Low-Power VLSI Decoder Architectures for LDPC Codes [*]

Mohammad M. Mansour and Naresh R.Shanbhag
iCIMS Research Center, ECE Dept.
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W. Main Street, Urbana, IL 61801
(mmansour,shanbhag)@uiuc.edu

ABSTRACT

Iterative decoding of low-density parity check codes (LDPC) using the message-passing algorithm have proved to be extraordinarily effective compared to conventional maximum-likelihood decoding. However, the lack of any structural regularity in these essentially random codes is a major challenge for building a practical low-power LDPC decoder. In this paper, we jointly design the code and the decoder to induce the structural regularity needed for a reduced-complexity parallel decoder architecture. This interconnect-driven code design approach eliminates the need for a complex interconnection network while still retaining the algorithmic performance promised by random codes. Moreover, we propose a new approach for computing reliability metrics based on the BCJR algorithm that reduces the message switching activity in the decoder compared to existing approaches. Simulations show that the proposed approach results in power savings of up to 85.64% over conventional implementations.

Categories and Subject Descriptors

B.7.1 [**Types and Design Styles**]: VLSI; E.4 [**Coding and Information Theory**]: Error control codes

General Terms

Design

Keywords

LDPC codes, lower power architectures, BCJR algorithm.

1. INTRODUCTION

Turbo codes [1] and LDPC codes [2] are the two best known codes that are capable of achieving low bit error rates (BER) at code rates approaching Shannon's capacity limit. However, in order to achieve desired power and throughputs for current applications (e.g., $>$ 1Mbps in 3G wireless systems, $>$ 1Gbps in magnetic recording systems), fully parallel and pipelined iterative decoder architectures are needed.

Compared to turbo codes, LDPC codes enjoy a significant advantage in terms of computational complexity and are known to have a large amount of inherent parallelism [3]. However, the randomness of LDPC codes results in stringent memory requirements that amount to an order of magnitude increase in complexity compared to those for turbo codes.

A direct approach to implementing a parallel decoder architecture would be to allocate, for each node or cluster of nodes in the graph defining the LDPC code, a function unit for computing the reliability messages, and employ an interconnection network to route messages between function nodes (see Fig.1). A major problem with this approach is that the interconnection networks require complex wiring to perform global routing of messages and hence must be deeply pipelined (e.g., bidirectional multilayered networks in [4] and 4096-input multiplexers per function unit in [5]). Moreover, the randomness in the pattern of communicating messages leads to routing and congestion problems on the networks which require extensive buffering to resolve.

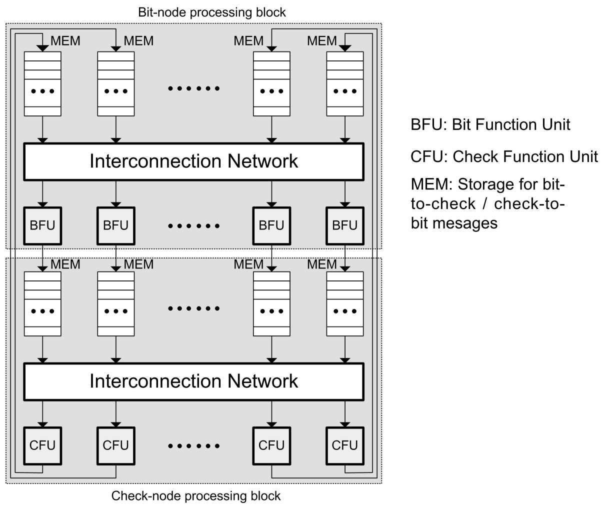


Figure 1: Fully parallel decoder architecture.

It is well known that global interconnects primarily determine the system performance in modern semiconductor processes. This suggests exploiting data locality to reduce communication overhead in a parallel VLSI implementation, and

[*]This work was supported with funds from NSF under grants CCR 99-79381 and CCR 00-73490.

motivates a *joint code-decoder design* approach to construct a class of LDPC codes having the structural regularity and locality properties favorable for a practical parallel architecture. Previous LDPC decoder architectures [3]-[6] have not addressed the interconnection network problem in random LDPC codes. Issues related to routing conflicts on the network and scheduling in the function units significantly increase power consumption and degrade system performance and hence need to be accounted for explicitly.

All current implementations of the message-passing algorithm employ Gallager's original algorithm [2] for computing reliability messages. However, due to quantization effects, this algorithm suffers from performance loss and requires large number of iterations to converge, which increases the switching activity, power consumption and decoding latency.

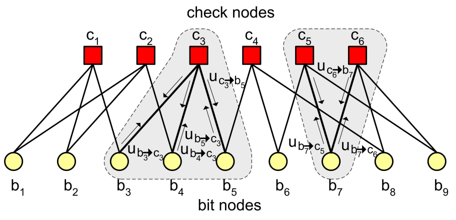
The contributions of this paper are twofold. First, we propose a new interconnect-driven LDPC code design approach that eliminates the power problem in the interconnection network by inducing desired regularity characteristics in the code. Second, we propose an optimized version of the BCJR algorithm [7] to compute reliability messages, which minimizes the effect of quantization noise on algorithmic performance and improves the power consumption. Thus, a low-power VLSI architecture for the decoder is obtained by optimizing both the interconnection network and the function units.

