VLSI Architectures for Multi-Gbps Low-Density Parity-Check Decoders

by

Ahmad Darabiha

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
University of Toronto

VLSI Architectures for Multi-Gbps Low-Density Parity-Check Decoders

Ahmad Darabiha
Doctor of Philosophy, 2008
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Near-capacity performance and parallelizable decoding algorithms have made Low-Density Parity Check (LDPC) codes a powerful competitor to previous generations of codes, such as Turbo and Reed Solomon codes, for reliable high-speed digital communications. As a result, they have been adopted in several emerging standards. This thesis investigates VLSI architectures for multi-Gbps power and area-efficient LDPC decoders.

To reduce the node-to-node communication complexity, a decoding scheme is proposed in which messages are transferred and computed bit-serially. Also, a broadcasting scheme is proposed in which the traditional computations required in the sum-product and min-sum decoding algorithms are repartitioned between the check and variable node units. To increase decoding throughput, a block interlacing scheme is investigated which is particularly advantageous in fully-parallel LDPC decoders. To increase decoder energy efficiency, an efficient early termination scheme is proposed. In addition, an analysis is given of how increased hardware parallelism coupled with a reduced supply voltage is a particularly effective approach to reduce the power consumption of LDPC decoders.

These architectures and circuits are demonstrated in two hardware implementations. Specifically, a 610-Mbps bit-serial fully-parallel (480, 355) LDPC decoder on a single Altera Stratix EP1S80 device is presented. To our knowledge, this is the fastest FPGA-based LDPC decoder reported in the literature. A fabricated 0.13-μm CMOS bit-serial (660, 484) LDPC decoder is also presented. The decoder has

a 300 MHz maximum clock frequency and a 3.3 Gbps throughput with a nominal 1.2-V supply and performs within 3 dB of the Shannon limit at a BER of 10^{-5}. With more than 60% power saving gained by early termination, the decoder consumes 10.4 pJ/bit/iteration at E_b/N_0=4dB. Coupling early termination with supply voltage scaling results in an even lower energy consumption of 2.7 pJ/bit/iteration with 648 Mbps decoding throughput.

The proposed techniques demonstrate that the bit-serial fully-parallel architecture is preferred to memory-based partially-parallel architectures, both in terms of throughput and energy efficiency, for applications such as 10GBase-T which use medium-size LDPC code (e.g., 2048 bit) and require multi-Gbps decoding throughput.

Acknowledgment

I feel truly honored for having the chance to work with Prof. Anthony Chan Carusone and Prof. Frank R. Kschischang as my Ph.D. advisors. I would like to thank them both for their academic and personal mentorship throughout this work. The completion of this project would not have been possible without Tony's continuous support, guidance, dedication and friendship. I have also learned so much from Frank's insightful research approach, skillful teaching methods, and his inspiring weekly group meetings.

I thank the members of my PhD supervising committee, Prof. Glenn Gulak and Prof. Roman Genov, for their insightful suggestions during the entire course of this work. I also thank the external members of my final oral examination, Prof. Wei Yu and Prof. Bruce F. Cockburn, for their time and for their valuable comments.

I gratefully acknowledge access to fabrication and CAD tools from Gennum Corporation. I also thank the FPGA research group at UofT for providing access to the Transmogrifier-4 prototype board.

I am grateful beyond measures to my dear parents, Mohammad Karim and Saeedeh, who have been my best teachers in countless ways. This thesis is dedicated to them, in the hope to acknowledge a tiny fraction of their unconditional and everlasting love and support.

I thank my lovely sister, Ladan, and my dear brothers, Mehdi and Majid, who have been my constant source of encouragement even though we have been more than eight time zones away for most of the past few years. I also thank my uncles and their dear families in Mississauga. Their kindness and hospitality have always made me feel like being close to home.

And last, but definitely not least, I feel blessed for getting to know so many good friends during my studies at UofT. I have learned a lot from them and I am grateful to them all. I thank Kostas Pagiamtzis, for being the most fun and knowledgeable cubicle mate I could ever hope for. Special thanks to Samira Naraghi for her invaluable and friendly consultations when they were badly needed. Many thanks to Amir Azarpazhooh for his true friendship (and for paying (most of) his dues in 'The wood game'!). Special thanks goes to Farsheed Mahmoudi for his energizing 'Zoor Khooneh' spirit. I also thank other friends from LP392, BA5000, BA5158, Tony's group, Frank's

group, and those from outside the department. In particular, I would like to thank Rubil Ahmadi, Mohamed Ahmed Youssef Abdulla, Masoud Ardakani, Navid Azizi, Mike Bichan, Horace Cheng, Yadi Eslami, Tooraj Esmailian, Vincent Gaudet, Amir Hadji-Abdolhamid, Afshin Haftbaradaran, Mohammad Hajirostam, Meisam Honarvar, Farsheed Mahmoudi, Mahsa Moallem, Alireza Nilchi, Ali Pakzad, Peter Park, Jennifer Pham, Mohammad Poustinchi, Saman Sadr, Sanam Sadr, Siamak Sarvari, Mahdi Shabany, Mehrdad Shahriari, Farzaneh Shahrokhi, Tina Tahmoures-zadeh, Maryam Tohidi, Alexi Tyshchenko, Marcus van Ierssel, and Kentaro Yamamoto, in the alphabetical order.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Error Control Codes

The objective in this work is to investigate VLSI architectures for high-throughput power-efficient low-density parity-check (LDPC) decoders for applications such as fiber-optic communications and 10 Gbps Ethernet. LDPC codes are a sub-class of linear error control codes (ECC) [3]. Error control coding—also referred to as channel coding—is a powerful technique in digital communications for achieving reliable communication over an unreliable channel. ECC has evolved significantly since the advent of information theory by Shannon in 1948 [4] and has become an essential transceiver block in a wide range of applications. Error control codes have received a lot of attention recently due to two main reasons. First, significant progress has been made by the information theory community in designing capacity-approaching codes. Second, progress in VLSI technology has enabled the implementation of computationally-intensive decoding algorithms.

Figure 1.1 shows block diagrams of a generic transmitter and receiver in a communication system incorporating channel coding. The encoder module in the transmitter maps the input sequence of information bits into codewords that include some redundancy. The task of the decoder block in the receiver is to use the redundancy added by the encoder to detect and correct the errors induced in the channel.

