A Bit-Node Centric Architecture for Low-Density Parity-Check Decoders

Conference Paper · December 2007

DOI: 10.1109/GLOCOM.2007.57 · Source: IEEE Xplore

3 authors, including:

Ruwan Ratnayake
London Metropolitan University
7 PUBLICATIONS **21** CITATIONS

SEE PROFILE

Gu-Yeon Wei
Harvard University
234 PUBLICATIONS **7,697** CITATIONS

SEE PROFILE

A Bit-Node Centric Architecture for Low-Density Parity-Check Decoders

Ruwan N.S. Ratnayake
School of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
Email: ratnayak@fas.harvard.edu

Erich F. Haratsch
LSI Corporation
Allentown, PA 18109

Gu-Yeon Wei
School of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
Email: guyeon@eecs.harvard.edu

***Abstract*—A bit-node centric decoder architecture for low-density parity-check codes is proposed. This architecture performs the optimum sum-product algorithm. A bit node processing unit computes the bit-to-check node messages sequentially, while the computation of the check-to-bit node messages is broken up into several steps. A stand-alone decoder architecture, and a decoder architecture for a concatenated detector-decoder system are presented. The proposed stand-alone decoder architecture requires significantly less memory compared to other known serial architectures. The hardware requirements are reduced even further for the concatenated detector-decoder system.**

I. Introduction

The phenomenal performance of turbo codes based on iterative decoding and belief propagation has revitalized the interest in low-density parity-check (LDPC) codes [1], [2]. Though Gallager first discovered LDPC codes several decades ago [3], until recently these codes were not considered for practical applications due to their immense computational complexity. The near Shannon limit performance of LDPC codes, as shown by Chang *et. al.* [4] makes them a strong competitor to turbo codes. Although various architectures have been proposed in the past years for the implementation of LDPC codes, there is still a need for more efficient architectures for area-constrained applications, such as magnetic recording.

Inherently, LDPC codes have less structure compared to other codes such as trellis or turbo codes. An LDPC code is fully defined by a parity-check matrix, which determines the internal connections within the decoder. The sparse, random-like nature of the matrix makes the implementation of these connections challenging. In contrast, turbo codes have a more regular structure. Though randomizing the messages is a necessary feature in turbo codes, it is fulfilled by an external interleaver. Because the computation of messages and message passing are performed separately in turbo codes, the decoder primarily consists of computational circuitry. On the other hand, randomness is inherent to the LDPC code and the message passing is defined by the code itself. Therefore, the message computations and message passing are inseparable, which complicates the implementation of LDPC codes.

One approach for the implementation of the LDPC decoder is a fully parallel architecture, where the exact copy of the bi-partite graph of the code is laid out in hardware. Each bit and check node are constructed as combinational logic and the wires represent the edges of the graph, which connect the bit and check nodes. A fully parallel architectures achieves high throughput since an entire code word is processed simultaneously. However this approach has several disadvantages. One drawback is the lack of decoding flexibility. Since the decoder is hardwired it can generally decode only a single code. Another significant disadvantage is the enormous complexity associated with the wire connections. The number of messages passed along the edges is large in a typical LDPC code, and for an adequate resolution each message needs to be represented with a sufficient number of bits. Therefore, the number of wires required to connect the nodes exceeds tens of thousands even if the code word length is moderate and in the range of about one thousand bits. A significant portion of the chip needs to be allocated simply for wiring, which reduces area utilization. The paper by Blanksby *et al.* [5] describes a fully parallel LDPC decoder designed and implemented in 0.16μm technology. The chip measures about 53.5mm^2 and half of the area is allocated for wire routing. Moreover this excessive wiring complexity is typically associated with non-uniform delays and cross talk issues which further diminish the merits of the fully parallel approach.

The wiring and computational complexity associated with the fully parallel architecture can be avoided with a serial architecture. In this approach, the wiring overhead is reduced by saving the messages in dedicated memory. Furthermore, bit and check node messages are computed sequentially by reading the relevant messages from the memory, processing them and writing the results back into the memory. Additionally, since the message passing in not hardwired, the decoder can accommodate more than a single code. Recently, several serial architectures have appeared in the literature [6], [7], [8]. In particular, the papers by Wu *et al.* [9] and Hocevar [10] present serial architectures, which process either the check node or bit node computations on the fly. The architecture proposed in this paper performs the check node computations on the fly. However, our architecture implements the optimum sum-product algorithm while requiring significantly less memory and in certain instances less latency compared to the other serial architectures.