The rest of the paper is organized as follows. Section 2 presents a joint code-decoder design approach for the design of low-power decoder architectures for LDPC codes. Section 3 proposes a new optimized version of the BCJR algorithm for reducing power in the function units. Section 4 presents some simulations results and Section 5 provides some concluding remarks.

2. LOW-POWER LDPC DECODER ARCHITECTURES

After briefly introducing LDPC codes, the interconnect-driven LDPC code design approach is proposed in section 2.1. The resulting decoder architecture and its memory synchronization scheme are presented in sections 2.2 and 2.3.

An LDPC code is described by a bipartite graph (see Fig. 2) whose *reduced* adjacency matrix is $\mathbf{H} = [h_{ij}]_{m\times n}$ having m *check* nodes $\{c_1, c_2, \cdots, c_m\}$ and n *bit* nodes $\{b_1, b_2, \cdots, b_n\}$ corresponding to the m rows and n columns of $\mathbf{H}$, respectively. A bit node b_j is connected to a check node c_i if the entry h_{ij} of $\mathbf{H}$ is 1. In a *regular* (r, c)-LDPC code, bit nodes have degree c and check nodes have degree r in the graph. Typically, regular codes are easier to encode and have a simpler decoder architecture than irregular codes, however the latter achieve higher coding gain.



$$\begin{bmatrix}111000000\\110100000\\001110000\\000011010\\000001101\\000000111\end{bmatrix}$$

Figure 2: Bipartite graph and parity check matrix H of a regular (2,3)-LDPC code of length 9.

LDPC codes are decoded iteratively using Gallager's message passing (MP) algorithm [2]. It is based on evaluating extrinsic reliability values associated with each bit using disjoint parity check equations that the bit participates in [2]. Each iteration is composed of a two-phase schedule in which updates of all check nodes are done in **phase 1** by sending messages ($\mu_{c\to b}$) to neighboring bit nodes, and then updates of all bit nodes are done in **phase 2** by sending messages ($\mu_{b\to c}$) to neighboring check nodes (see Fig. 2). Updates in each phase are independent and can be parallelized.

For randomly constructed LDPC codes, the non-zero entries in $\mathbf{H}$ are randomly distributed across the rows and columns (while still satisfying the regularity constraints), which makes the interconnection networks in Fig 1 for communicating messages from check nodes to bit nodes, and vice versa, very complex. In Section 2.1, we propose an approach for designing the codes, i.e., the matrix $\mathbf{H}$, such that this interconnection complexity is minimized.

2.1 Interconnect-Driven LDPC Code Design

We propose to construct short length LDPC codes ($< 2K$ bits) having simple graph connectivity properties while still maintaining the performance of randomly constructed codes. One easy way to achieve the desired structural properties is to partition the parity check matrix $\mathbf{H}$ into blocks of $p \times p$ matrices, for some appropriately chosen p, such that each bit in a block participates in only one check equation in the block, and each check equation in the block involves only one bit from the block. If $\mathbf{H}$ contains r blocks per column and $c > r$ blocks per row, this construction defines a regular (r, c)-LDPC code of length $n = cp$ and dimension $k \geq (c - r)p$. For example, Fig. 3 shows the partitioning of $\mathbf{H}$ into $r = 3$ rows and $c = 5$ columns of $p \times p$ matrices.

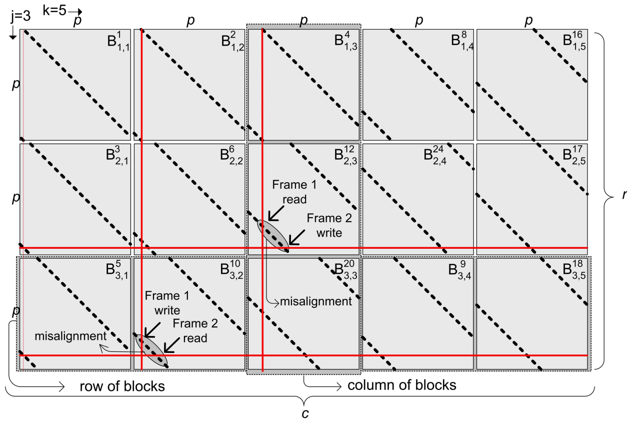


Figure 3: Partitioning of H into $r = 3$ rows and $c = 5$ columns of $p \times p$ block matrices. Dotted lines represent entries of 1 in H, other entries are 0.