Fig. 1.2 illustrates the effect of channel coding on the bit error rate (BER) performance of a communication system. In this figure, the dotted curve corresponds to an uncoded system whereas the solid curve corresponds to a system with a (256, 215) Reed Solomon code [5]. Both of the curves assume a BPSK modulated signal and an Additive White Gaussian Noise (AWGN) channel. The figure shows that for the same input SNR, the channel coding can significantly reduce the output BER compared to an uncoded scheme. As an example, for the case in Fig. 1.2, the BER is reduced from 10^{-3} in the uncoded BPSK scheme to less than 10^{-9} with the coded scheme for

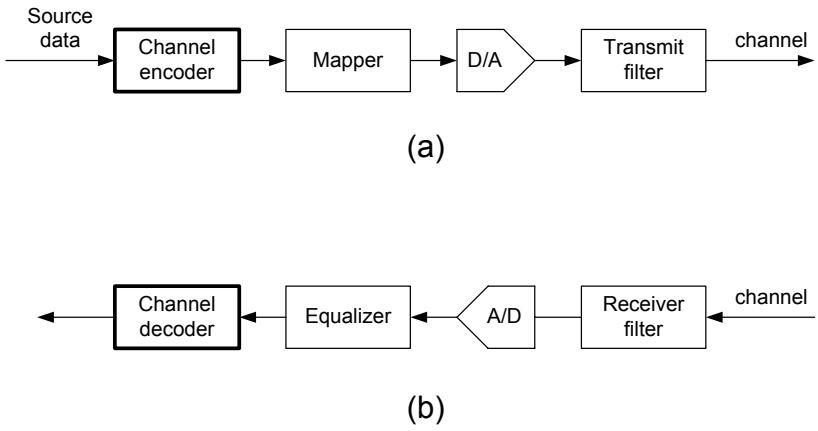


Figure 1.1: An example communication system with ECC coding: (a) the transmitter, (b) the receiver.

an input SNR of about 7 dB. Similarly, the ECC code can reduce the required input SNR to achieve the same output BER. For example, the uncoded coding scheme in Fig. 1.2 requires an SNR of about 12 dB for BER of 10^{-8} whereas the RS code can have the same BER with less than 7 dB of input SNR.

Several families of error control codes have been developed over the past few decades. Reed-Solomon (RS) codes [6] are now being extensively used in Compact Disks, DVD players, hard drives and long-haul optical communications to protect the information bits against the storage and communication errors. Deep space satellite communications and 3G-wireless are also widely using another powerful family of codes called Turbo codes [7].

In the past decade, LDPC codes have been found with superior error correction performance than Turbo codes. Although LDPC codes were originally introduced by Gallager in 1960's [3], they were not further explored until 1990's [8] due to their complex decoding hardware at the time. In [9] an LDPC code of length one million was designed which performed within 0.13 dB of the theoretical Shannon limit at a BER of 10^{-6}. In [10] it was shown that a regular LDPC code with a modest length of 2048 bits

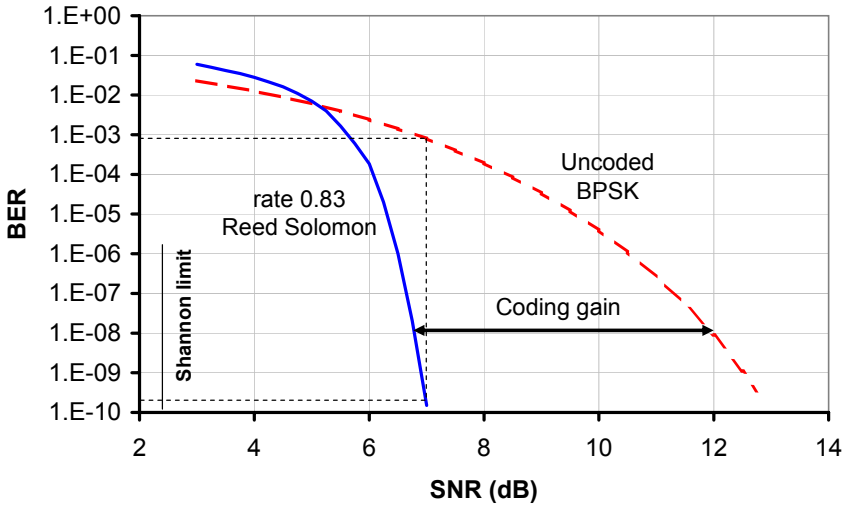


Figure 1.2: The effect of ECC on BER performance.

can also perform within only 1.5 dB of the Shannon limit. In addition to excellent BER performance, the decoding algorithm used to decode LDPC codes is highly parallel, providing the potential for very high decoding throughputs. Furthermore, thanks to Gallager's early publication of LDPC codes, their use is not restricted by patents, unlike Turbo codes [11]. Because of the above three reasons, LDPC codes have recently been adopted for different digital communication standards, including the European Digital Satellite Broadcast standard [12] in 2004 and 10-Gbit Ethernet wireline standard (IEEE802.3an) [13] in 2005. The recent Mobile WiMAX standard (IEEE802.16e) [14] also suggests LDPC codes as an optional error correction scheme.

For LDPC codes, the decoding process is typically more challenging than the encoding process. Whereas the encoding can be done in a non-iterative fashion, the decoding is usually an iterative process comprising computationally-intensive operations on soft-information. One iterative decoding algorithm commonly used for LDPC codes is called message-passing decoding. The algorithm operates on a set of "mes-

sages" each of which represents the decoder's belief about the value of a received bit. In each message-passing iteration, the algorithm refines the messages by considering known constraints imposed by the encoder on the data. For LDPC codes, the message-passing algorithm is highly parallel as there are typically a large number of messages which can be updated independently.

Hardware implementations of LDPC decoders may be categorized into two main groups: partially-parallel and fully-parallel. Although the fully-parallel architecture potentially provides the highest throughput, most of the decoders reported so far have focused on partially-parallel architecture where a number of shared processing units are used for updating the messages in each iteration. This is mainly due to two reasons. First, a fully-parallel decoder occupies a large area due to the large number of required processing units [15]. Second, due to the random structure of LDPC code graphs, the processing nodes in a fully-parallel decoder must communicate over a large and irregular network. This results in a large number of long and random wires across the decoder chip required to convey the messages between nodes. The complex interconnect in turn results in routing congestion and degrades the timing performance. In partially-parallel decoders, the large and irregular network of node-to-node communication translates into large and power-hungry memory and address generation circuitry.

The goal in this work is to develop LDPC decoders with high energy efficiency and multi-Gbps decoding throughput while occupying practical chip area. As described above, a major challenge in implementing high-throughput decoders is the complexity of the node-to-node communications. We will propose a technique that reduces the decoder's node-to-node communication complexity by re-formulating the conventional message passing update rules. We will show that the proposed half-broadcasting technique results in 26% global wirelength reduction in the presented 2048-bit fully-parallel decoder.