The remaining part of this paper is organized as follows. In Section II we explain the considered LDPC decoding

1930-529X/07/$25.00 © 2007 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2007 proceedings.

algorithm. Afterwards in Section III we give an in-depth description of our proposed architecture. The memory requirements, throughput characteristics and a comparison of the proposed architecture with other architectures are given in Subsections III-A and III-B, respectively. Section IV describes the proposed architecture for a concatenated detector-decoder system. Final concluding remarks are given in Section V.

II. Sum-Product Algorithm for LDPC Decoding

An LDPC code is defined by a binary parity-check matrix consisting of n columns and m rows. The matrix is typically large in size, but sparse. The parity-check matrix can be mapped to a bi-partite graph composed of nodes and edges. A given column in the matrix corresponds to a unique bit node and a given row in the matrix corresponds to a unique check node. A bit node i is connected to a check node j by an edge if the element at the $(j,i)^{th}$ position in the parity-check matrix is non-zero. The set of check nodes connected to a given bit node i is denoted by $\mathcal{B}_i$. These are the non-zero positions in the i^{th} column in the matrix. In the same manner the set of bit nodes connected to a given check node j is denoted by $\mathcal{C}_j$, which corresponds to the non-zero elements in the j^{th} row. The cardinality of $\mathcal{C}_j$ and $\mathcal{B}_i$ are called row and column degrees and denoted by d_r and d_c, respectively. For simplicity we consider only regular codes, where the row and column degrees are constant, independent of the row or column index. A message passed from bit node i to check node j is denoted by $Q_{i,j}$, where $j \in \mathcal{B}_i$. Similarly, a message from check node j to bit node i is denoted $R_{j,i}$, where $i \in \mathcal{C}_j$.

LDPC codes can be decoded iteratively using the sum-product algorithm. The algorithm consists of four main steps. In the following, iterations internal to the decoder are indexed as k. The *a priori* and *a posteriori* log-likelihood ratios (LLR) of the bit i are denoted by λ_i and Λ_i, respectively:

1) Initialization
   All the messages from check node j to bit node i are initialized to zero, i.e. $R^0_{j,i} = 0$.
2) Bit node processing
   For each $i \in \{1,...,n\}, j \in \mathcal{B}_i$ compute $Q^k_{i,j}$ based on the messages from check nodes generated in the previous iteration and the *a priori* LLR,

$$Q^k_{i,j} = \sum_{l \in \mathcal{B}_i, l \neq j} R^{k-1}_{l,i} + \lambda_i \tag{1}$$

3) Check node processing
   For each $j \in \{1,...,m\}, i \in \mathcal{C}_j$ compute $R^k_{j,i}$,

$$R^k_{j,i} = s^k_{j,i} \times \phi^{-1}\left(\sum_{l \in \mathcal{C}_j, l \neq i} \phi\left(|Q^k_{l,j}|\right)\right) \tag{2}$$

   where the sign bit $s^k_{j,i}$ is given by

$$s^k_{j,i} = \prod_{l \in \mathcal{C}_j, l \neq i} \text{sign}(Q^k_{l,j}) \tag{3}$$

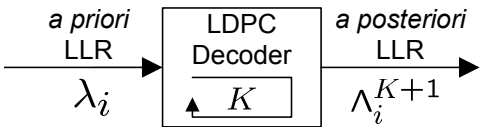

Fig. 1. LDPC decoder in stand-alone mode.

and

$$\phi(x) = -\log\left(\tanh\frac{x}{2}\right) = \log\left(\frac{e^x+1}{e^x-1}\right) = \phi(x)^{-1}$$

4) *A posteriori* LLR computation
   After K iterations, the *a posteriori* LLR for each bit i is computed as

$$\Lambda^{K+1}_i = \sum_{l \in \mathcal{B}_i} R^K_{l,i} + \lambda_i. \tag{4}$$

The equations (1) - (4) are well known and cited here for reference. Fig. 1 shows the decoder in stand-alone mode where iterations are performed internal to the decoder.