The easiest way to define the $p \times p$ block matrices is by suitable permutations of the rows of the identity matrix $\mathbf{I}_{p\times p}$ in an analogous fashion to the construction of BCH codes. Let $\mathbf{B}^i_{j,k}$ be an $\mathbf{I}_{p\times p}$ identity matrix located at the jth block row and kth block column of the parity check matrix having its rows shifted to the right $i \mod p$ positions for $i \in \mathsf{S} = \{0, 1, 2, \cdots, p-1\}$. Assume there exists a q such that $q^c \equiv 1 \pmod p$. Then the operation of multiplying by $q \mod p$ divides S into cyclotomic cosets mod p. A cyclotomic coset containing the integer

s is the set $\{s, sq, sq^2, \cdots, sq^{m_s-1}\}$ where m_s is the smallest positive integer satisfying $sq^{m_s} \equiv s \pmod{p}$. For example, if $p = 31$ and $c = 5$, then $q = 2$ and the cosets are $\mathsf{C}_0 = \{0\}$, $\mathsf{C}_1 = \{1, 2, 4, 8, 16\}$, $\mathsf{C}_3 = \{3, 6, 12, 24, 27\}$, $\mathsf{C}_5 = \{5, 10, 20, 9, 18\}$, $\cdots$, $\mathsf{C}_{15} = \{15, 30, 29, 27, 23\}$. The elements of r of these sets are then used to set i in blocks $\mathbf{B}^i_{j,k}$ for $j = 1, \cdots, r$, $k = 1, \cdots, c$. The matrix shown in Fig. 3 defines a regular $(3, 5)$-LDPC code constructed for $p = 31$ using the cosets $\mathsf{C}_1, \mathsf{C}_3, \mathsf{C}_5$. Note that the locations of 1's in $\mathbf{H}$ can be determined using the sets $\mathsf{C}_1, \cdots, \mathsf{C}_r$ and the parameter p.

Figure 4 compares the performance over an AWGN channel of two codes constructed from cyclotomic cosets, a $(3, 5)$-LDPC code of length 1055, rate 0.4 and a $(3, 5)$-LDPC code of length 305, rate 0.4, versus two randomly constructed codes of similar complexity. As shown the codes compare well and even outperform randomly constructed codes.

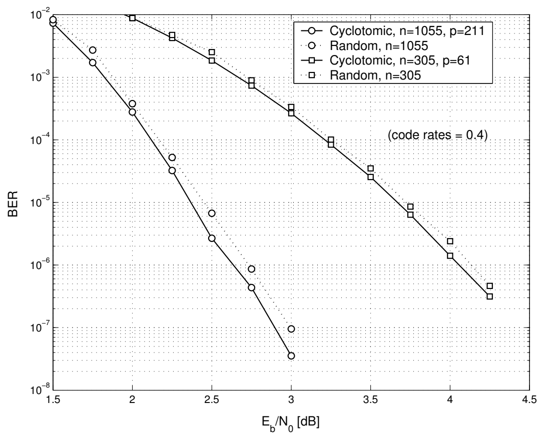


Figure 4: BER vs. SNR performance curves.

2.2 Interconnect-Driven Low-Power Decoder Architecture

This subsection presents a parallel decoder architecture for the LDPC codes designed via the interconnect-driven code construction method described in Section 2.1. The codes have length $n = cp$ and their bipartite graph has cp bit nodes and rp check nodes.

The proposed decoder in Fig. 5 is composed of two main blocks, BLOCK1, the bit-node processing block and BLOCK2, the check-node processing block. Two frames are processed simultaneously by the decoder: **phase 2** of the first frame is performed in the check node processing block while **phase 1** of the second frame is performed in the bit-node processing block. BLOCK1 contains c Bit Function Units (BFU's) and c memory banks for storing the check-to-bit messages ($\mu_{c\to b}$) obtained from BLOCK2 during the previous iteration. Each memory bank consists of r memory blocks and a set of r counters for address generation in one-to-one correspondence with a single *column* of $p \times p$ blocks in the parity check matrix. In addition, there are 2 memory blocks for storing the intrinsic reliability metrics (IRM) of the two frames obtained from the channel. The ith memory block in the jth bank containing the bit messages is denoted by $\text{MemB}_{i,j}$, the block containing the intrinsic messages by IRM_j, and the corresponding counters for memory access are denoted by $\text{RD1}_{i,j}$. One BFU operates on a single memory bank by fetching $r + 1$ operands in parallel from the $r + 1$ memory blocks every cycle and storing the results in the appropriate memory bank in BLOCK2.

Similarly, BLOCK2 is composed of r Check Function Units (CFU's) and r memory banks for storing the bit-to-check messages ($\mu_{b\to c}$) obtained from BLOCK1 during the previous iteration. Each memory bank consists of c memory blocks and c counters in one-to-one correspondence with a single *row* of $p \times p$ blocks in the parity check matrix. The ith memory block in the jth bank containing the check messages is denoted by $\text{MemC}_{i,j}$ and the corresponding counters by $\text{RD2}_{i,j}$. One CFU is allocated to each memory bank with c operands fetched in parallel from the c memory blocks every cycle and the results are stored in the appropriate memory banks in BLOCK1.