To increase the decoding throughput, we will discuss a block-interlacing technique where two independent frames can be decoded simultaneously. We compare the throughput improvement and the hardware overhead associated with this technique. For the two decoders reported in Chapter 4, the post-layout simulations show that the block interlacing improved the throughput 60% and 71% at the cost of only 5.5% and 9.5% more gates, respectively.

We will present an analysis of the energy efficiency of LDPC decoders as a function

of the degree of parallelism in the decoder architecture. The analysis shows that for a fixed throughput, the fully-parallel architecture is more power efficient than partially-parallel decoders. In addition, we show how an early termination scheme can further reduce the power consumption by terminating the decoding process as soon as a valid codeword has converged. To facilitate the early termination, we introduce an efficient method of detecting the decoder convergence with minimal hardware overhead.

To reduce the interconnect complexity and decoder area, we also propose bit-serial message passing in which the multi-bit messages are calculated and conveyed between processing in a bit-serial fashion. We will also propose an approximation to the min-sum decoding algorithm that reduces the area of the CNUs by more than 40% compared with conventional min-sum decoding with only 0.1dB performance penalty at BER=10^{-6}.

Finally, we report on two different fully-parallel decoder implementations: one on an FPGA and one on an ASIC. The fabricated 0.13-μm CMOS bit-serial (660, 484) LDPC decoder has a 300-MHz maximum clock frequency with a nominal 1.2-V supply corresponding to a 3.3-Gbps total throughput. It performs within 3 dB of the Shannon limit at a BER of 10^{-5}. With the power saving achievable by early termination, the decoder consumes 10.4 pJ/bit/iteration at E_b/N_0=4 dB at nominal supply voltage. Coupling early termination with supply voltage scaling results in 648 Mbps total decoding throughput with 2.7 pJ/bit/iteration energy efficiency which even compares favorably with analog decoders [16] aimed for energy efficiency.

1.2 Outline

The outline of this thesis is as follows. Chapter 2 provides background on the prior work on LDPC codes, decoding algorithms and decoder implementations. Chapter 3 describes a technique called broadcasting to reduce interconnect complexity. We will show the benefits of this technique both for partially-parallel and fully-parallel decoders. Chapter 4 discusses a block-interlacing scheme that further increases the decoding throughput. The power analysis and power reduction achievable by early decoding termination is presented in Chapter 5. A bit-serial fully-parallel decoding architecture is proposed in Chapter 6. Finally, Chapter 7 concludes the thesis and provides potential venues for future work.

2 Low-Density Parity-Check (LDPC) Codes

2.1 Code Structure

A binary LDPC code, C, can be described as the null space of a sparse $M \times N$ $\{0,1\}-$valued *parity-check* matrix, H [3]. In other words, C consists of codewords $u = (u_1, u_2, \ldots, u_N)$ such that

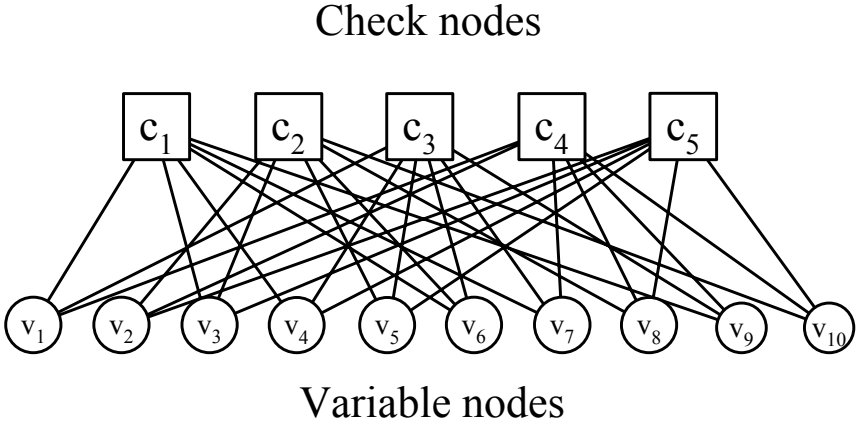
$$uH^T = 0, \tag{2.1}$$

where uH^T is computed in the Galois field $GF(2)$. An LDPC code can also be described by a bipartite graph, or *Tanner graph* [17], as shown in Fig. 2.1. In this figure, variable nodes $\{v_1, v_2, \ldots, v_N\}$ represent the columns of H and check nodes $\{c_1, c_2, \ldots, c_M\}$ represent the rows. An edge connects check node c_m to variable node v_n if and only if H_{mn}, the entry in H at row m and column n, is nonzero. We denote the set of variables that participate in check c_m as $N(m) = \{n : H_{mn} = 1\}$ and the set of checks in which the variable v_n participates as $M(n) = \{m : H_{mn} = 1\}$. A particular variable-node configuration (i.e., an assignment of $\{0,1\}$-values to each of the variable nodes) is a codeword of C if and only if all the checks are satisfied, i.e., if and only if

$$\sum_{n \in N(m)} v_n = 0 \pmod 2,$$

for all $m \in \{1, 2, \ldots, M\}$. An LDPC code is called (d_v, d_c)-regular if the number of ones in all columns of H is d_v and the number of ones in all rows of H is d_c. If the number of ones in all columns or the number of ones in all rows is not equal, the code is referred to as an irregular code. For a code with a full-rank $M \times N$ parity-check matrix, H, the code rate is $R = (N - M)/N = 1 - M/N$. Fig. 2.1 shows the Tanner graph for a trivial (3,6)-regular LDPC code with $N = 10$ and $M = 5$. It should be noted that in more realistic LDPC codes the code length N is much longer (typically

greater than 100) and the number of ones in the parity check matrix is much lower than the number of zeros.



$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Figure 2.1: Tanner graph for a (3,6)-regular LDPC code and the corresponding parity check matrix.

