high hardware-exploitation ratio for the CUs, reaching a utilization of 88%, compared to 68% achieved in similar solutions reported in the open literature.

I. Introduction

The authors of [1], [2], [3] have predicted the ubiquity of Wireless Sensor Networks (WSNs) and the Internet of Things (IoT) for the monitoring and control of residential and commercial environments. These systems are typically required to maintain reliable wireless communications over extended periods of time, while relying only on scarce energy resources. The life-span of these energy-constrained wireless communication applications can be significantly extended by employing sophisticated iterative receiver techniques [4], which reduce the required transmission energy. More specifically, it was argued in [5] that error correction codes can be used for redistributing the energy consumption from the energy-constrained sensor nodes to the central data fusion node, which typically has access to more plentiful energy resources. More specifically, the employment of error correction encoding reduces the transmit energy required by the sensor nodes in order to maintain reliable communication. While the use of error correction decoding increases the processing energy consumption at the receiver, this is acceptable since the receiver will often be unconstrained by the available energy, especially at the uplink receiver fed by the mains. This energy redistribution concept was extended in [4] to include multi-hop networks, where jointly considering the transmit energy and processing energy at each hop can lead to a significant overall reduction in energy consumption. As a benefit of this, life-spans of the order of years are potentially facilitated in applications having low duty-cycles, low throughputs and multiple short-range hops. However, in these scenarios, the iterative receiver processing techniques are associated with a significant portion of the overall energy dissipation. Furthermore, the available processing resources of WSN nodes remain limited at the time of writing, hence preventing the application of iterative decoding techniques. As a result, most previous publications on WSNs have considered non-iterative Reed-Solomon (RS) [6] and convolutional codes [7]. Furthermore, the IEEE 802.15.4 standard [8] does not include any channel coding in its PHY layer [7], exemplifying the absence of error correction codes in existing WSNs. Therefore, further significant life-span extensions would be facilitated, if this processing energy dissipation could be reduced.

Previous work has proposed Application-Specific Integrated Circuit (ASIC) designs for a variety of iterative receiver techniques, including synchronization [9], channel estimation [10], equalization [11], bit-to-symbol demapping [12], turbo decoding [13], Low-Density Parity-Check (LDPC) decoding [14] and source decoding [15]. Despite this, there are only a handful of papers that propose iterative decoding architectures specifically designed for wireless sensor networks. In order to facilitate the lowest possible transmission energies, an iterative receiver can benefit from combining several of these techniques. While it would be possible to combine several of these different ASIC decoders into a single System on Chip (SoC), this would not produce the most energy and chip-area efficient design. This is because only one of the various decoders in an iterative receiver is operated at a time, with all the others remaining idle until their turn is reached. Furthermore, the iterative decoding complexity can be minimized by performing more iterations of the less complex techniques and fewer iterations of the complex ones [16]. Idle hardware may be deemed to waste chip area, whilst its static energy consumption will waste energy, particularly for smaller technology scales [17]. This motivated the energy-efficient turbo decoder ASIC of [18], and the LDPC ASIC of [19], both of which efficiently exploit their hardware resources, in order to minimize energy dissipation. However, neither of these architectures supported any of the other iterative receiver techniques that are listed above.

The authors wish to gratefully acknowledge the financial support of the EPSRC, Swindon UK under the auspices of grant EP/J015520/1 and EP/L010550/1, as well as the TSB, Swindon UK under the auspices of grant TS/L009390/1.

Against this background, this paper proposes a programmable ASIC, which can be used for implementing a wide variety of iterative receiver techniques, while maintaining a low decoding energy dissipation. More specifically, this ASIC is designed for the efficient exploitation of its computational resources, regardless of its particular application, in order to minimize both the energy dissipation and area wastage. Hence this ASIC is particularly suitable for WSN applications, where having a low energy consumption and a low chip area are more important than achieving a high throughput. We characterize the proposed architecture using an application example that iteratively activates a Quadrature Phase Shift Keying (QPSK) demodulator [12], a turbo decoder and a Unary Error Correction (UEC) based joint source and channel decoder [20]. In contrast to the family of source codes [21], [22] previously designed for WSN applications, the UEC improves the error correction capability and it is designed to have a low complexity. In this work, a turbo decoder is employed to target applications requiring high throughputs, such as image or video transmission [23]. Furthermore, we employ the UEC code, since it is well suited for encoding the symbols produced by a H.264 or H.265 [24] video encoder, as may be employed in camera-based sensor networks. Furthermore, a high throughput WSN architecture may also be invoked for applications where a receiver may have to decode frames generated by hundreds of low-throughput sensor nodes. The novel contributions of this paper are as follows

- This paper presents the first hardware implementation of a UEC scheme [25], where the proposed architecture is applied to a scheme comprising a UEC decoder, a parallel concatenation of two URC decoders and an iterative demodulator, similar to that of [16].
- The proposed architecture is comprised of a set of general-purpose Computation Units (CUs). We extend the CUs of our previous work [18], so that they can perform the basic operations common to a wider range of iterative decoders, under the instruction of the controller.
- In this way, the same hardware is used for processing the above-mentioned four different decoders in the receiver.
- A novel controller has been developed for handling the scheduling of the four different decoders, and for ensuring that the hardware is kept as busy as possible, in order to achieve a high utility and hardware efficiency. Since the controller can be programmed to implement any particular combination of iterative receiver techniques, this allows the operation of the decoder to be changed, without requiring the ASIC to be redesigned.

We commence by introducing the proposed programmable architecture in Section II. In Section III, we use the proposed architecture for implementing our application example. Section IV characterizes the utility, energy consumption, throughput and chip area of the proposed architecture, while comparing these with previous architectures. Finally, Section IV offers our concluding remarks.

II. Programmable ASIC Architecture

In this section, we provide a brief outline of the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm in Section II-A. This algorithm is fundamental to the operation of iterative receivers, as detailed in [26]. The architecture proposed for implementing iterative receiver processing and the BCJR algorithm will then be described in Section II-B, where we also discuss the specific features of the architecture that allow it to be flexible in its operation.

A. The BCJR Algorithm

Iterative receivers operate on the basis of an iterative exchange of soft information between the various receiver components, which may perform synchronization [9], channel estimation [10], equalization [11], demapping [12], turbo decoding [13], LDPC decoding [14] and source decoding [15] for example. This soft information expresses not only *what* the most likely value of each transmitted bit is, but also *how* likely that bit value is. Therefore, Soft-In Soft-Out (SISO) versions of the various receiver components are required for converting the soft inputs into soft outputs. The BCJR is a flexible SISO algorithm, which can be employed as the basis of the above-mentioned iterative receiver components.