2.3 Memory Synchronization

The operations of BLOCK1 and BLOCK2 on the two frames simultaneously are synchronized dynamically through the set of counters $\text{RD1}_{i,j}$ and $\text{RD2}_{i,j}$ in the memory banks. $\text{RD1}_{i,j}$ runs over the column indices of the block matrix $\mathbf{B}_{i,j}$ in $\mathbf{H}$ and $\text{RD2}_{i,j}$ runs over its row indices. At startup, BLOCK1 is initialized with intrinsic reliability messages from the channel corresponding to **frame 1** and **frame 2** and the counters in BLOCK1 are initialized as $\text{RD1}_{i,j} = \mathsf{C}_{1,j}$ for all i, j, and in BLOCK2 as

$$\text{RD2}_{i,j} = \begin{cases} 1 & \text{if } i = 1, \\ \mathsf{C}_{1,1} - \mathsf{C}_{i,1} \pmod{p} & 2 \le i \le r, \end{cases}$$

for $j = 1, \cdots, c$. The function units in BLOCK1 process the messages for **frame 1** and store the results in the memory blocks $\text{MemC}_{i,j}$ in BLOCK2. Then, BLOCK1 and BLOCK2 alternate their operations between **frame 1** and **frame 2** in each processing phase. The function units BFU_i and CFU_j first read their operands from their respective memory banks according to RD1 and RD2. The messages in these locations are not needed anymore so they can be overwritten with new results from the opposite BLOCK. For this to be possible, the messages for both frames in BLOCK1 and BLOCK2 must be consumed and produced in a coherent manner to avoid new messages overwriting unprocessed messages. It can be shown that the misalignment of the messages in both BLOCKs can be adjusted by incrementing counters $\text{RD1}_{i,j}$ and $\text{RD2}_{i,j}$ by

$$(\mathsf{C}_{1,j} - \mathsf{C}_{i,j}) - (\mathsf{C}_{1,1} - \mathsf{C}_{i,1}) \pmod{p}$$

for $2 \le i \le r$ and $2 \le j \le c$, at the end of every iteration. Since the first row and column of blocks of the parity check matrix can be taken as references, counters $\text{RD1}_{i,1}$, $\text{RD1}_{1,j}$, $\text{RD2}_{i,1}$, $\text{RD2}_{1,j}$ are not updated.

As an example, the middle block in Fig. 3 shows the misalignment of data between the frames (with reference to the vertical and horizontal lines) as a result of BLOCK2 overwriting messages from **frame 1** still unprocessed by BLOCK1. Thus, BLOCK2 instead overwrites the locations already processed by BLOCK1 with updated messages for **frame 2**, and the counters for read access are updated to account for the new positions of the messages in the memory at the end of every iteration. Note that this addressing scheme *completely eliminates the need for a multilayered interconnection for accessing messages from the memory banks such as the one proposed in* [4].