LDPC codes are encoded using a *generator* matrix, $G_{K\times N}$, where K is the number of information bits per codeword. For a full-rank M by N parity check matrix we have $K = N - M$. The encoder generates the codeword $u = (u_1, u_2, \ldots, u_N)$ from the input information vector $v = (v_1, v_2, \ldots, v_K)$ based on

$$u = vG. \tag{2.2}$$

Since all the valid codewords should satisfy the equality in (2.1), we have

$$vGH^T = 0, \tag{2.3}$$

and since (2.3) is true for every valid v, we have

$$GH^T = 0. \tag{2.4}$$

It can be shown that any full-rank H matrix can be rearranged by Gaussian elimination and row/column permutations into the form $H = [P|I_{N-K}]$, where P is an $N-K$ by K matrix and I_{N-K} is an $N-K$ by $N-K$ identity matrix. From (2.4) it can be shown that the generator matrix G can be constructed as

$$G = [I_K|P^T]. \tag{2.5}$$

A generator matrix with the form as in (2.5) is called a *systematic* generator matrix. It has the property that the encoded codeword consists of the information bits concatenated with $N-K$ parity check bits. A systematic generator matrix simplifies the encoding process as only the parity bits need to be calculated to generate encoded words from blocks of information bits.

2.2 LDPC Decoding Algorithms

LDPC codes are decoded with a general family of decoding algorithms called iterative message-passing decoding algorithms. The sum-product algorithm (SPA) and min-sum (MS) algorithm are the two most commonly-used message-passing decoding algorithms and can be described using the Tanner graph representing the LDPC code. In these algorithms the decoding process starts by observing the channel inputs (*intrinsic* messages) corresponding to the variable nodes in the current received frame. Then each decoding iteration consists of updating and transferring *extrinsic* messages between neighboring variable and check nodes in the graph as shown in Fig. 2.2. A message encodes a *belief* about the value of a corresponding received bit and is usually expressed in the form of a log-likelihood ratio (LLR). Through message-passing these beliefs are refined and, in the case of a successful decoding operation, eventually converge on the originally transmitted codeword. The details of how messages are updated in SPA and MS decoding are presented in the next two sub-sections.

2.2.1 Sum-product algorithm

Suppose that a binary codeword $W = (w_1, w_2, \ldots, w_N) \in C$ is transmitted over a communication channel and that a vector $Y = (y_1, y_2, \ldots, y_N)$ of bit signals is

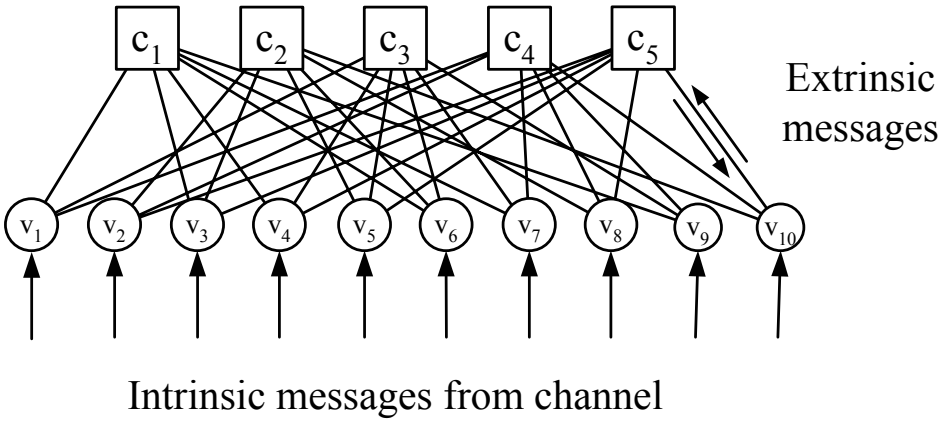


Figure 2.2: Exchange of information in a message-passing decoding algorithm.

received. Let $z_{mn}^{(i)}$ and $q_{mn}^{(i)}$ represent the messages sent from v_n to c_m and from c_m to v_n in the ith iteration, respectively. Let $N(m) = \{n : H_{mn} = 1\}$ and $M(n) = \{m : H_{mn} = 1\}$. Let I_M denote the maximum number of iterations. SPA decoding [18] consists of the following steps.

1. Initialize the iteration counter, i, to 1.

2. Initialize $z_{mn}^{(0)}$ to the *a posteriori* log-likelihood ratios (LLR), $\lambda_n = \log\big(P(v_n = 0|y_n)/P(v_n = 1|y_n)\big)$ for $1 \le n \le N$, $m \in M(n)$.

3. Update the check nodes, i.e., for $1 \le m \le M$, $n \in N(m)$, compute:

$$q_{mn}^{(i)} = 2\tanh^{-1}\left(\prod_{n' \in N(m)\backslash n} \tanh(z_{mn'}^{(i-1)}/2)\right). \tag{2.6}$$

4. Update the variable nodes, i.e., for $1 \le n \le N$, $m \in M(n)$, compute:

$$z_{mn}^{(i)} = \lambda_n + \sum_{m' \in M(n)\backslash m} q_{m'n}^{(i)}. \tag{2.7}$$

5. Make a hard decision, i.e., compute $\hat{W} = (\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_N)$, where element $\hat{w}_n$

is calculated as

$$\hat{w}_n = \begin{cases} 0 & \text{if } \lambda_n + \sum_{m' \in M(n)} q^{(i)}_{m'n} \geq 0; \\ 1 & \text{otherwise.} \end{cases}$$

for $1 \leq n \leq N$. If $\hat{W}H^T = 0$ or $i \geq I_M$ output $\hat{W}$ as the decoder output and halt; otherwise, set $i = i + 1$ and go to step 3.

The sign (+ or -) of an LLR message indicates the belief about the value (0 or 1, respectively) of the received bit on the corresponding variable node. The magnitude of the message indicates the reliability of that belief. The variable update function in (2.7) combines all the beliefs about the value of the bit by adding the incoming messages. The check update function in (2.6) calculates the LLR for each outgoing message based on the fact that the parity check on all the edges connected to each check node has to be satisfied. This can be noted from the fact that for an LLR message λ, defined as $\lambda = \log\big(P(0)/P(1)\big)$, one can show that $\tanh(\lambda/2) = P(0) - P(1)$.

2.2.2 Min-sum algorithm

MS decoding [19] can be considered an approximation to SPA decoding [18]. The difference is that in MS, the check node update function of (2.6) is replaced with

$$\epsilon^{(i)}_{mn} = \min_{n' \in N(m) \backslash n} |z^{(i)}_{mn'}| \prod_{n' \in N(m) \backslash n} \mathrm{sgn}(\mathrm{z}^{(\mathrm{i})}_{\mathrm{mn}'}). \tag{2.8}$$

Although the performance of MS is generally a few tenths of a dB lower than that of SPA decoding, it requires much simpler computational resources for the check node functions. Moreover, it is more robust against quantization errors when implemented with fixed-point operations [20].