The Maximum Logarithmic Bahl-Cocke-Jelinek-Raviv (Max-Log-BCJR) algorithm [27] is a reduced-complexity variant of the BCJR that is particularly well-suited for implementation. Rather than operating on the basis of bit likelihoods having a high dynamic range, the Max-Log-BCJR algorithm processes the Logarithm of bit Likelihood Ratios (LLRs), which have a low dynamic range and can be readily represented using fixed-point arithmetic. Furthermore, the additions and multiplications of the BCJR algorithm are converted to the reduced-complexity maximum and addition operations in the Max-Log-BCJR algorithm.

The Max-Log-BCJR accepts vectors of *a priori* LLRs as its input and generates vectors of higher-quality extrinsic LLRs as its output. For example, the upper Unity Rate Coding (URC) decoder of Fig. 1 operates on the basis of the Max-Log-BCJR algorithm, having the *a priori* LLR vectors $\tilde{\mathbf{u}}_1^{\mathrm{a}}$ and $\tilde{\mathbf{v}}_1^{\mathrm{a}}$ as its inputs, while generating the extrinsic LLR vectors $\tilde{\mathbf{u}}_1^{\mathrm{e}}$ and $\tilde{\mathbf{v}}_1^{\mathrm{e}}$ as its outputs. The Max-Log-BCJR algorithm operates on the basis of a trellis, which comprises N number of states for each bit. The states corresponding to a particular bit are connected to the states of the next bit using T number of legitimate trellis-transitions. These states and transitions describe the logical constraints and relationships between the consecutive transmitted bits, that are imposed by the corresponding iterative receiver component. The Max-Log-BCJR algorithm converts the *a priori* LLRs into the higher-quality extrinsic LLRs using a sequence of four intermediate steps, as detailed in [26].

Firstly, a set of α values is calculated for each state. Each α value depends on the value of *a priori* LLRs of the corresponding bit, as well as on some α values from the previous bit, depending on the specific transitions in the trellis. Owing to the dependences between the α values of consecutive bits, they must therefore be calculated in a forward-recursive manner along the trellis. This is achieved using low-complexity addition and max operations.

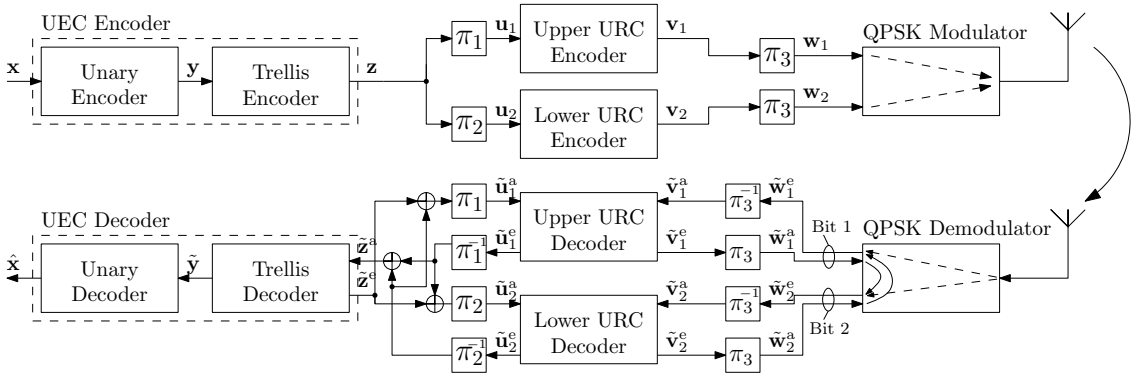


Fig. 1. The UEC-Turbo-QPSK Scheme Schematic.

Secondly, the β values are calculated in a similar manner to the α values. However, instead of depending on the previous bit, each β value depends on some β values from the next bit. Hence the latter values are calculated using a backward recursion along the trellis.

The penultimate step before generating the output is to calculate a δ value for each transition between each bit in the trellis. These are calculated using additions of the α, β and *a priori* LLR values related to the corresponding bit. In the fourth step, extrinsic Logarithmic Likelihood Ratios (LLRs) can then be generated from the set of δ values, using low-complexity subtraction and max operations.

Note that while the reduced-complexity Max-Log-BCJR algorithm relies on the $\max$ operation, the optimal Logarithmic Bahl-Cocke-Jelinek-Raviv (Log-BCJR) algorithm opts instead for using the $\max^*$ operation, where $\max^*(A, B) = \max(A, B) + \ln(1 + e^{-|A-B|})$. By contrast, the Log-BCJR algorithm reduces the transmission energies required for maintaining an adequate SNR at the receiver for near-error-free communication by about 0.5 dB–1 dB, at the cost of a higher complexity. The exact savings depend on the design of the iterative decoding scheme [28]. However, extrinsic scaling [28] may be used in conjunction with the Max-Log-BCJR algorithm, in order to close this performance gap with respect to the Log-BCJR algorithm. To elaborate a little further, extrinsic scaling multiplies the extrinsic LLRs output from the Max-Log-BCJR decoders by a constant in the range of 0.6 – 0.8. This scaling represents our reduced confidence in the extrinsic LLRs, which is imposed by the sub-optimal use of the Max-Log-BCJR algorithm. We adopt this technique in the decoder of Section II-B, which results in a capable yet low-complexity design, having both a low energy consumption as well as a small chip area, as required for WSN applications.

B. The Decoder Top Level

The decoder described in this and the following sections has been designed based on the nature of the Max-Log-BCJR algorithm. As described in Section II-A, the only operations required by the Max-Log-BCJR algorithm are addition, subtraction and $\max$, where the $\max$ operation can be broken down to a subtraction, compare and select operation. Note that the multiplication required by extrinsic scaling can be implemented using shifting and addition operations, as described below. The fundamental Add-Compare-Select (ACS) operations are also the fundamental operations of the Max-Log-BCJR required for other iterative decoding algorithms, such as the min-sum algorithm of LDPC decoders [14].

The grade of parallelism facilitated for the Max-Log-BCJR algorithm is limited by the forward and backward recursions, which impose data dependencies between the α and β values of neighboring trellis stages. However, there are no data dependencies between the N number of α and β values *within* each trellis stage, hence facilitating a parallel processing opportunity as determined by the number of trellis states N. This motivates the conception of a processing architecture comprised of N parallel CUs, where each CU is designed for performing the ACS operations employed by the Max-Log-BCJR algorithm. Note that this architecture could be readily adapted to the scenario, where fewer parallel CUs are used. This would support applications that have a lower throughput requirement in order to benefit from a reduced chip area. More specifically, when operating with fewer CUs, each CU would have to process multiple states for each trellis stage. This leads to a small overhead of including extra registers in each CU to store intermediate values for multiple states, rather than just one state. Furthermore, some additional multiplexers would be required to choose between these registers, depending on which state is being decoded. While the logic area will scale with the number of CUs, the memory requirement will remain the same.