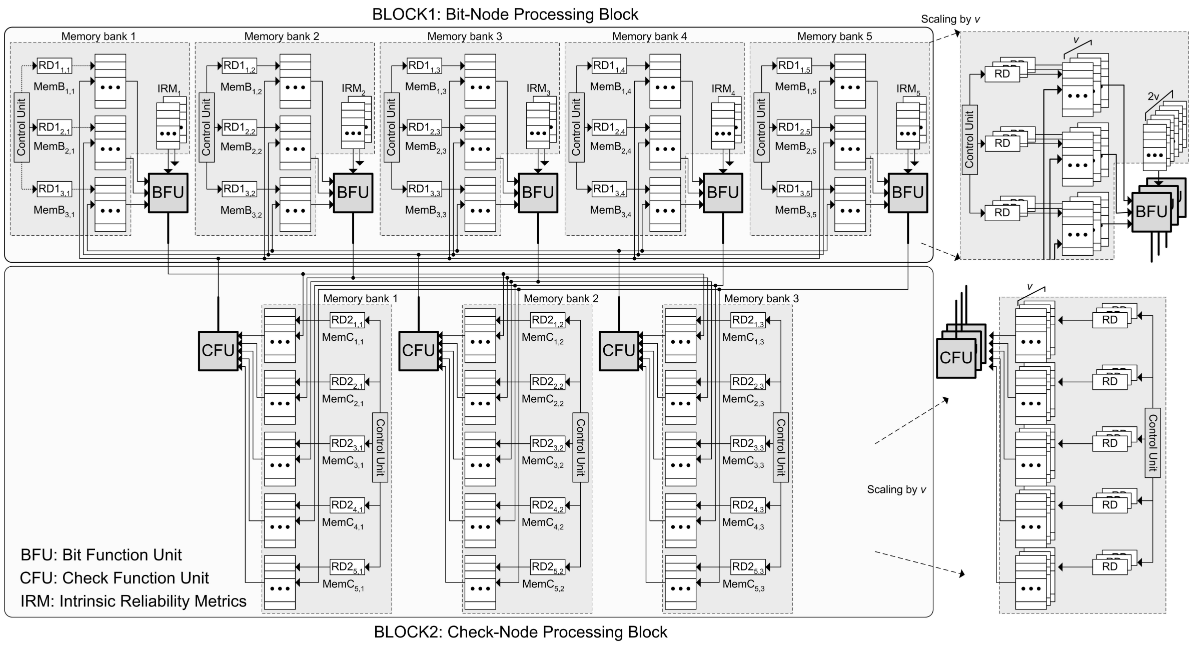


Figure 5: Decoder architecture for a regular $(r = 3, c = 5)$-LDPC code. It is composed of 2 blocks, the bit-node processing block and check-node processing block that perform the two phases of Gallager's message-passing algorithm. The decoder processes 2 frames simultaneously by alternating between each phase of both frames in BLOCK1 and BLOCK2. The dynamic addressing scheme eliminates the need for an interconnection network.

The decoder completes one decoding iteration for 2 frames in $2p$ clock cycles. For I iterations per frame and clock frequency f, the throughput is cf/I bps. The throughput can be scaled up by a factor of ν by dividing the memory blocks in each memory bank in BLOCK1 and BLOCK2 into ν sub-blocks such that every c sub-blocks in BLOCK1 (respectively r sub-blocks in BLOCK2) form a sub-bank on which a BFU (CFU) operates as shown on the right side of Fig. 5. Note however that the interconnection complexity also increases.

3. LOW-POWER BCJR-BASED CFU ARCHITECTURE

In this section, we present a low-power approach based on the BCJR algorithm [7] to generate the check-to-bit messages performed by the CFU blocks in Fig. 5. The original algorithm for computing the messages is summarized by the following two log-domain equations [2]:

$$\mu_{c_j \to b_i} = \psi^{-1}\Big[\sum_{i' \in R[j]\setminus i} \psi(\mu_{b_{i'} \to c_j})\Big] \cdot \delta(i,j) \qquad (1)$$

$$\mu_{b_i \to c_j} = \sum_{j' \in C[i]\setminus j} \mu_{c_{j'} \to b_i} + \Lambda_{\text{in}}(i) \qquad (2)$$

where $\mu_{c_j \to b_i}$ is the message from check j to bit i, $\mu_{b_i \to c_j}$ is the message from bit i to check j, $\psi(x) = -\log(|\tanh(x/2)|)$ is the Gallager function with inverse $\psi^{-1}(x) = 2\tanh^{-1}(e^{-x})$ $= \psi(x)$, $R[j]$ is the set of bits involved in check j, $C[i]$ is the set of checks involving bit i, $\delta(i,j)$ is ± 1 depending on the terms in the summation of (1) and $|R[j]|$, and $\Lambda(i)$ is the intrinsic reliability metric of bit i.

Typically $\psi(x)$ is implemented as a small look-up table (LUT). Equation (1) however is prone to quantization noise due to the non-linear function $\psi(x)$ and its inverse. Figure 6(a) compares the composite function $\psi^{-1}(\psi(x))$ using floating point values with a fixed-point version quantized to 5 bits. As shown the function $\psi(x)$ is not invariant under its inverse and the representable dynamic range is limited. If the argument of $\psi(x)$ is scaled to increase the dynamic range, the quantization levels become coarser and deviate from the ideal line. These disadvantages translate to algorithmic performance loss where for a given SNR more iterations are needed for convergence increasing the decoding latency, switching activity and hence the power consumption of the decoder. This problem has been overlooked in all current implementations which solely resort to scaling the input to mitigate quantization effects.

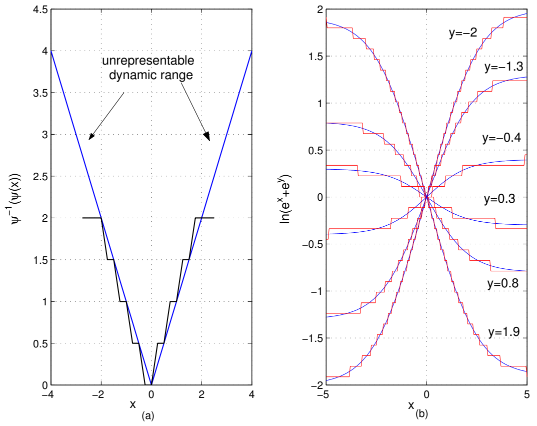


Figure 6: Quantization effects on the (a) Gallager function (b) BCJR recursion functions.