To reduce the performance gap between MS and SPA algorithms, in [21] a modified check node update is proposed for the case of degree-3 check nodes by applying a correction factor to the check node update function in (2.8). In [22] a degree-matched modification is proposed for any check degree by applying a correction factor which is a function of the check node degree. It is shown that the modified min-sum algorithm provides almost the same BER performance as the SPA.

2.3 Capacity-Approaching LDPC Codes

In spite of recent progress in the information theory community, there are still no general methods to predict the performance of LDPC codes or to design LDPC codes with excellent performance that do not require extensive simulations. Some techniques, such as density evolution [23] and EXIT charts [24], exist to analyze and predict the convergence behavior of LDPC codes, but they usually are applied to families (or ensembles) of codes with infinite code length and certain check and variable degree distribution. Density evolution can be used to predict the decoding threshold value (i.e., the maximum noise level that can be added to the transmit signals while the decoder BER can be kept arbitrarily small). The density evolution and EXIT charts have also been used to optimize the degree distribution for constructing high performance irregular LDPC codes [25],[26].

Although the above techniques have enabled researchers to design long codes ($N > 10^6$) that perform less than a tenth of a dB from the Shannon limit, their accuracy tends to decrease for more practical codes with much shorter length. This is because the analysis in density evolution and EXIT chart techniques are only valid when the incoming messages arriving at each node are independent, which only happens when the code graph has no cycles. While this might be a reasonable approximation for long codes, the code graph of short LDPC codes inevitably contain some short cycles which make the above approximation inaccurate.

The length of the shortest cycle in the code graph is called the *girth* of the graph. Among other affecting parameters such as minimum weight and the number of short cycles in the code graph, it is known that in general LDPC codes with higher girth have better BER performance. Several algorithms have been proposed to construct high-girth LDPC codes [27],[28],[29]. As an example, in the progressive edge growth algorithm in [28] the code graph is constructed by incrementally adding the edges to the graph such that a high girth is maintained.

Also, families of LDPC codes have been constructed using finite geometries [30] and algebraic methods [10]. The method proposed in [10] is based on Reed-Solomon codes with two information bits. It is shown that the generated LDPC codes have no cycles of length 4 and have high minimum distance. Due to their superior performance and relatively short code length, the (6, 32)-regular (2048, 1723) RS-based LDPC code generated in [10] is now adopted as the standard code for the IEEE802.3an 10GBase-T standard [31].

2.4 Decoder Implementation

Depending on the particular application, the objective when designing LDPC decoders is to meet a set of design specifications such as decoding throughput, power consumption, silicon area, decoding latency and testability. In the cases where the throughput and latency are explicitly mentioned in standards, the design goal is to achieve these specs while optimizing other parameters such as power and area.

As mentioned above, message-passing decoding requires a large number of messages to be updated and transferred on the code graph in each iteration. Previous researchers have proposed several approaches for representing and updating these messages.

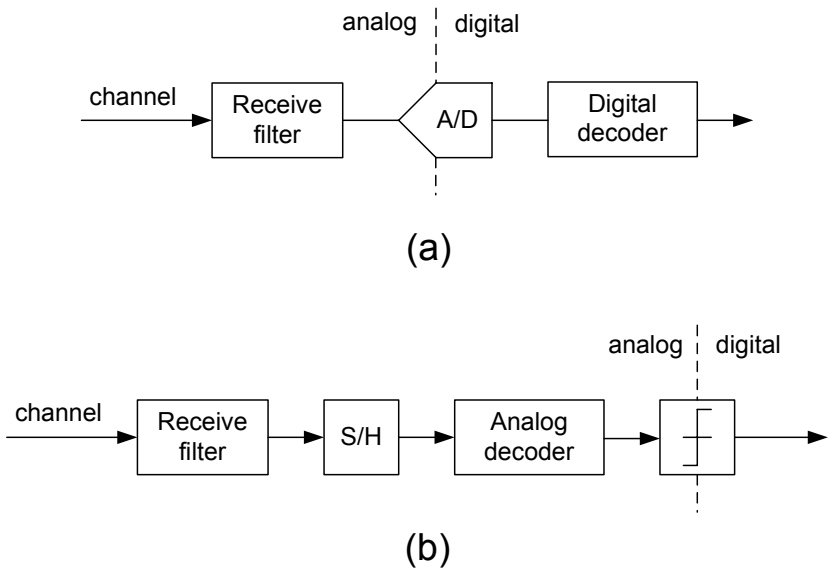


Figure 2.3: (a) Conventional receiver with digital ECC decoder, (b) An alternative receiver with analog decoder.

In [32], [33] and [16], analog signals are used to represent the extrinsic messages. Fig. 2.3 shows the difference between a conventional receiver architecture with digital decoder and an alternative architecture with an analog decoder. In analog decoders, the exponential voltage-current relationship of a bipolar device or a sub-threshold CMOS device is used to realize the required message-passing update functions. Al-

though analog decoders have the advantage of relatively low power consumption, they are faced with the following challenges. First, due to process mis-matches and various sources of noise, analog decoders do not scale well with code length. This in turn limits the designer to adopt short LDPC codes (typically less than 500 bits) which usually have inferior BER performance. The limited scalability in analog decoders also limits the decoder throughput typically to less than 50 Mbps, which is far below the throughput requirement for current high-speed applications such as 10GBase-T or Digital Video Broadcast. Another major challenge in analog LDPC decoding is the need to store a relatively large number of analog signals at the beginning and also during the decoding process. Other issues include the need for framing the received symbols, the need for a customized design flow and finally the lack of testability.

In [34] an LDPC decoder with stochastic computation is proposed in which the messages are represented using Bernoulli sequences. Although this representation results in a very simple check and variable node architecture, it needs a significant amount of hardware overhead in order to interface the stochastic messages at the decoder inputs and outputs. In addition, since the stochastic computation uses a redundant number representation, it requires a large number of clock cycles to decode each frame (the decoder in [34] on average requires several hundred clock cycles at low SNRs and about 100 cycles at high SNRs) which limits the decoding throughput.

More conventional LDPC decoders often use a synchronous digital circuit with multi-bit digital signals to represent the messages. In these decoders, the decoding throughput, η, in $bits/s$ is calculated as

$$\eta = \frac{Nf}{IL}, \tag{2.9}$$

where N is the the code block length, I is the number of decoding iterations performed per block, L is the number of clock cycles required per iteration and f is the operating clock frequency. Parameters N and I are usually determined in the code design phase, so to increase the decoder throughput one must increase f/L. The maximum clock frequency of a synchronous digital circuit is determined by the propagation delay in the critical path (i.e., the longest path between sequential storage elements , e.g., flip-flops or latches). The value L (and also to some extent f) is a direct function of the decoder architecture.