III. Bit-Node Centric Architecture for a Stand-Alone LDPC Decoder

This paper proposes to break up the check node computations in (2) and (3) into several steps so that check-to-bit node message can be computed on the fly:

$$\rho^k_{i,j} \triangleq \phi(|Q^k_{i,j}|) \qquad j \in \{1,...,m\}, i \in \mathcal{C}_j \tag{5}$$

$$\sigma^k_{i,j} \triangleq \text{sign}(Q^k_{i,j}) \qquad j \in \{1,...,m\}, i \in \mathcal{C}_j \tag{6}$$

$$P^k_j \triangleq \sum_{l \in \mathcal{C}_j} \rho^k_{l,j} \qquad j \in \{1,...,m\} \tag{7}$$

$$S^k_j \triangleq \prod_{l \in \mathcal{C}_j} \sigma^k_{l,j} \qquad j \in \{1,...,m\} \tag{8}$$

where $\rho^k_{i,j}$ is the transformed magnitude of the bit-to-check node message $Q_{i,j}$ and $|\,|$ is the notation for magnitude. Based on these quantities, the sign and magnitude of the message from check node j to bit node i is given by:

$$|R^k_{j,i}| = \phi(P^k_j - \rho^k_{i,j}) \tag{9}$$

$$\text{sign}(R^k_{j,i}) = S^k_j \times \sigma^k_{i,j} \tag{10}$$

The check node computations are performed in sign-magnitude format, while the bit node computations can be performed in 2's complement format.

Fig. 2(a) illustrates the block diagram of the proposed architecture. The block diagram is for a simplified example where $d_c = 3$, $\mathcal{B}_i = \{j1, j2, j3\}$. At each time cycle the bit node processing unit (BNPU) performs computations pertaining to a single bit node and generates d_c number of bit-to-check node messages. The messages from the d_c check nodes to the same bit node are computed on the fly using equations equations (5) - (10). Hence this approach is called bit-node centric architecture. Given that the decoder processes d_c bit-to-check node messages and d_c check-to-bit node messages, it

1930-529X/07/$25.00 © 2007 IEEE

takes n cycles to process the entire code of length n. Within the first time cycle, the first bit node $(i = 1)$ is computed. Then at $(n(k-1)+i)^{th}$ time cycle, the decoder is computing the i^{th} bit node at the k^{th} iteration and produces messages $Q^k_{i,j}, \forall j \in \mathcal{B}_i$.

The bit-to-check node messages generated by the BNPU are converted from 2's-complement to sign-magnitude format. The magnitude values of the messages are sent to combinational logic units that perform the function ϕ and the outputs from these units are the transformed magnitudes, $\rho^k_{i,j}, j \in \mathcal{B}_i$ as defined in (5). The sign values are $\sigma^k_{i,j}, j \in \mathcal{B}_i$ as defined in (6). Without loss of generality it is assumed that each message is represented by q bits, where both the magnitude and transformed magnitude consist of $q-1$ bits and the sign comprises 1 bit. The sign and transformed magnitude pairs, denoted by $[\sigma^k_{i,j}, \rho^k_{i,j}]$, are fed to the type-1 update processing units (UPU1) as shown in 2(a). Each pair $[\sigma^k_{i,j}, \rho^k_{i,j}]$ corresponds to a message from the i^{th} bit node to the j^{th} check node, and is a q-bit sign-magnitude quantity.

The type-1 update processing unit comprises simple circuitry, which includes an adder and an XOR gate and is connected to suitable memory as shown in Fig. 2(b). A single memory element stores a q-bit quantity. At each cycle the bank of UPU1s can access any d_c number of memory elements from m memory elements. The UPU1s update the relevant memory elements such that at the end of an iteration (i.e. nk^{th} time cycle) the values $[S^k_j, P^k_j], \forall j \in \{1...m\}$ as defined in (7) and (8) are stored in the m memory elements. In other words, each memory element keeps a running product of the σ's and a running sum of the ρ's. If the intermediate value in the memory element $j \in \{1...m\}$ for the k^{th} iteration is given by $[S^k_j(i), P^k_j(i)]$ for $i \in \{1...n\}$, then all the running products and sums saved in the memory at the $(n(k-1)+i)^{th}$ time instance are given by:

$$S^k_j(i) = \begin{cases} S^k_j(i-1) \times \sigma^k_{i,j} & \text{if } j \in \mathcal{B}_i \\ S^k_j(i-1) & \text{else} \end{cases} \quad (11)$$

$$P^k_j(i) = \begin{cases} P^k_j(i-1) + \rho^k_{i,j} & \text{if } j \in \mathcal{B}_i \\ P^k_j(i-1) & \text{else.} \end{cases} \quad (12)$$

At the end of an iteration the memory contains:

$$S^k_j = S^k_j(n) \qquad j \in \{1...m\} \quad (13)$$

$$P^k_j = P^k_j(n) \qquad j \in \{1...m\}. \quad (14)$$

Fig. 2(b) shows the circuitry for UPU1. At each cycle each UPU1 accesses a q-bit memory element, where the sign value $S^k_j(i-1)$ is fed to the XOR gate and the magnitude value $P^k_j(i-1)$ is fed to the adder. The quantities $\sigma^k_{i,j}$ and $\rho^k_{i,j}$ generated within the current cycle are also fed to the XOR gate and adder, respectively. The output of the XOR gate, $S^k_j(i)$ and the output of the adder, $P^k_j(i)$ are paired as $[S^k_j(i), P^k_j(i)]$ and written back to the q-bit memory element.