We employ the BCJR algorithm [7], typically used in turbo codes, to compute the check-to-bit messages by tailoring it to the trellis of a single parity check equation shown in Fig 7. The BCJR algorithm computes a posteriori reliability metrics (check-to-bit messages) on the trellis using the bit-to-check messages according to eqs. (3)-(5):

$$\alpha_1' = \ln\left(e^{\alpha_1+\lambda_1} + e^{\alpha_2+\lambda_2}\right) \qquad \alpha_2' = \ln\left(e^{\alpha_1+\lambda_2} + e^{\alpha_2+\lambda_1}\right) \quad (3)$$

$$\beta_1' = \ln\left(e^{\beta_1+\lambda_1} + e^{\beta_2+\lambda_2}\right) \qquad \beta_2' = \ln\left(e^{\beta_1+\lambda_2} + e^{\beta_2+\lambda_1}\right) \quad (4)$$

$$\Lambda = \ln\left(e^{\alpha_1+\beta_2} + e^{\alpha_2+\beta_1}\right) \quad - \quad \ln\left(e^{\alpha_1+\beta_1} + e^{\alpha_2+\beta_2}\right) \qquad (5)$$

where $\lambda_2 - \lambda_1 = \mu_{b\to c}$ and $\Lambda = \mu_{c\to b}$. Equations (3) and (4) are called the forward and backward state metric recursions, respectively. Referring to Fig. 7, the forward recursion is first performed on all trellis states from left to right and the results are stored. Next, the backward recursion is performed from right to left in parallel with the output metric computation given by (5) using the stored forward state metrics.

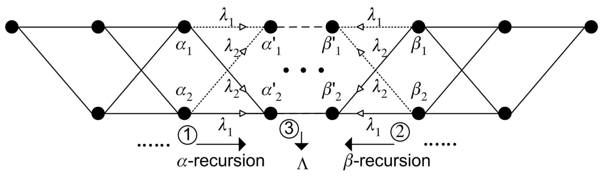


Figure 7: Trellis of a parity check equation.

In the context of LDPC codes, the length of the trellis is the number of bits per check node, or c for regular (r, c)-LDPC codes, which is typically very small ($5 \sim 15$). To minimize the effects of quantization noise on the computations, eqs. (3)-(5) can be transformed so that metric differences $\Delta\alpha \triangleq \alpha_2 - \alpha_1$, $\Delta\beta \triangleq \beta_2 - \beta_1$, and $\Delta\lambda \triangleq \lambda_2 - \lambda_1$ rather than absolute metrics are involved. Such a transformation maximizes the effectiveness of the limited dynamic range of the metrics and eliminates the need for normalization to avoid overflow. Moreover, only one set of forward state metrics instead of two needs to be stored. The transformed equation for (3) can be written in the following simple form:

$$\Delta\alpha' = \begin{cases} -\Delta\lambda + Q(-\Delta\lambda) & \text{if } \Delta\alpha \geq |\Delta\lambda|, \\ \Delta\alpha + Q(\Delta\alpha) & \text{if } -\Delta\lambda \leq \Delta\alpha \leq \Delta\lambda, \\ -\Delta\alpha + Q(-\Delta\alpha) & \text{if } \Delta\lambda \leq \Delta\alpha \leq -\Delta\lambda, \\ \Delta\lambda + Q(\Delta\lambda) & \text{if } \Delta\alpha \leq -|\Delta\lambda| \end{cases} \qquad (6)$$

where $Q(x) = [\alpha_2' - \alpha_1' - x]_b$ is a correction factor, $[\cdot]_b$ denotes quantization to b bits, and $\Delta\lambda = \mu_{b\to c}$ are the appropriate bit-to-check messages. Similar equations for (4) can be derived by replacing $\Delta\alpha$ by $\Delta\beta$ in (6), and for (5) by replacing $\Delta\lambda$ by $\Delta\beta$ in (6). Note that (6) can be implemented simply as a Compare-Select (CS) block using two comparators, two 2's complementers, a multiplexer and a LUT which stores the sum $[x + Q(x)]_b$ as shown in Fig. 8(a), assuming comparators are implemented using adders. A similar argument holds for $\Delta\beta$ and Λ. Figure 8(b) shows the structure of the Check Function Node (CFU) constructed from the 3 CS blocks implementing the $\Delta\alpha$, $\Delta\beta$, and Λ equations. Note that 2 simple FIFO buffers are used to store the forward state metric differences $\Delta\alpha$ and the messages $\mu_{b\to c}$. This CFU block has comparable complexity to a CFU block implementing (1) using a tree-adder, sign correction logic and a pair of LUT's.

Figure 9 shows the Bit Function Unit implementing (2). Note that the only difference among the messages computed in (2) is the effect of the check node under consideration. Hence, the equation can be implemented using the Save-Add-Subtract (SAS) structure where the operands are first saved and then the summation is computed serially (Fig. 9(a)) or using a tree-adder (Fig. 9(b)), and then the appropriate operands are subtracted to generate all the messages.

In Fig. 6(b) we compare the quantization effects on the computation of the check-to-bit messages using the transformed equations of the BCJR algorithm. The entire dynamic range can now be represented, and the LUT provides quantized values that closely track the desired output compared to the quantized Gallager equations in Fig. 6(a). It is worth mentioning the absence of adders required to compute the metrics needed in (1) which are subject to quantization noise under $\psi^{-1}(\psi(\cdot))$.