This novel approach of designing a decoder processor architecture facilitates a Single Instruction, Multiple Data (SIMD) like approach for exploiting the parallelism of the flexible decoder. Fig. 2 shows the top level schematic of the proposed iterative receiver processing architecture, which is centered around the group of N parallel CUs. The output generated by each CU is input to a permutation network, if it is to be used as an input to another CU in the next clock cycle. This permutation network allows a CU to pass its calculated values to any other CU, according to the specific requirements imposed by the transitions between states in the trellis. By contrast, if the output generated by a CU is to be used in a later clock cycle, then it is stored in intermediate memory until needed, whereupon the permutation network may be activated to deliver it to the correct CU. For example, the α values generated during the forward recursion are stored

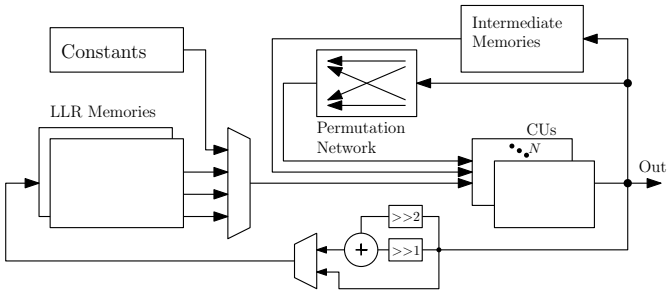


Fig. 2. Top-level schematic of the proposed iterative receiver processing architecture.

in the intermediate memory until used during the backward recursion as part of the δ calculations. During the backward recursion, the extrinsic LLRs are generated and stored in the LLR memories. If extrinsic scaling is used, the LLRs can be multiplied by 0.75 before being stored. This may be achieved at a low hardware cost by using a single adder and two fixed bit shifters, as shown in Fig. 2.

Note that storing all α values generated during a forward recursion of the entire trellis would require a large amount of intermediate memory. One technique for reducing this memory requirement is Previous Iteration Value Initialization (PIVI) [27], which decomposes the trellis into a number of shorter processing windows, which are processed separately. This reduces the intermediate memory requirement to a value that is proportional to the window length, rather than to the trellis length. The unknown α and β values at the ends of each window are initialized using the α and β values that were obtained at the adjacent ends of the neighboring windows during the previous decoding iteration. This requires additional PIVI memory for storing the boundary values between iterations. In the proposed architecture, only the boundary β values for the three separate decoding blocks have to be stored. By contrast, the boundary α values do not need storing, since each window is calculated sequentially in increasing order, so that boundary α values can be passed between windows as they are calculated. However, in addition to the PIVI memory, an α memory is required for storing all of the α values calculated during the forwards recursion of each window. Therefore, in the proposed PIVI windowing architecture, the total number of values which need storing is given by $3NL/\omega$ for PIVI, together with $N\omega$ values for the forward α recursion. Here, N is the number of states in the trellis, ω is the window length and L is the frame length. By comparison, when dispensing with windowing as well as with PIVI and the entire α forwards recursion has to be stored in memory, the total number of state metrics that would have to be stored is NL, which is many times higher. Note that Ilnseher and Kienle [29] describes a 'Re-Computation' method, which stores only every sixth α value during the forwards recursion. This reduces the amount of memory required for storing state metrics, albeit at the cost of requiring extra hardware for recomputing the values, which were not stored when they are later required.

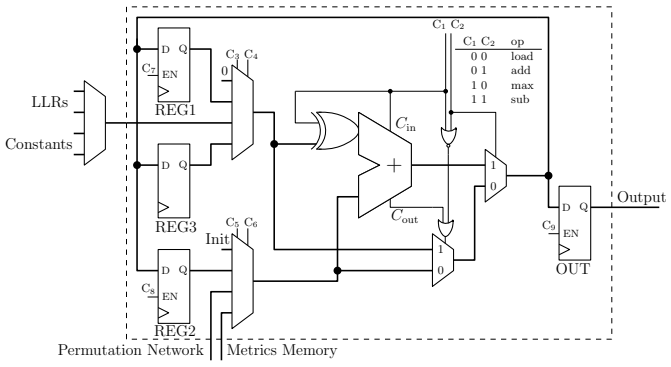


Fig. 3. The block diagram of a CU.

C. The Computation Unit

Fig. 3 shows the internal schematic of the proposed CUs, which performs the ACS operations required by the Max-Log-BCJR algorithm. In each clock cycle, a CU can undertake either an addition, a subtraction or a max operation. Alternatively, it can perform a load instruction, which does not perform any calculation operations, but allows the data to be loaded into the CU's registers for use in subsequent clock cycles. The architecture owes its flexibility partly to the ability for all the parts of the algorithm to be undertaken in a single low-complexity block, which is in contrast to the majority of previous work [28], [30], where each part of the architecture is dedicated to a single function. This characteristic of WSN applications, which requires a low energy consumption and a small chip area, rather than targeting high throughputs, like the suite of existing architectures. The core of the CU is the adder of Fig. 3, which is reused by all the addition, subtraction and max operations. Here a subtraction is carried out by using an XOR gate to invert one of the adder inputs

and then using the carry in for completing the subtraction. The max operation is carried out by performing a subtraction and using the adder's carry out C_{out} for selecting the appropriate input, as shown in Fig. 3.

The CU of Fig. 3 includes three data storage registers provided for storing intermediate results between clock cycles, namely REG1, REG2 and REG3. Our investigations revealed that this number of intermediate data storage registers offers an attractive tradeoff between the chip area, processing throughput and flexibility. Furthermore, the register OUT is used for holding the output, while it is being loaded into the memories or permutation network of Fig. 2. In total, nine control signals, namely C_1-C_9 are used for controlling the registers, multiplexers and the operational mode of the CU, as shown in Fig. 3.

Note that the proposed CU can be readily extended to perform other operations for different algorithms. For example, the proposed CU could implement other max* approximations [31], such as the Look-Up Table (LUT) based max* operation by replacing the max operation in the existing datapath by the circuitry required for the LUT max* operation. If this operation is implemented such that it can be completed within a single clock cycle, then no modification is required for the controller. However, if a multi-cycle LUT max* circuit was employed, such as the LUT max* circuit used in our previous work [18], then the controller would also have to be modified for scheduling these four steps. Furthermore, the datapath of the flexible CUs could be modified to perform the min* (a.k.a. boxplus) operations [32], which would allow these CUs to form the basis of a flexible LDPC decoder.