The transformed magnitude and sign bit pairs $[\sigma^k_{i,j}, \rho^k_{i,j}]$ are also fed into the first-in first-out (FIFO) buffer as shown in Fig. 2(a). Since d_c number of such pairs are computed at each time cycle, the buffer requires $d_c \times q$ bits per row. At each cycle

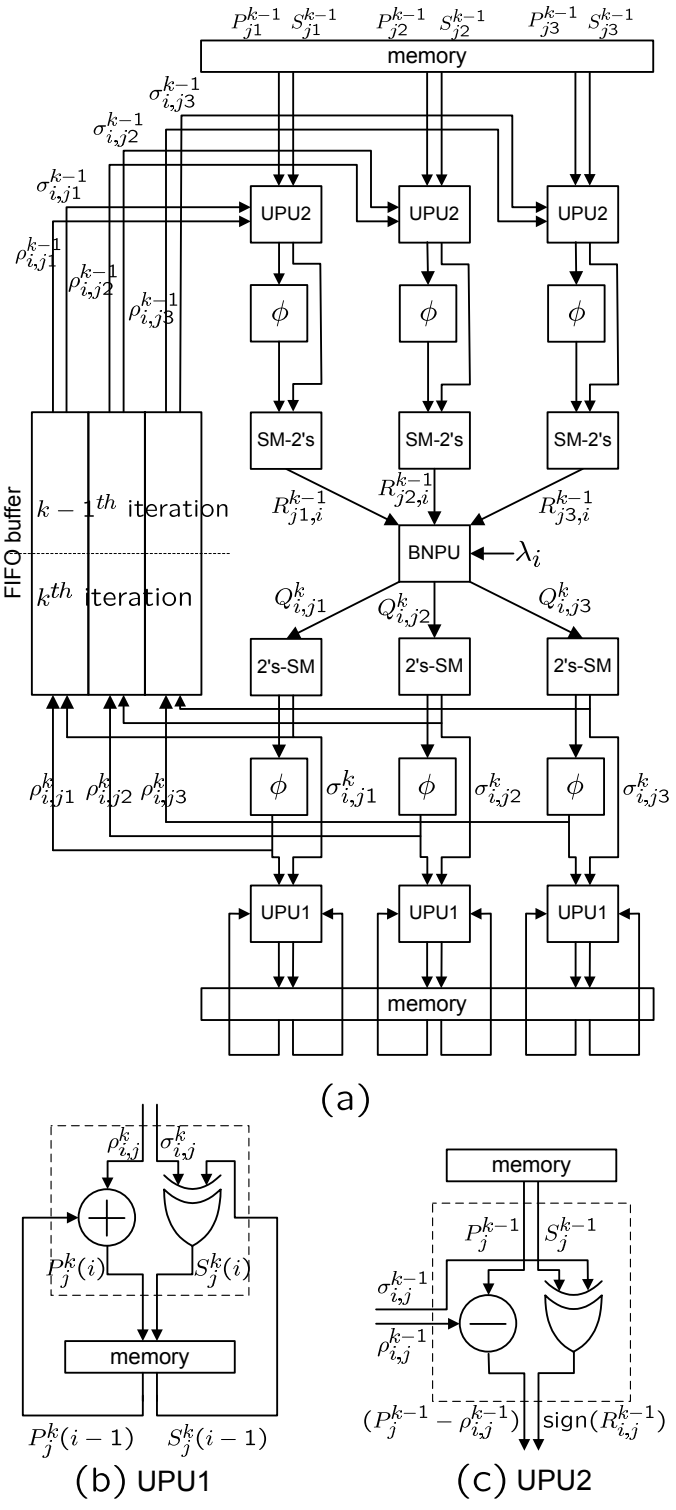

Fig. 2. Bit-node centric decoder architecture for $d_c = 3, \mathcal{B}_i = \{j1, j2, j3\}$. (a) block diagram, (b) type-1 update processing unit (UPU1), (c) type-2 update processing unit (UPU2).

the messages fill in one row of the buffer. Since the decoder saves information pertaining to the entire code, the buffer has n such rows.