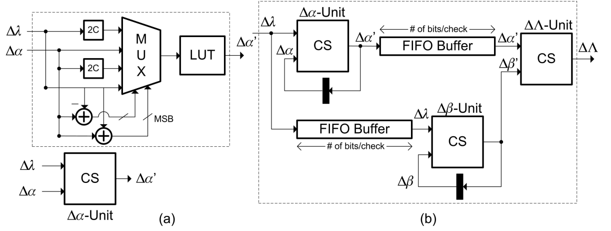


Figure 8: (a) Compare-Select unit implementing the forward state metric differences $\Delta\alpha$ of (6) (b) The check function unit.

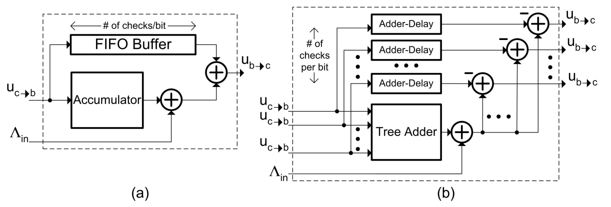


Figure 9: Bit function unit: (a) serial implementation (b) parallel implementation.

To compare the effect of quantization on the algorithmic performance of the message-passing algorithm using Gallager's method and the proposed method, a low-level simulator was developed for both algorithms, and the results are shown in Fig. 10. Figures 10(a)-(b) show the percentage of valid frames decoded and the number of iterations required for convergence, respectively, using the unquantized Gallager algorithm, a 6-bit quantized Gallager algorithm, and a 6-bit quantized BCJR algorithm. Figures 10(c)-(f) show the results for 5-bit and 4-bit quantization, respectively. The results demonstrate that the optimized BCJR algorithm is superior to the original algorithm particularly for 4-bit quantization where it attains more than 1 dB of improvement in coding gain. Moreover, the BCJR algorithm quantized to 5 bits achieves even better performance than a 6-bit quantized Gallager algorithm. The average improvement in coding gain achieved at 4-bit, 5-bit, and 6-bit quantization levels is 5.2%, 11.98%, and 119.19%, respectively. Note also the reduction in switching activity due to the decrease in the number iterations.

4. SIMULATION RESULTS

To simulate the power savings of the decoder architecture resulting from applying the interconnect-driven design approach and the calculation of the reliability messages via the proposed BCJR approach, the architecture (**A1**) for a regular $(3, 5)$-LDPC code of length $n = 1055$ and rate 0.4 constructed from cyclotomic cosets is used as an example. The results are compared with the decoder architecture (**A2**) for a random LDPC code of similar complexity constructed using interconnection networks like the ones shown in Fig. 1 and utilizing Gallager's original algorithm for computing reliability messages. Architecture **A2** has twice the memory requirements of **A1** and operates on a single frame, while **A2** decodes two frames simultaneously. Note that due to the irregularity of the code matrix in **A2**, the two phases of computations, bit-to-check and check-to-bit, cannot be overlapped which necessitates two copies of each memory bank that alternate between read/write. The algorithmic performance of both codes over an AWGN channel is shown in Fig. 4.

The memory blocks for **A1** were designed as circular buffers, while those for **A2** were built as FIFO buffers and stacks similar to [5]. The interconnection networks for **A2** were built using 631-input multiplexers and 210-output demultiplexers that serve as read and write ports for the CFU blocks, and similarly for the BFU blocks using 1055-input multiplexers and 1055-output demultiplexers as read/write ports.

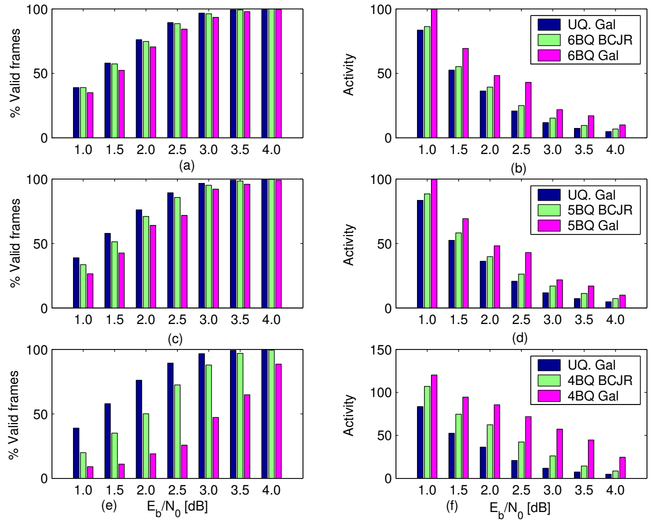


Figure 10: Comparison of quantization effects on the performance of Gallager's algorithm vs. the BCJR algorithm 6-bit (a)-(b), 5-bit (c)-(d), and 4-bit (e)-(f) quantization levels.