D. Controller

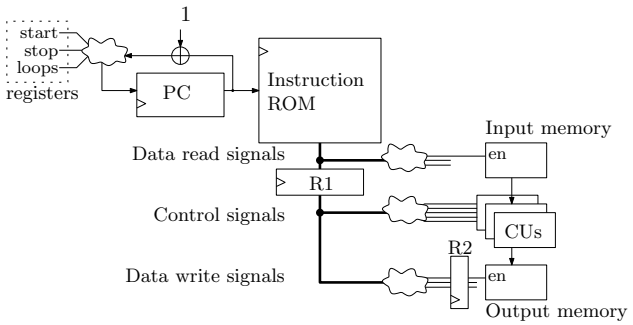


Fig. 4. Controller Block Diagram

The controller's block diagram is shown in Fig. 4, which has to schedule the operations undertaken by the CUs, in order to successfully implement the Max-Log-BCJR algorithm. In contrast to the approach of most other decoder architectures, the controller of the proposed architecture is designed to be programmable, so that it can flexibly carry out the iterative operations of the decoders of different types, as will be demonstrated in Section III. In order to maximize the processing throughput and to minimize the energy consumption of the proposed architecture, it is necessary for the controller to maximize the utility of the CUs. Since the Max-Log-BCJR algorithm is comprised of many identical calculations that are performed using different data for different states of the trellis, a SIMD approach to the control of the CUs is motivated. This has the advantage of minimizing the complexity overhead of the controller, since it is not required to control the CUs individually. The proposed controller can either issue a single instruction to all CUs, or it can decompose the CUs into two groups of any size and simultaneously issue two different instructions to these.

As shown in Fig. 4, the proposed controller is based around an instruction memory, which can be loaded with sequences of instructions that control the operation of each stage of the Max-Log-BCJR algorithm, for each of the concatenated decoders. A program counter (PC) is used for addressing the successive instructions of the current sequence within the instruction memory, resembling the instruction fetching behavior of a simple processor. As shown in Fig. 4, looping around and jumping between the sequences of instructions stored in the instruction memory is achieved by loading the PC with values loaded from the looping registers. These store the address of the start and end of each instruction sequence, as well as the number of times that each sequence should be looped over, before jumping to the next sequence.

The control signals of the CUs in the decoder are derived directly from the bits in the current instruction, with some added pipeline delays, labeled as R1 and R2 in Fig. 4. These are required for matching the delays imposed by reading and writing to both the LLR and to the metrics memories of Fig. 2. This results in a low-complexity, yet flexible controller, which can perform one of the operations described in Section II-C in every clock cycle, leading to a high utilization of the hardware.

III. Application to a Joint Source Coding, Channel Coding and Modulation Scheme

In this section, we demonstrate the employment of the proposed architecture for implementing the receiver of a joint source coding, channel coding and modulation scheme. This application includes a variety of different receiver components, which employ different versions of the Max-Log-BCJR algorithm, exemplifying the flexibility of the proposed programmable ASIC architecture. This example employs the UEC code for joint source and channel coding, which is particularly suited to zeta-distributed integer values in the range of one to infinity. which are produced by sources obeying Ziph's law [33]. This models the symbols generated by a wide variety of physical processes, such a H.265 video encoder

TABLE I
THE UNARY CODEWORDS.

x_i	Unary(x_i)
1	0
2	10
3	110
4	1110
5	11110
6	111110
7	1111110
⋮	⋮

[25], for example. However, the flexibility of the proposed architecture allows for other iterative decoder blocks to be used, depending on the specific requirements of the WSN system. In particular, the flexibility of the architecture allows it to implement the classic systematic turbo code, as used in mobile telephony. We commence in Section III-A by detailing the transmitter design, in order to aid the description of the receiver design in Section III-B. The mapping of this receiver design onto the proposed programmable ASIC architecture is detailed in Section III-C.

A. Transmitter Design

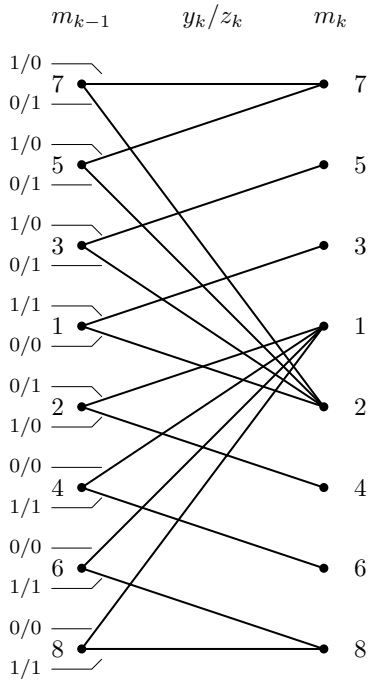


Fig. 5. Trellis for the UEC trellis encoder.

The transmitter of the joint source coding, channel coding and modulation scheme is shown in Fig. 1. This scheme is designed for conveying a stream $\mathbf{x}$ of zeta-distributed source symbols. The UEC encoder comprises a unary encoder and an $N = 8$-state UEC trellis encoder. This is employed for encoding the stream of source symbols $\mathbf{x}$, transforming it into the stream of bits $\mathbf{z}$ [16]. More specifically, for each input symbol x_i, the unary encoder outputs the bit vector Unary(x_i) according to Table I. Note that the length of each unary codeword is equal to the corresponding symbol value x_i. These unary encoded bit-vectors are then concatenated and partitioned into bit-vectors $\mathbf{y}$, all having the length L. Following this, the UEC trellis encoder transforms the bit-vector $\mathbf{y}$ into the bit-vector $\mathbf{z}$, using the 8-state trellis shown in Fig. 5. At the start of each frame of bits, the trellis encoder is reset to its default initial state of $m_0 = 1$. For each successive bit y_k in $\mathbf{y}$, the trellis traverses to a new state m_k, and outputs a bit z_k. Each bit z_k output from the trellis encoder is concatenated to form the bit-vector $\mathbf{z}$. The bit-vector $\mathbf{z}$ is then interleaved by the interleavers π_1 and π_2 to obtain the bit-vectors $\mathbf{u}_1$ and $\mathbf{u}_2$. Turbo channel encoding is performed by a pair of parallel concatenated $N = 8$-state URC encoders [16] and the resultant bit-vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ are interleaved by a pair of interleavers having the same design π_3. The bits in the resultant vectors $\mathbf{w}_1$ and $\mathbf{w}_2$ are paired up and QPSK modulated onto the channel using a natural mapping. This mapping motivates iterative demodulation in the receiver and improves the attainable error correction performance [12].

B. Receiver Design

The receiver designed for this scheme is shown in Fig. 1, which closely mirrors the transmitter design, except that it processes frames of LLRs, rather than frames of bits. Furthermore, these frames of LLRs are iteratively exchanged

bidirectionally between the demodulator, the URC decoders and the UEC decoder, via the interleavers π_1 – π_3 and the deinterleavers π_1^{-1} – π_3^{-1}.

Upon reception of a transmission, the QPSK demodulator outputs the pair of extrinsic LLR frames $\tilde{\mathbf{w}}_1^{\mathrm{e}}$ and $\tilde{\mathbf{w}}_2^{\mathrm{e}}$, which pertain to the corresponding bit frames $\mathbf{w}_1$ and $\mathbf{w}_2$ in the transmitter. These extrinsic LLRs are obtained by combining the information received from the channel with any information available within the *a priori* LLR frames $\tilde{\mathbf{w}}_1^{\mathrm{a}}$ and $\tilde{\mathbf{w}}_2^{\mathrm{a}}$ provided by the URC decoders, as shown in Fig. 1. More specifically, the demodulator algorithm [12] generates the first extrinsic LLR frame $\tilde{\mathbf{w}}_1^{\mathrm{e}}$ by combining the information received from the channel with the second *a priori* LLR frame $\tilde{\mathbf{w}}_2^{\mathrm{a}}$. Likewise, the demodulator generates the second extrinsic LLR frame $\tilde{\mathbf{w}}_2^{\mathrm{e}}$ by combining the information received from the channel with the first *a priori* LLR frame $\tilde{\mathbf{w}}_1^{\mathrm{a}}$. This relationship is shown in the circuit of Fig. 1 and it is exploited by the hardware mapping of Section III-C for reducing the number of memory accesses required.