LDPC decoders can be broadly categorized into partially-parallel (also known as memory-based) decoders and fully-parallel decoders as described next. A generic

partially-parallel LDPC decoder architecture is shown in Fig. 2.4. It consists of shared variable node update units (VNUs), shared check node update units (CNUs), and a shared memory fabric used to communicate messages between the VNUs and CNUs. Inputs to each CNU are the outputs of VNUs fetched from memory. After performing some computation (e.g., MIN operation for the magnitude and parity calculation for the signs in min-sum decoding), the CNU's outputs are written back into the extrinsic memory. Similarly, inputs to each VNU arrive from the channel and several CNUs via memory. After performing the message update (e.g., SUM operation in min-sum decoding), the VNU's outputs are written back into the extrinsic memory for use by the CNUs in the next decoding iteration. Decoding proceeds with all CNUs and VNUs alternately performing their computations for a fixed number of iterations, after which the decoded bits are obtained from one final computation performed by the VNUs.

A common challenge in partially-parallel architectures is to manage the large number of memory accesses and prevent memory collisions since multiple messages must be accessed simultaneously by the check and variable nodes. Examples of partially-parallel decoders include [35],[36] and [37]. The "hardware-aware code design" methodology in [35] provides 640 Mbps throughput (with 10 iterations per frame) for an LDPC code of block length 2048 with a tunable code rate; the design occupies 14.3 mm^2 in a 0.18-μm CMOS technology. The partially-parallel DVB-S2 compliant LDPC decoders reported in [36] are programmable for 16200-bit or 64800-bit modes and for a wide range of code rates. They achieve a maximum throughput of 90 Mbps and 135 Mbps in 130 nm and 90 nm technologies and occupy 49.6 mm^2 and 15.8 mm^2, respectively. A 3.33 Gbps hardware-sharing (1200, 720) LDPC decoder is reported in [37].

In fully-parallel decoders, each node in the code's Tanner graph is assigned a dedicated hardware processing unit, and messages are communicated between nodes by wires. Fig. 2.5 shows the high-level architecture of a fully-parallel decoder. It can be seen that the extrinsic memory block of Fig. 2.4 is replaced with the interconnections. This is because in a fully-parallel architecture, each extrinsic message is only written by one VNU or CNU, so the extrinsic memory can now be distributed amongst the VNUs and CNUs and no address generation is needed. The drawbacks of a fully-parallel architecture are the large circuit area required to accommodate all the processing units, as well as a complex and congested global interconnect net-

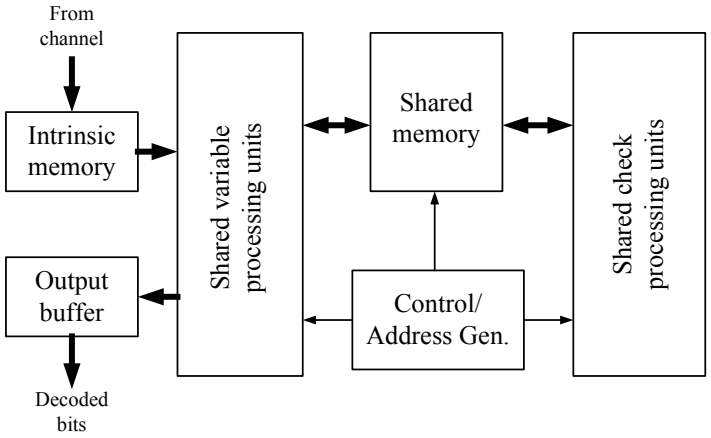


Figure 2.4: A partially-parallel LDPC decoder.

work. Routing congestion leads to longer interconnect delays and can degrade decoder timing-performance and dynamic power dissipation. Examples of fully-parallel decoders include [15] [38] [39].

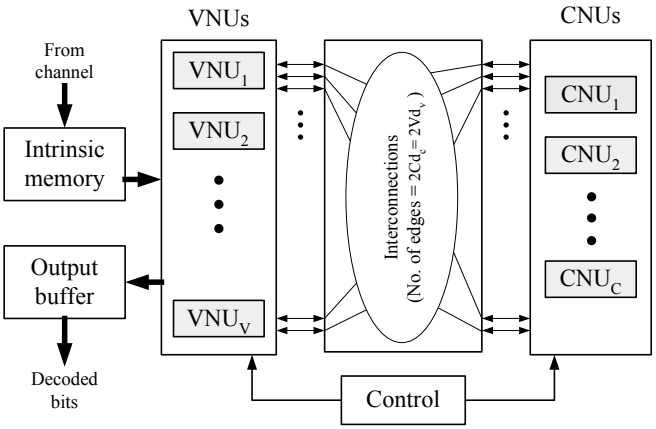


Figure 2.5: The fully-parallel iterative LDPC decoder architecture.

Both fully-parallel and partially-parallel decoders need to store the extrinsic messages so that they can be accessed in the next decoding iteration. The size of memory should be at least equal to the number of edges in the code Tanner graph. Usually in pipelined designs the memory size is a constant multiple of the number of edges.

In partially-parallel decoders the messages are stored in large memory blocks and are accessed through read/write operations. However, in fully-parallel decoders the storage is more distributed among processing units typically in the form of flip-flops. Although in comparison with an SRAM memory cell individual flip flops occupy larger area and consume more power, they have their advantages: they do not require address generation and in addition they can be accessed simultaneously by their corresponding processing units.

2.5 Architecture-Aware LDPC Codes

It is known that although randomly-generated LDPC codes with no regular structure in the parity check matrix tend to have high performance, they are not usually suitable for VLSI implementation. The reason is that randomly-structured codes require complex memory module addressing schemes that tend to decrease the decoding throughput and increase the area and power consumption.

To overcome these issues, researchers have proposed a decoder-first code design [40] methodology in which the LDPC code is designed to fit a pre-determined decoding architecture. The goal has been to minimize the degradation due to the structure in the code graph. In [41] a joint code-decoder design methodology is proposed. Using this methodology, Zhang and Parhi designed a (3, k)-regular LDPC code with $L \cdot k^2$ variable nodes and $3L \cdot k$ check nodes. It is suitable for its proposed partially-parallel decoder architecture with k^2 variable processing units and k check processing units.

In [42] a Turbo-Decoding Message-passing (TDMP) decoding architecture is proposed along with an architecture-aware code construction that is shown to improve the convergence speed of the decoding process and reduces memory requirements.