At time cycle $n(k-1)+i$, the set of pairs $[\sigma^k_{i,j}, \rho^k_{i,j}], j \in \mathcal{B}_i$

1930-529X/07/$25.00 © 2007 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2007 proceedings.

is fed to the bottom of the buffer and the set of pairs $[\sigma_{i,j}^{k-1}, \rho_{i,j}^{k-1}], j \in \mathcal{B}_i$ from the previous iteration is read from the top of the buffer. After completing n such cycles, i.e. at nk^{th} time cycle, the buffer contains $\sigma_{i,j}^{k}$ and $\rho_{i,j}^{k}, \forall i \in \{1...n\}, j \in \mathcal{B}_i$. Row i in the buffer contains the information pertaining to bit node i. The bottom row of the buffer contains $[\sigma_{1,j}^{k}, \rho_{1,j}^{k}], j \in \mathcal{B}_1$, and the top row contains $[\sigma_{n,j}^{k}, \rho_{n,j}^{k}], j \in \mathcal{B}_n$. At this time cycle the updated memory contains the corresponding final signs S_j^k and magnitudes $P_j^k, \forall j \in \{1...m\}$.

The check-to-bit node messages $R_{j,i}^{k-1}$ are computed based on the values computed in the previous iteration, namely $[S_j^{k-1}, P_j^{k-1}]$, $j \in \{1...m\}$ saved in the memory and $[\sigma_{i,j}^{k-1}, \rho_{i,j}^{k-1}], j \in \mathcal{B}_i$ saved in the FIFO buffer. These quantities are fed to the type-2 update processing unit (UPU2). UPU2 consists of an adder and an XOR gate as shown in Fig. 2(c). At each iteration, S_j^{k-1} and $\sigma_{i,j}^{k-1}, \forall j \in \mathcal{B}_i$ are read from the memory and the FIFO buffer, respectively and the product $(S_j^{k-1} \times \sigma_{i,j}^{k-1})$ is computed using XOR gates. These are the sign values of the corresponding messages from the d_c check nodes to the i^{th} bit node for the previous iteration, namely $\text{sign}(R_{j,i}^{k-1}), j \in \mathcal{B}_i$ according to (10).

The magnitude of the check-to-bit node messages is evaluated in a similar manner. At the k^{th} iteration, the values P_j^{k-1} and $\rho_{i,j}^{k-1}, j \in \mathcal{B}_i$ pertaining to bit node i are read from the memory and FIFO buffer, respectively. The relevant intermediate results $(P_j^{k-1} - \rho_{i,j}^{k-1})$ are computed for each $j \in \mathcal{B}_i$ and passed to combinational units that perform the function ϕ. The outputs of these functional units are the magnitudes of the messages from the corresponding check nodes to the i^{th} bit node for the previous iteration, namely $|R_{j,i}^{k-1}|, \forall j \in \mathcal{B}_i$ according to (9). The sign and magnitude values are passed through combinational logic that converts from the sign-magnitude to 2's-complement format. The results $R_{j,i}^{k-1}, \forall j \in \mathcal{B}_i$ are the inputs to the BNPU for the i^{th} bit node from the previous iteration.

A. Memory Requirements and Throughput

The FIFO buffer requires nd_cq memory bits. The memory capacity needed to store all the pairs $[S_j, P_j], \forall j \in \{1...m\}$ is $2mq$ memory bits. The multiplication factor 2 is due to the need to save the current computations pertaining to iteration k while simultaneously reading the values from the previous iteration $k-1$. Thus the decoder requires a total of $(2m+nd_c)q$ memory bits.

At each time cycle the proposed architecture computes d_c number of check-to-bit node messages and d_c number of bit-to-check node messages. Thus it takes n cycles to process a code of length n using a single BNPU and d_c number of UPU1s and UPU2s.

B. Advantages over Other Architectures

The proposed architecture performs the sum-product algorithm as defined by the equations (1) to (4). Therefore, this architecture achieves better bit error rate performance than other architectures that implement the min-sum decoding algorithm such as [9], where only the minimum values are considered for the message computation. Moreover the architecture described in [9] is for a concatenated detector-decoder system only.