The UEC trellis decoder and the URC decoders apply the Max-Log-BCJR algorithm to their $N = 8$-state trellises, as previously described in Section II-A. However, the URC decoders operate on the basis of a different trellis design from that of the UEC trellis decoder. More specifically, trellis transitions connect each state in the URC trellis to two other states, whereas different states are connected to different number of other states in the UEC trellis, as shown in Fig. 5. A further difference is that the knowledge of the symbols' zeta probability distribution can be incorporated into the α and β calculation of the UEC trellis decoder, in order to enhance its error correction capability [25].

As shown in Fig. 1, the URC decoders convert the *a priori* LLR frames $\tilde{\mathbf{u}}_1^{\mathrm{a}}$, $\tilde{\mathbf{v}}_1^{\mathrm{a}}$, $\tilde{\mathbf{u}}_2^{\mathrm{a}}$ and $\tilde{\mathbf{v}}_2^{\mathrm{a}}$ into the extrinsic LLR frames $\tilde{\mathbf{u}}_1^{\mathrm{e}}$, $\tilde{\mathbf{v}}_1^{\mathrm{e}}$, $\tilde{\mathbf{u}}_2^{\mathrm{e}}$ and $\tilde{\mathbf{v}}_2^{\mathrm{e}}$. Similarly, the UEC trellis decoder converts the *a priori* LLR frame $\tilde{\mathbf{z}}^{\mathrm{a}}$ into the extrinsic LLR frame $\tilde{\mathbf{z}}^{\mathrm{e}}$. The *a priori* LLR frame input to each of the URC decoders and to the UEC trellis decoder are obtained as the sum of the extrinsic LLR frames provided by the other two decoders, as shown in Fig. 1. At the start of the iterative decoding process, all *a priori* LLR frames are initialized with zero-valued LLRs and iterative decoding continues until all of these frames have been updated a certain number of times. At this point, the UEC trellis decoder inputs $\tilde{\mathbf{y}}$ to the unary decoder, which carries out the final hard decision and outputs the final received symbols $\hat{\mathbf{x}}$, according to Table I. Since the bit representation of each symbol varies in length, there may be trailing bits in the frame owing to the partitioning of the bit-vector $\mathbf{y}$ into fixed length parts. Here, these trailing bits represent an incomplete symbol, which may be stored in the unary decoder, until the next frame is received. The total number of symbols in each frame may be sent as side information to assist the decoder, which also ensures that the decoded symbols in the receiver stay synchronized to the transmitter.

C. Mapping of the Receiver Design to the Proposed Architecture

In this section, we describe how the decoder architecture described in Section II-B can be employed for implementing the receiver described in Section III-B. The proposed implementation supports interleaver lengths of up to 6144 bits, in conjunction with UEC and URC window lengths of $\omega = 64$ trellis stages. These long frame lengths are well suited to the WSN applications such as video transmission, which have relatively high throughputs, and high numbers of bits per video frame. A 9-bit fixed point 2's complement format is used for representing the internal values, in accordance with the findings of [34]. Furthermore, the demodulator symbol probabilities are represented by unsigned 4-bit fixed point values, which we found to be the minimum number of bits imposing a negligible performance degradation. Finally, we scale the extrinsic LLRs of $\tilde{\mathbf{z}}^{\mathrm{e}}$, $\tilde{\mathbf{u}}_1^{\mathrm{e}}$ and $\tilde{\mathbf{u}}_2^{\mathrm{e}}$ by 0.75 in order to mitigate the performance loss imposed by using the Max-Log-BCJR over the Log-BCJR, as described in Section II-A. The following sections further detail how the individual parts of the scheme seen in Fig. 1 are mapped to the proposed architecture.

1) URC Decoder: In order to reduce the memory requirement of the Max-Log-BCJR algorithm, the URC trellis [27, Fig. 1] is decomposed into groups of n trellis stages referred to as windows, which are processed separately and in order, as described in Section II-B. The decoding process invoked for each window of the URC decoder proceeds using PIVI, as follows. Firstly, the decoder undertakes a forwards recursion for calculating the α values for the window, storing them in the intermediate memories of Fig. 2. This forward recursion is initialized using the α values calculated at the end of the previous window. As shown in Fig. 6, each CU is used for calculating the α value for a different one of the $N = 8$ URC states in each trellis stage, using three clock cycles. This is followed by the window's backwards recursion, which is initialized using PIVI, where the β values are gleaned from the beginning of the next window required from the previous iteration of the decoder. Then, on a trellis-stage by trellis-stage basis, the β values are calculated, followed by the δ values and the output extrinsic LLR using a total of 10 clock cycles, as shown in Fig. 6. Initially for the first bit of each window the β value calculations require three clock cycles. However, for the subsequent bits a partial result gleaned from the δ calculation is saved in registers, which is used by the β calculation, hence resulting in this calculation requiring only a single clock cycle, as shown in Fig 6. Note that the extrinsic LLR calculation requires two 8-input max operations and a subtraction to be undertaken. This is achieved using eight 2-input max operations in a first clock cycle, four 2-input max operations in a second clock cycle, two 2-input max operations in a third clock cycle, and a subtraction in a fourth clock cycle, as shown in Fig. 6 However, this process does not require all $N = 8$ CUs in all four clock cycles, hence resulting in a slight under-utilization of the CUs for this part of the decoding process.

Owing to the flexible nature of the decoder, these idle cycles of the CU can be used for performing the iterative demodulator operations. Fig. 1 shows that each of the demodulator's extrinsic LLRs $\tilde{\mathbf{w}}_1^{\mathrm{e}}$ and $\tilde{\mathbf{w}}_2^{\mathrm{e}}$ depends only on one *a priori* LLR from $\tilde{\mathbf{w}}_2^{\mathrm{a}}$ or $\tilde{\mathbf{w}}_1^{\mathrm{a}}$ respectively. As a benefit, the demodulator can be operated immediately after the *a priori* LLR has been obtained by interleaving the corresponding extrinsic LLR of $\tilde{\mathbf{v}}_1^{\mathrm{e}}$ or $\tilde{\mathbf{v}}_2^{\mathrm{e}}$. Fig. 6 shows how each of these extrinsic LLRs is stored in the registers of a CU, until the next period in which there are idle CUs. The demodulator

can be operated using four additions in a first clock cycle, plus two max operations in a second clock cycle and a final addition in a third clock cycle, as shown in Fig. 6. The extrinsic LLRs generated by the URC decoder and demodulator are then both saved into the LLR memories.