Quasi-cyclic (QC) LDPC codes have the advantage of efficient encoding architecture using feedback shift registers. In [43] two classes of QC-LDPC codes are generated that perform close to computer-generated random codes. In [44] two methods are proposed to construct a systematic cyclic generator matrix from a QC parity-check matrix.

A convolutional version of LDPC codes has been proposed in [45]. At any given time, each code bit in an LDPC convolutional code (LDPC-CC) is generated using only previous information bits and previously generated code bits. It is shown that for certain applications, such as streaming video and variable length packet switch-

ing networks, LDPC-CCs are preferred over the conventional block LDPC codes [46]. LDPC-CCs also have the advantage of simple encoder structure. An ASIC implementation of a 175-Mbps, rate-1/2 (128, 3, 6) LDPC-CC encoder and decoder is reported in [47].

2.6 Overview

The motivation for this work is to design LDPC decoders with high throughput for applications such as fiber-optic communication systems and 10-Gbit Ethernet. This leads naturally to the investigation of fully-parallel architectures as these exploit all of the available parallelism in the message-passing decoding algorithm. Our power consumption analysis shows that fully-parallel architectures may also have advantages in energy efficiency.

A common challenge for fully-parallel LDPC decoders is the routing congestion between variable and check nodes which is the result of the randomness in the code parity check matrix. In this work, we present a technique called broadcasting and illustrate how it can reduce the routing congestion.

An overlapped message-passing architecture is then presented to increase the decoding throughput with a relatively small hardware overhead. We show that for the two presented fully-parallel decoders, the throughputs are improved by more than 60% at the cost of less than 10% logic overhead.

The power consumption is always a very important criteria in LDPC decoders. We will present an analysis of how the increased parallelism in the LDPC decoder architecture results in reduced total power consumption for a given target throughput. In addition we will illustrate an efficient early termination technique to further reduce the decoder dynamic power consumption.

We demonstrate implementations of a bit-serial message-passing decoder that reduce the routing congestion and also result in lower decoder core area. We also propose an approximation to the min-sum decoding algorithm which simplifies the check node logic with minimal BER penalty.

Although most of the proposed techniques in this work are mainly developed for fully-parallel decoders, we will explain how some of them can also be applied to partially-parallel decoders.

3 Reducing Interconnect Complexity

As mentioned in Chapter 2, a common challenge when implementing LDPC decoders is communicating the extrinsic messages through the complex interconnections between the variable and check nodes. In partially-parallel decoders, the interconnection network results in a complicated memory access scheme where multiple processing units need to access a large number of shared memory blocks in parallel. In fully-parallel decoders, the complex interconnection results in a complicated routing network all across the chip in order to transfer the messages between the parallel check and variable processing units.

In this chapter we describe a technique called broadcasting that reduces the node-to-node communication complexity in LDPC decoding. We will show how this technique can be applied to both fully-parallel and partially-parallel decoder architectures. We will also discuss the trade-offs that two variants of broadcasting (half-broadcasting and full-broadcasting) provide in terms of logic overhead and reduced interconnect complexity.

3.1 Half-broadcasting

We start by re-writing (2.6) and (2.7) of the sum-product algorithm as follows. Let

$$q_{mn}^{(i)} = 2\tanh^{-1}\left(P_m^{(i)} / \tanh(z_{mn}^{(i-1)}/2)\right), \tag{3.1}$$

where

$$P_m^{(i)} = \prod_{n' \in N(m)} \tanh(z_{mn'}^{(i-1)}/2) \tag{3.2}$$

and let

$$z_{mn}^{(i)} = S_n^{(i)} - q_{mn}^{(i)}, \tag{3.3}$$

where

$$S_n^{(i)} = \lambda_n + \sum_{m' \in M(n)} q_{m'n}^{(i)}. \tag{3.4}$$

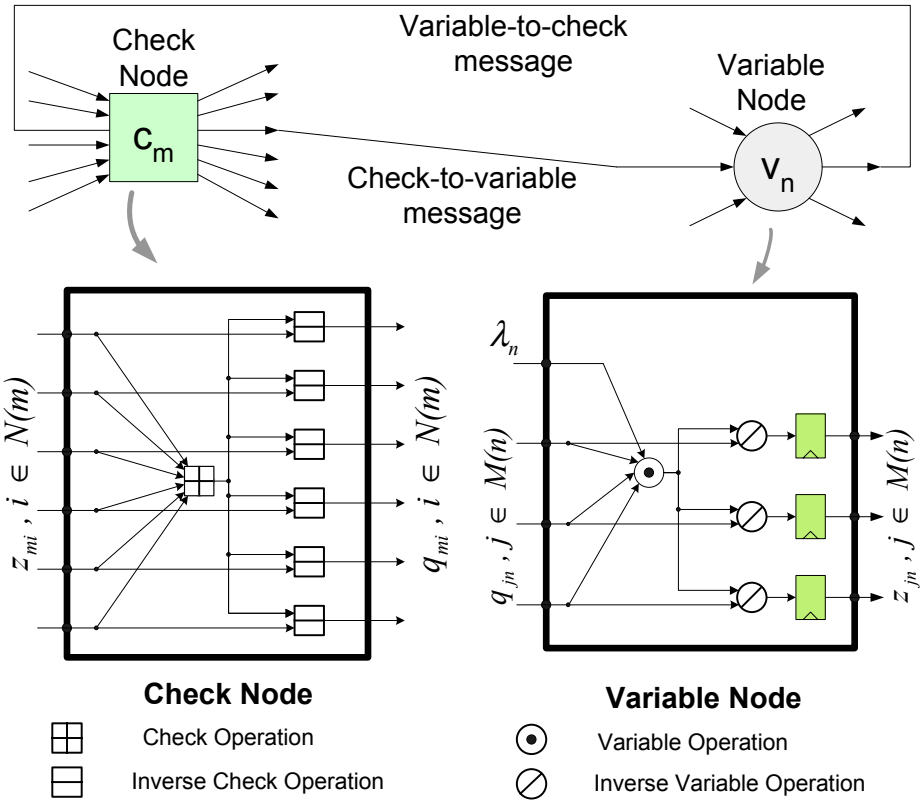


Figure 3.1: A conventional fully-parallel message-passing LDPC decoder with generic functions for check and variable nodes.

Fig. 3.1 shows the block diagram of a check and variable processing unit using (3.1)-(3.4). Symbols $\boxminus$ and $\boxplus$ denote the operations that are performed in (3.1) and (3.2), respectively. Similarly, symbols $\oslash$ and $\odot$ denote operations performed in (3.3) and (3.4), respectively.