TABLE I
ARCHITECTURE COMPARISON

Architecture	Memory	Clock cycles	
		Low code rate	High code rate †
Yeo	$2nd_cq$	$2n$	$2n$
CNC	$(2n + md_r)q$	m	$\approx n$
BNC	$(2m + nd_c)q$	n	n

Yeo: Serial architecture proposed in [6]

CNC: Check-node centric architecture

BNC: Bit-node centric architecture

† : Assumes maximum of d_c parallel memory accesses per memory bank per cycle

Hocevar describes an architecture in [10] that also computes both bit-to-check and check-to-bit messages within one cycle. However, Hocevar's architecture is check-node centric since it uses a check node processing unit and breaks up the bit node computations. Mansour *et al.* [8] also describe an architecture that is very similar to the one in [10]. The check-node centric architecture described in [10] requires $(2n + md_r)q$ memory bits to save the required messages and intermediate values. On the other hand, our bit-node centric architecture requires $(2m+nd_c)q$ memory bits. The difference in memory requirements is $2(n-m)q$ bits, since $md_r == nd_c$. Thus the bit-node centric architecture always requires less memory than the check-node centric architecture since $m < n$. The memory savings are particulary significant for high code rates, which are desirable in magnetic storage systems. Assuming $d_c = 3$ and a code rate equal to $\frac{8}{9}$ (which in turn dictates $d_r = 27$ for regular codes) the check-node centric architecture needs $45mq$ memory bits whereas the proposed architecture needs only $29mq$ bits. Thus in this typical scenario, the check-node centric architecture requires close to 50% more memory compared to the bit-node centric approach.

Furthermore, a check node processing unit needs to sum d_r number of messages whereas a bit node processing unit only needs to sum up d_c number of messages. For high code rates d_r is considerably larger than d_c. Thus, for such codes the check-node centric architecture would need a significantly larger adder tree. To alleviate a resulting speed bottleneck the adder tree in the check-node centric architecture may need to be pipelined.

For minimum decoding latency, both the bit-node centric and check-node centric architecture need to access the memory multiple times within a cycle. In the bit-node centric architecture, the decoder needs to access d_c memory elements from a bank of m elements within a cycle. Although this memory access can be serialized at the cost of increased number of

1930-529X/07/$25.00 © 2007 IEEE

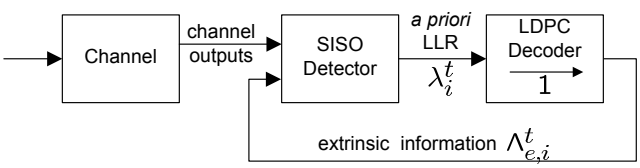

Fig. 3. LDPC decoder concatenated with a soft-input soft-output detector.

cycles per iteration, for high code rates d_c is small (e.g. 3 or 4) making parallel access a feasible option. On the other hand, if the check-node centric architecture processes one check node within a cycle it needs to randomly access any permutation of d_r memory elements from n elements. Typically, in magnetic recording applications both d_r and n are prohibitively large (long codes and high code rates) making such parallel access impractical to implement. Therefore, for such applications the memory access may need to be serialized for check-node centric decoders. This significantly increases the number of cycles needed per iteration compared to the m required cycles if parallel access is feasible.

Table I summarizes the memory requirements and number of required clock cycles for different architectures. For comparison, we have also considered the serial architecture proposed by Yeo *et. al.* [6]. This architectures has one bit node processing unit and d_c number of pipelined check node processing units. Each check node processing unit needs d_r number of cycles to complete the computations for a single check node. Thus it takes n cycles to process all check nodes and another n cycles to process all bit nodes. The scheme switches read and write operations between two memory banks. At each cycle it reads messages from one memory bank and writes the results to the other.

For low code rates with small d_r, the check-node centric architecture requires less cycles per iteration compared to the bit-node centric architecture. However for high code rates, this advantage diminishes since the memory access may need to be serialized. If we consider a fixed number of memory accesses per cycle then both the bit-node and check-node centric architectures require approximately the same number of cycles per iteration. The last column in the table shows the number of cycles needed per iteration when parallel memory access is restricted to d_c elements per cycle.

IV. Bit-Node Centric Architecture for an LDPC Decoder in a Concatenated System

A receiver where the LDPC decoder is concatenated with a soft-input soft-output (SISO) detector is shown in Fig. 3. The SISO detector takes in channel outputs and extrinsic information from the LDPC decoder and produces its own extrinsic information which becomes the *a priori* LLR (λ_i) for the LDPC decoder at the next iteration. Within one iteration through the concatenated system, the data is processed once by both the detector and LDPC decoder. (In some receiver systems the LDPC decoder performs additional internal decoding iterations. In this paper we only consider one single local decoding iteration for each global iteration through the system). In this concatenated system, LDPC decoding involves passing the *a priori* LLRs along the edges to the check nodes, processing the check nodes and computing extrinsic information at the bit nodes.