Finally, after the decoding of a window has been completed, the last set of β values are saved in the intermediate memories, in order to initialize the decoding of the adjacent window during the next iteration of that decoder, in accordance with PIVI.

2) UEC Trellis Decoder: The UEC trellis decoder operates in a similar fashion to the URC decoder, employing windowing and PIVI. Fig. 6 shows that more clock cycles are needed for the UEC trellis decoder, owing to its different trellis structure. More specifically, while each state of the URC trellis has only two merging transitions, an $N = 8$-state UEC trellis has two central states with four merging transitions, which results in a max operation that requires two clock cycles, using two CUs, as shown in Fig. 6.

As described in Section III-B, the other key difference associated with the UEC trellis decoder is that the *a priori* knowledge of the source symbols' probability distribution can be incorporated into the α and β calculation, in order to enhance the attainable performance. These values are constants, which can be loaded into the CUs by the input multiplexer, as shown in Figures 2 and 3. This extra addition can be seen in Fig. 6, and results in two extra clock cycles being required per bit.

The $N = 8$-state UEC trellis [16, Fig. 3] was chosen to match the number of URC trellis states, which was found to provide the best exploitation of the proposed architecture's hardware resources. Note that the number of states that are used in a UEC trellis can be selected independently of the number used in the encoder and can be adjusted at run-time to strike a trade-off between error correction capability and complexity [25]. Motivated by this, the proposed architecture may also be configured to operate in a mode, where only $N = 4$ states are used in the trellis decoder. This doubles the throughput of the UEC decoder, since two bits are scheduled to be decoded at the same time.

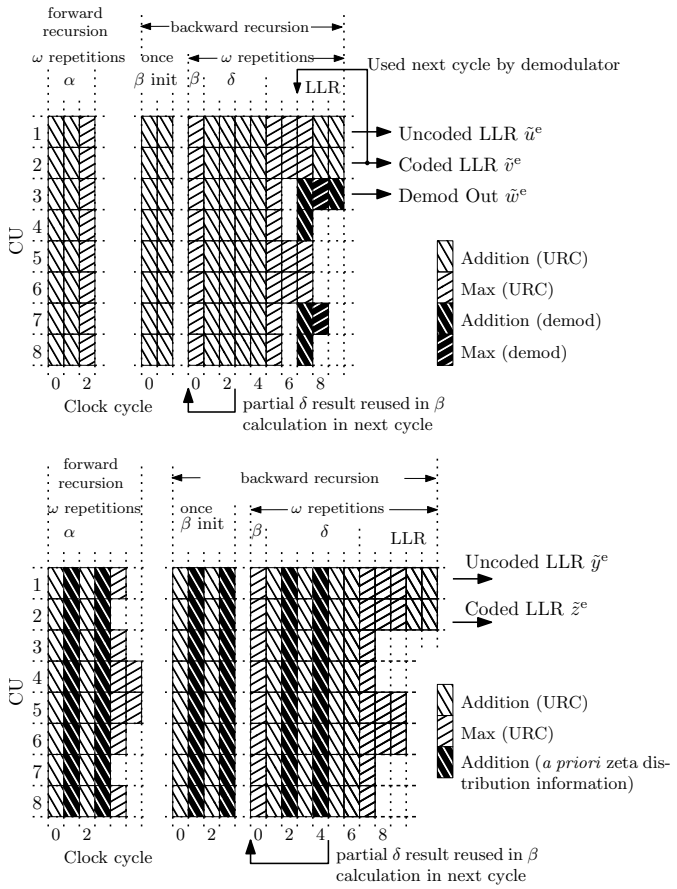


Fig. 6. Timing diagram for the CUs when performing URC and UEC decoding. Here, the operations for one trellis stage are shown for both the forwards and backwards recursion. Before performing the first backwards recursion for each window, the operations 'β init' are activated. Since the δ calculation and the β calculation in the next trellis stage share an identical operation, the result of this operation is saved in the CU's registers during the δ calculation for use in the subsequent β calculation, as indicated by the arrows on the two timing diagrams.

IV. Results

In this section we shall characterize the performance of the proposed architecture and the implementation of the iterative decoder shown in Fig. 1. Since this is the first implementation of a UEC-based iterative decoder, there is no previous work that provides a direct comparison. However, in order to confirm that our design operates within reasonable boundaries, we will compare it – wherever possible – to the architecture of [18]. Fig. 7 shows the Symbol Error Ratio (SER) performance of

TABLE II
THE KEY CHARACTERISTICS OF THE PROPOSED ARCHITECTURE COMPARED TO OTHER WSN CHANNEL DECODERS.

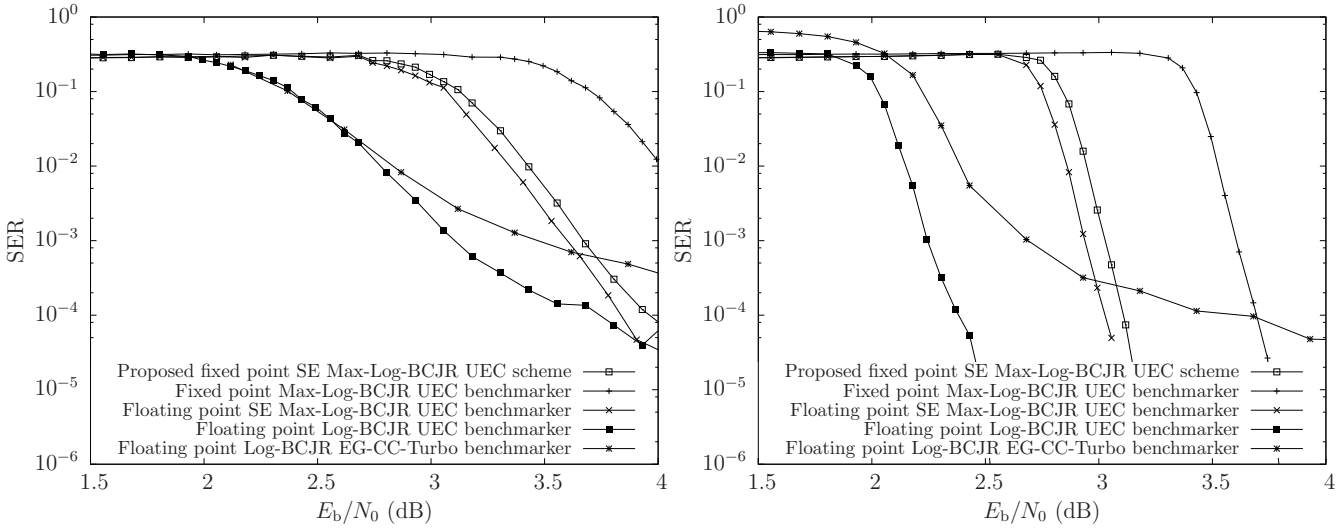
	This work	Li 2013 [18]	Biroli 2012 [19]
Decoder type	Max-Log-BCJR	Log-BCJR	Serial LDPC
Frame length L	6144	6144	576
Technology scale (nm)	90	90	90
Voltage (V)	1.0	1.0	1.0
Frequency (MHz)	333	333	20
Iterations I	8	5	10
Memory requirement (kbit)	289	188	
Area (mm^2)	0.46	0.35	0.13
Throughput (Mbit/s)	0.93	1.03	0.25
Area efficiency (Area/Throughput)	0.49	0.34	0.52
Energy efficiency (nJ/bit/iteration)	0.80	0.81	0.27