Power estimates for the CFU and BFU blocks used in both architectures were obtained by synthesizing the blocks in Figs. 8 and 9(b) (for **A1**) and their counterparts for **A2** using 3.3V 0.35 μm standard CMOS technology. Similarly, power estimates for the circular and FIFO buffers, stacks and (de)multiplexers were obtained through synthesis. The switching activity for the logic was estimated through simulations for different SNR.

Figure 11 shows the power consumed by **A1** and **A2** as a function of SNR for the same algorithmic performance. At the same SNR, **A1** implemented using 5-bit datapaths achieves comparable and even better performance than **A2** implemented using 6-bit datapaths mainly due to quantization effects in the CFU blocks as seen in Fig. 10, and partly due to the better performance of the cyclotomic code itself over the random one as shown in Fig. 4. As seen from the figure, the power savings range between 82.42% and 85.64%. The power consumed by **A2** at SNR of 4 dB is almost the same as that consumed by **A1** at SNR of 1.5 dB which reflects the reduction in switching activity in the decoder due to faster convergence as a function of SNR.

Figure 12 shows a breakdown of the power distribution as a function of SNR in **A1** and **A2** in terms of message computations, communication, and memory accesses. The figure demonstrates the effectiveness of the interconnect-driven code design approach in eliminating the effect of the interconnection network, and the efficiency of the dynamic memory addressing scheme in reducing the power consumption in the memory units by approximately 80%. The power savings due to computations demonstrates that, contrary to common perception, the optimized BCJR when tailored to the trellis of a single parity check equation is a viable alternative to the Gallager algorithm.

5. CONCLUSIONS

The LDPC decoding algorithm achieves significant coding gain but at the expense of large power consumption in the decoder due to the lack of structural regularity and the inefficiency of the algorithm employed for computing reliabilities. We have shown that through an interconnect-driven code design approach, coupled with a dynamic addressing scheme and an optimized version of the BCJR algorithm for computing reliabilities, power savings of up to 85.64% can be achieved.

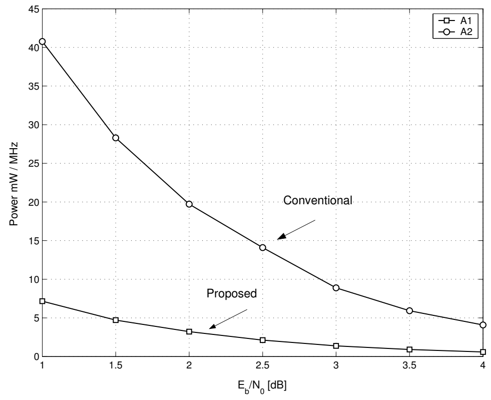


Figure 11: Power consumption for A1 and A2 at same algorithmic performance as a function of SNR.

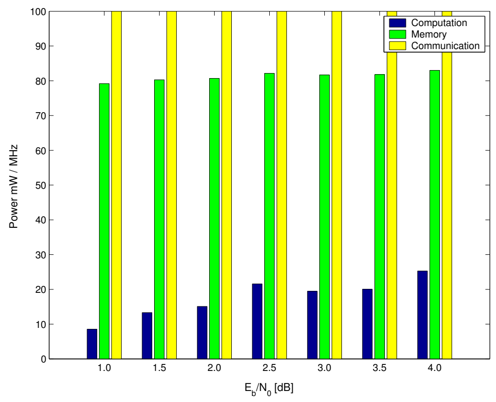


Figure 12: Breakdown of power savings in terms computations, communication and memory operations for A1 and A2 as a function of SNR.

6. ACKNOWLEDGEMENTS

The authors would like to thank Makram M. Mansour for providing the 3.3V 0.35 μm CMOS parameterized-cell library.

7. REFERENCES

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes," in *IEEE ICC*, 1993, pp. 1064–1070.

[2] R. G. Gallager, *Low-Density Parity-Check Codes*, MIT Press, Cambridge, MA, 1963.

[3] G. Al-Rawi and J. Cioffi, "A highly efficient domain-programmable parallel architecture for LDPCC decoding," in *ITCC 2001*, April 2001, pp. 569–577.

[4] T. Zhang and K. K. Parhi, "VLSI implementation-oriented (3,k)-regular low-density parity-check codes," in *SiPS 2001, Antwerp, Belgium*, Sept. 2001, pp. 25–36.

[5] E. Yeo et al., "VLSI architectures for iterative decoders in magnetic recording channels," *IEEE Trans. on Magnetics*, vol. 37, no. 2, pp. 748–755, March 2001.

[6] C. Howland and A. Blanksby, "Parallel decoding architectures for low density parity check codes," in *Proc. of 2001 IEEE ISCS, Sydney*, May 2001, pp. 742–745.

[7] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. on I.T.*, pp. 284–287, Mar. 1974.