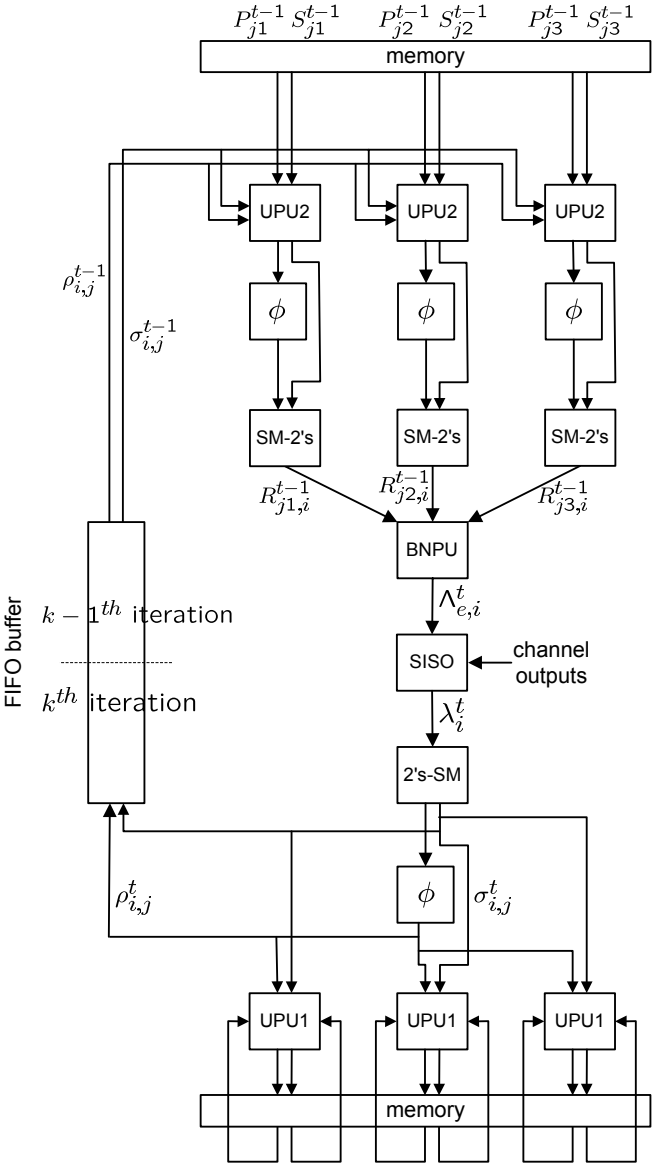

Fig. 4. Block diagram of the proposed architecture for a concatenated detector-decoder system where $d_c = 3, \mathcal{B}_i = \{j1, j2, j3\}$.

Since the LDPC decoder performs only one local iteration per global iteration, the processing inside the LDPC decoder is given by equations (1) - (4) where the LDPC iteration index equals $k = 1$. Further, since $R_{i,j}^0 = 0, \forall i, j$ based on (1), the messages passed along the edges from bit node i to check node j are given by:

$$\hat{Q}_{i,j} \triangleq Q_{i,j}^1 = \lambda_i, \ \ \forall j \in \mathcal{B}_i. \tag{15}$$

Thus, at each pass through the LDPC decoder, all the messages from bit node i to its check nodes, namely $\hat{Q}_{i,j}$, $\forall j \in \mathcal{B}_i$ are equal to the *a priori* LLR of the bit node i. The global iteration index is not shown in (15). Denoting the global iteration index

1930-529X/07/$25.00 © 2007 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2007 proceedings.

as t and the *a priori* LLR pertaining to the i^{th} bit node as λ_i^t, the messages from the i^{th} bit node to the j^{th} check node at the t^{th} iteration are given by:

$$\hat{Q}_{i,j}^t = \lambda_i^t, \ \forall j \in \mathcal{B}_i. \tag{16}$$

Considering this simplification, the transformed magnitude and sign bit information needed to be saved per bit node are given by:

$$\hat{\rho}_i^t = \phi(|\lambda_i^t|) \tag{17}$$
$$\hat{\sigma}_i^t = \text{sign}(\lambda_i^t). \tag{18}$$

Utilizing these quantities the check-to-bit node messages, $R_{j,i}^{t-1}, j \in \mathcal{B}_i$ are computed in a similar fashion as described in Section III. Based on these check-to-bit node messages, the extrinsic information can be calculated as

$$\Lambda_{e,i}^t = \sum_{l \in \mathcal{B}_i} R_{l,i}^{t-1} \tag{19}$$

and is passed to the SISO detector for the next iteration.