the implemented iterative decoder after $I = 8$ decoding iterations, when receiving $L = 624$-bit or $L = 6144$-bit frames of zeta-distributed symbols having the parameter $p_1 = 0.797$, which have been transmitted over an Additive White Gaussian Noise (AWGN) channel. Our implementation is compared against three benchmarkers, in order to explicitly quantify the impact of our design decisions. The first of these represents the lower-bound performance of the scheme, which is obtained under the idealized conditions of using floating point numbers and the exact Log-BCJR. The second benchmarker employs the floating point scaled extrinsic Max-Log-BCJR, in order to show the impact of our fixed point numbers, as well as to demonstrate how the scaled extrinsic LLRs close the performance gap between the Max-Log-BCJR and the exact Log-BCJR. In order to quantify the advantage of using Joint Source and Channel Coding (JSCC), the third benchmarker is a Separate Source and Channel Coding (SSCC) scheme. This benchmarker is referred to as the EG-CC-Turbo scheme, since it replaces the UEC code of our proposed scheme with the Elias Gamma (EG) source code and a separate Convolutional Code (CC), while retaining the turbo code comprising the two URCs. More specifically, the EG source code encodes input symbols, yielding a bit-vector which does not have equiprobable bit values, which would normally lead to capacity loss [16]. In order to mitigate this capacity loss, the 8-state CC encoder is invoked for encoding the EG encoder's output into a bit-vector which has equiprobable bit values. A parallel concatentation of URC encoders is then used as the channel code, before QPSK modulation. In the EG-CC-Turbo receiver, iterations occur between the two URC decoders and the CC decoder. The benchmarker has a similar complexity to the proposed scheme, since the three decoders of the proposed scheme and the three decoders of the benchmarker all employ 8 states. Note that the EG-CC-Turbo benchmarker employs floating point numbers and the exact Log-BCJR. We employ the parameter value of $p_1 = 0.797$ for the zeta distribution that is used for generating the random source symbols, since this results in the same effective throughput of $\eta = 0.762$ bits per symbol for both the UEC schemes and the EG-CC-Turbo benchmarker of [16]. Compared to the corresponding JSCC UEC benchmarker, the SSCC EG-CC-Turbo benchmarker requires an SNR of about 0.7 dB higher, in order to reach the SER turbo cliff at frame lengths of both 624 and 6144 bits. This benchmarker also suffers from a pronounced error floor, which potentially causes a high SER at higher SNRs.

Table II shows the key performance characteristics of the proposed architecture, compared to similar decoders for WSN applications, namely the turbo decoder of [18] and the LDPC decoder of [19]. The proposed implementation has a total memory requirement of 289 kbits, which is significantly higher than the 188 kbit of [18]. However, the memory requirement of the demodulator's symbol probabilities constitutes the majority of the memory in the proposed implementation. Motivated by the fact that [18] does not consider the implementation of the demodulator, ignoring the demodulator's memory requirement in our implementation leads to a total memory requirement of 190 kbits, which is comparable to [18]. This paper has demonstrated support for frame lengths of 6144 bits, however, further memory reduction can be achieved if the ASIC is not required to support long frame lengths. Note that further throughput improvements could be achieved by employing early stopping [28] to halt the iterative decoding process, as soon as a Cyclic Redundancy Check (CRC) is satisfied. In this case, the results provided in Table II would represent the worst-case performance of the proposed scheme.

The timing diagrams of Fig. 6 can be used for obtaining the number of clock cycles required for processing each interleaved bit. It can be seen that 13 clock cycles per bit are required for the URC decoder, while 18 clock cycles per bit are necessitated for the UEC trellis decoder [16, Fig. 3]. Furthermore, the exploitation of the CUs is 88% for the URC decoder and demodulator, while it is 81% for the UEC trellis decoder. These exploitation ratios constitute significant improvements over the 68% recorded in [18].

The proposed ASIC design was synthesized by Synopsis Design Compiler using the ST 90nm 1.00V design kit. The design can achieve a clock frequency of 333 MHz after design compilation and uses approximately 0.46 mm^2 of die area,



(a) $L = 624$ (b) $L = 6144$

Fig. 7. SER plot for the proposed scheme after 8 iterations, for frames comprising $L \in \{624, 6144\}$ bits. The implementation is compared to benchmarkers which implement the exact Log-BCJR to show the implementation loss, as well as a SSCC benchmarker to show the improvement offered by the UEC.

most of which is for the memories. This is comparable to the solution of [18], which has an area of 0.35 mm^2 at the same clock frequency and technology scale. With the aid of $I = 8$ decoding iterations, our design achieves a decoded throughput of 0.93 Mbps, while the previous solution of [18] achieved 1.03 Mbps at the same clock frequency, albeit without the overhead of UEC decoding and demodulation. Meanwhile the LDPC decoder of [19] achieved a decoded throughput of 0.25 Mbps, but has a smaller chip area of 0.13 mm^2, at the same technology scale. The worst-case post-synthesis energy consumption of our design is 0.80 nJ/bit/iteration, which is higher than that of the comparable architectures due to the inclusion of the UEC trellis decoder. While the architecture of [18] achieved a comparable energy efficiency of 0.81 nJ/bit/iteration, the LDPC design of [19] achieved an energy efficiency of 0.27 nJ/bit/iteration. As shown in Table II, the three decoders achieve a similar area efficiency, despite the proposed design having the overhead of including UEC decoding and demodulation.

V. Conclusions

In this paper we have proposed a flexible and programmable architecture suitable for WSNs that can be used to implement iterative receivers employing a wide variety of different components. We have demonstrated the application of the proposed architecture for iteration between a UEC decoder, a turbo decoder and an iterative demodulator. The flexibility of the architecture allows for the iterative demodulator to use otherwise idle hardware, ultimately achieving a hardware exploitation of up to 88%, while restricting the area of the design to 0.46 mm^2 with a 0.93 Mbps throughput. This compares favorably to a significantly less-capable benchmarker, which achieved a hardware exploitation ratio of 68%, while having an area of 0.35mm^2 and a throughput of 1.03 Mbps.

References

[1] M. Zorzi and A. Gluhak, "From today's intranet of things to a future internet of things: a wireless-and mobility-related view," *Wireless Communications, IEEE*, 2010.

[2] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: a survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, oct 2010.

[3] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, sep 2013.

[4] M. F. Brejza, L. Li, R. G. 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 Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 8–28, 2016. [Online]. Available: http://eprints.soton.ac.uk/378161/

[5] R. G. Maunder, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, and L. Hanzo, "Iterative Decoding for Redistributing Energy Consumption in Wireless Sensor Networks," in *2008 Proceedings of 17th International Conference on Computer Communications and Networks*. IEEE, aug 2008, pp. 1–6.

[6] Y. Qassim and M. E. Magana, "Error-tolerant non-binary error correction code for low power wireless sensor networks," in *The International Conference on Information Networking 2014 (ICOIN2014)*. IEEE, feb 2014, pp. 23–27.

[7] A. Abedi, "Power-efficient-coded architecture for distributed wireless sensing," *IET Wireless Sensor Systems*, vol. 1, no. 3, pp. 129–136, sep 2011. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/iet-wss.2010.0077

[8] "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)," 2006.

[9] N. Noels, C. Herzet, A. Dejonghe, V. Lottici, H. Steendam, M. Moeneclaey, M. Luise, and L. Vandendorpe, "Turbo synchronization: an EM Algorithm Interpretation," *IEEE International Conference on Communications, 2003. ICC '03.*, vol. 4, pp. 2933–2937, 2003.

[10] S. Haene, A. Burg, N. Felber, and W. Fichtner, "OFDM channel estimation algorithm and ASIC implementation," in *2008 4th European Conference on Circuits and Systems for Communications*. IEEE, jul 2008, pp. 270–275.

[11] C. B. Catherine Douillard, Michel Jézéquel, "Iterative Correction of Intersymbol Interference: Turbo-Equalization," in *European transactions on telecommunications*, 1995, pp. 507–511.

[12] M. Valenti and S. Cheng, "Iterative demodulation and decoding of turbo-coded M-ary noncoherent orthogonal modulation," *Selected Areas in Communications, IEEE Journal on*, 2005.

[13] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error correcting coding and decoding: turbo codes," in *Proceedings of the IEEE International Conference on Communications*, vol. 2, Geneva, Switzerland, 1993, pp. 1064–1070.

[14] R. Gallager, "Low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, jan 1962.

[15] J. Hagenauer and N. Gortz, "The Turbo Principle in Joint Source-Channel Coding," in *Proceedings 2003 IEEE Information Theory Workshop*. IEEE, 2003, pp. 275–278.

[16] W. Zhang, Y. Jia, X. Meng, M. Brejza, R. G. Maunder, and L. Hanzo, "Adaptive iterative decoding for expediting the convergence of unary error correction codes," *IEEE Transactions on Vehicular Technology*, pp. 1–1, may 2014.

[17] T. Austin, D. Blaauw, T. Mudge, K. Flautner, M. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: moore's law meets static power," *Computer*, vol. 36, no. 12, pp. 68–75, dec 2003.

[18] L. Li, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A low-complexity turbo decoder architecture for energy-efficient wireless sensor networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 1, pp. 14–22, jan 2013. [Online]. Available: http://eprints.soton.ac.uk/271820/

[19] A. D. G. Biroli, M. Martina, and G. Masera, "An LDPC Decoder Architecture for Wireless Sensor Network Applications," *Sensors*, vol. 12, no. 12, pp. 1529–1543, feb 2012.

[20] M. F. Brejza, W. Zhang, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Adaptive iterative detection for expediting the convergence of a serially concatenated unary error correction decoder, turbo decoder and an iterative demodulator," in *Communications (ICC), 2015 IEEE International Conference on*, jun 2015, pp. 2603 – 2608. [Online]. Available: http://eprints.soton.ac.uk/375712/

[21] J. Kim, K. R. Vijayanagar, and W. Liu, "Low-complexity distributed multiple description coding for wireless video sensor networks," *IET Wireless Sensor Systems*, vol. 3, no. 3, pp. 205–215, sep 2013. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/iet-wss.2012.0115

[22] I. Balasingham, H. Nguyen, and T. Ramstad, "Wireless sensor communication system based on direct-sum source coder," *IET Wireless Sensor Systems*, vol. 1, no. 2, pp. 96–104, jun 2011. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/iet-wss.2010.0094

[23] W.-T. Chen, P.-Y. Chen, W.-S. Lee, and C.-F. Huang, "Design and implementation of a real time video surveillance system with wireless sensor networks," in *VTC Spring 2008 - IEEE Vehicular Technology Conference*. IEEE, may 2008, pp. 218–222.

[24] ITU-T, "Series H: audiovisual and multimedia systems, infrastructure of audiovisual services coding of moving video, high efficiency video coding," 2015. [Online]. Available: www.itu.int/rec/T-REC-H.265-201504-I

[25] R. G. Maunder, W. Zhang, T. Wang, and L. Hanzo, "A unary error correction code for the near-capacity joint source and channel coding of symbol values from an infinite set," *IEEE Transactions on Communications*, vol. 61, pp. 1977–1987, 2013.

[26] L. L. Hanzo, T. H. Liew, B. L. Yeap, R. Y. S. Tee, and S. X. Ng, *Turbo coding, turbo equalisation and space-time Coding: EXIT-chart-aided near-capacity designs for wireless channels*. John Wiley & Sons, 2011.

[27] D. Yoge and N. Chandrachoodan, "GPU Implementation of a Programmable Turbo Decoder for Software Defined Radio Applications," *VLSI Design (VLSID)*, pp. 149–154, 2012.

[28] C. Benkeser, A. Burg, T. Cupaiuolo, and Q. Huang, "Design and Optimization of an HSDPA Turbo Decoder ASIC," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 98–106, 2009.

[29] T. Ilnseher and F. Kienle, "A 2.15 GBit/s turbo code decoder for LTE advanced base station applications," in *Turbo Codes and Iterative Information Processing (ISTC), 2012 7th International Symposium on*, 2012, pp. 21–25.

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

[31] S. Papaharalabos, P. Mathiopoulos, G. Masera, and M. Martina, "On optimal and near-optimal turbo decoding using generalized max operator," *IEEE Communications Letters*, vol. 13, no. 7, pp. 522–524, jul 2009.

[32] S. Papaharalabos, P. Sweeney, B. Evans, P. Mathiopoulos, G. Albertazzi, A. Vanelli-Coralli, and G. Corazza, "Modified sum-product algorithms for decoding low-density parity-check codes," *IET Communications*, vol. 1, no. 3, p. 294, 2007. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/iet-com_20060173

[33] N. L. Johnson, A. W. Kemp, and S. Kotz, *Univariate discrete distributions*. John Wiley & Sons, 2005.

[34] L. Li, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Design of Fixed-Point Processing Based Turbo Codes Using Extrinsic Information Transfer Charts," in *Proceeding of IEEE Vehicular Technology Conference*, Ottawa, Canada, 2010, pp. 1–5.