Fig. 4 shows the block diagram of the proposed architecture for the concatenated detector-decoder system. In the Figure, the exemplary case is considered where $d_c = 3, \mathcal{B}_i = \{j1, j2, j3\}$. The BNPU calculates the extrinsic information according to (19). The UPU1s and UPU2s are implemented as shown in Fig. 2(b) and Fig. 2(c), respectively.

Since the messages from a given bit node to all its check nodes are the same, the FIFO buffer storage capacity is reduced by a factor of d_c. Hence, the buffer requires q bits per row to save the intermediate messages pertaining to a single bit node. Storing all $[S_j^t, P_j^k]$ and $[S_j^{t-1}, P_j^{k-1}], j \in \{1...m\}$ requires $2mq$ memory bits. The total storage requirement is $(2m + n)q$. At each time cycle the proposed architecture computes d_c check-to-bit node messages and a single extrinsic information value.

The advantages of the proposed bit-node centric architecture are significant for concatenated detector-decoder systems: No additional delay is needed for LDPC decoding. The delay for the decoder is in fact absorbed by the SISO detector. The detector generates extrinsic soft outputs, λ_i^ts which are immediately processed in the same order by the decoder without the need for buffering. Moreover the extrinsic information values generated by the decoder are in the same order as the channel outputs and can be immediately processed by the SISO detector for the next iteration.

On the other hand, the check-node centric architecture is less hardware-efficient in a concatenated detector-decoder system. The check-node centric architecture processes the extrinsic soft outputs from the channel, λ_i^ts in a different order than they are generated by the detector. This is due to the fact that the d_r number of λ_i^ts required for each check node are different. Therefore, first, the λ_i^ts need to be buffered before check node computations are performed. The extrinsic information can be computed on the fly by updating a bank of memory. The decoder needs m cycles in addition to the delay for the SISO detector. The serial architecture proposed by Yeo *et al.* [6] suffers from the same disadvantages when used in a concatenated detector-decoder system. Also Wu *et al.* [9] considered a concatenated detector-decoder system. There however, an architecture for the min-sum decoding algorithm was presented, which in general achieves worse bit error rate than the sum-product algorithm implemented by our architecture.

V. Conclusion

This paper presented a serial LDPC decoder architecture that performs the sum-product algorithm. Two applications were considered, namely a stand-alone LDPC decoder and an LDPC decoder concatenated with a SISO detector. The proposed architecture uses a standard bit node processing unit, while the check node computations are broken up and computed on the fly. This bit-node centric architecture requires significantly less memory than other serial architectures, especially for LDPC codes with high code rate. A simplified version of the architecture targeted at concatenated detector-decoder receivers was also proposed, which further reduces memory requirements and is associated with less latency than other architectures.

References

[1] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Info. Theory*, vol. 45, pp. 399–431, Mar. 1999.

[2] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Info. Theory*, vol. 47, pp. 599–618, Feb. 2001.

[3] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Info. Theory*, pp. 21–28, 1962.

[4] S. Y. Chang, G. D. Forney, T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 db of the Shannon limit," *IEEE Comm. Lett.*, vol. 5, pp. 58–60, Feb. 2001.

[5] A. Blanksby and C. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE J. Solid-State Circuits*, vol. 37, pp. 404–412, Mar. 2002.

[6] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "VLSI architectures for iterative decoders in magnetic recording channels," *IEEE Trans. Mag.*, vol. 37, pp. 748–755, Mar. 2001.

[7] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "High throughput low-density parity-check decoder architectures," *Proc. IEEE GLOBECOM*, vol. 5, pp. 3019–3924, Nov. 2001.

[8] M. M. Mansour and N. R. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. VLSI Systems*, vol. 11, pp. 976–996, Dec. 2003.

[9] Z. Wu and G. Burd, "Equation based LDPC decoder for intersymbol interference channels," *IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 5, pp. 757– 760, Mar. 2005.

[10] D. E. Hocevar, "LDPC code construction with flexible hardware implementation," *IEEE Int'l Conf. on Comm. (ICC)*, vol. 4, pp. 2708–2712, May 2003.

1930-529X/07/$25.00 © 2007 IEEE