Half-broadcasting is a repartitioning of the computations in Fig. 3.1. In this new partitioning, shown in Fig. 3.2, the $\boxminus$ functions are moved to the variable nodes without affecting the message-passing algorithm. This is because extrinsic messages, $q_{mn}^{(i)}$, are reconstructed in the variable nodes from the received $P_m^{(i)}$ and the $z_{mn'}^{(i-1)}$'s from iteration $i-1$. So, unlike the schemes in [48, 15, 2] in which each degree-d_c check node generates d_c separate messages, one for each neighboring variable node, in this scheme each check node *broadcasts* a single message (i.e., $P_m^{(i)}$) to all of its neighbors. This approach reduces the amount of information that needs to be conveyed from check nodes to variable nodes. In a fully-parallel decoder, this translates into a reduction in global interconnect. In a partially-parallel decoder, it translates into fewer memory accesses.

Although the broadcasting technique above was described using the sum-product algorithm, the same technique can be applied to other variations of message-passing decoding such as min-sum decoding and bit-flipping [3]. For example, in the case of min-sum decoding, the variable nodes would be designed to broadcast the total value, S_n, to their neighbors.

3.1.1 Half-broadcasting for fully-parallel decoders

Fig. 3.3 shows conceptually how the broadcasting technique can mitigate the interconnection problem in fully-parallel decoders. In this figure, a floorplan similar to [15] is used, where the check nodes are placed in the center of the chip layout and variable nodes are placed on the sides. In Fig. 3.3(a), a node architecture with no broadcasting is assumed, where one sample check node in the center of the chip sends different messages to its neighboring variable nodes. This is done by dedicating separate wires (or buses) for each destination. However, using broadcasting, we share a significant amount of interconnect wiring when conveying messages from each source check node to the destination variable nodes, as in Fig. 3.3(b).

Fig. 3.4 shows a zoomed-in portion of the check-to-variable interconnects for a 2048-bit LDPC code before and after applying a half-broadcast scheme. This figure is generated by Matlab simulation and assumes that wires can be in any arbitrary

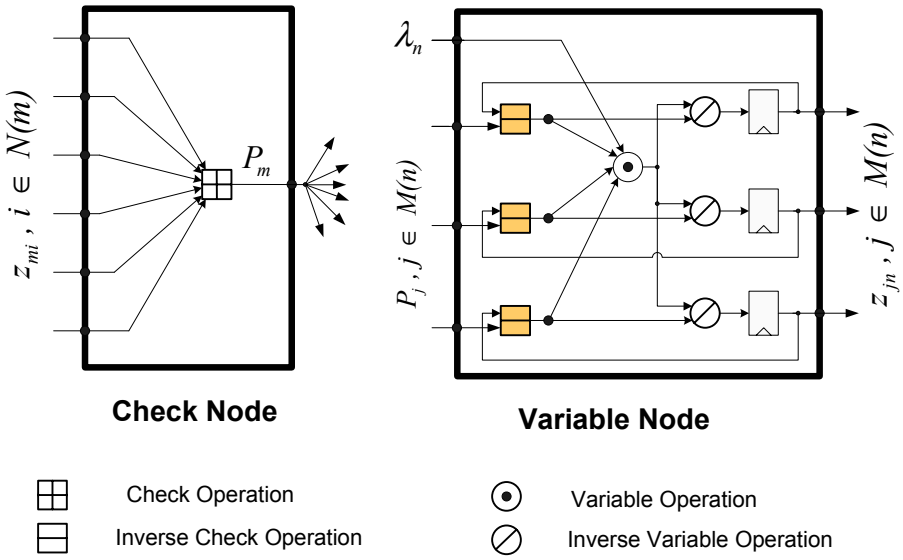


Figure 3.2: A half-broadcast architecture. The check node c_m broadcasts a single message, P_m, to all neighboring variable nodes.

direction. However, we can observe similar congestion effect in the layouts where only vertical and horizontal wiring is used.

Fig. 3.5 shows the effect of the broadcasting technique on a fully-parallel decoder layout. These are real layouts obtained by automated place-and-route tools from Cadence using a floorplan similar to [15] where the check nodes are instantiated in the center and the variable nodes are instantiated on the sides of the decoder layout. One check node and its neighboring 11 variable nodes and the nets for conveying check-to-variable intrinsic messages between them are highlighted in the figure for clarity. Fig. 3.5(a) shows the case where no broadcasting is applied. Fig. 3.5(b) shows that by using the broadcasting scheme of Fig. 3.2, a significant amount of interconnect wiring can be shared, hence mitigating the complexity of interconnections.

To compare the effects of half-broadcasting on reducing the global wirelength, we implemented a fully-parallel 1-bit quantized LDPC decoder for the (6, 32)-regular (2048, 1723) code adopted for the 10GBase-T Standard twice: first without half-

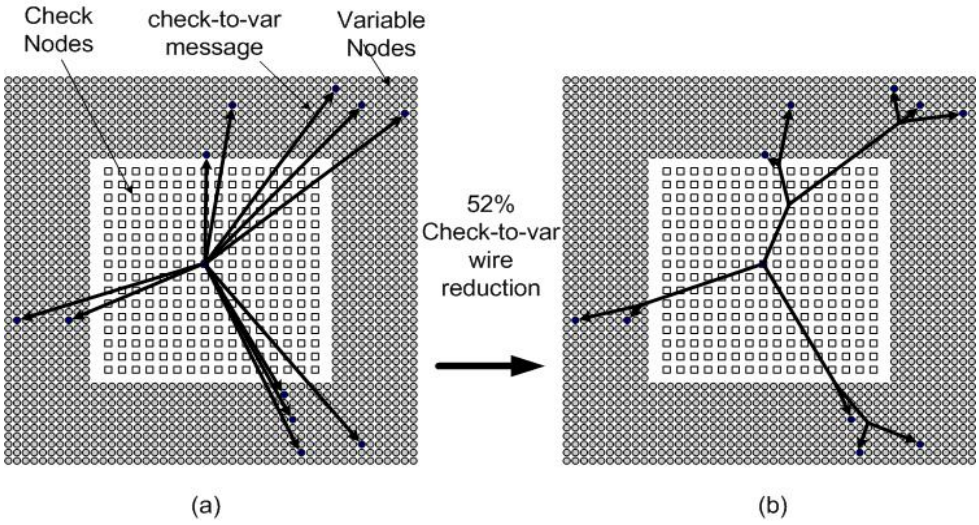


Figure 3.3: Broadcasting reduces the total top-level wirelength by sharing the wires. (a) Output messages of a check node without broadcasting (b) Sharing interconnect wires of a check node with broadcasting

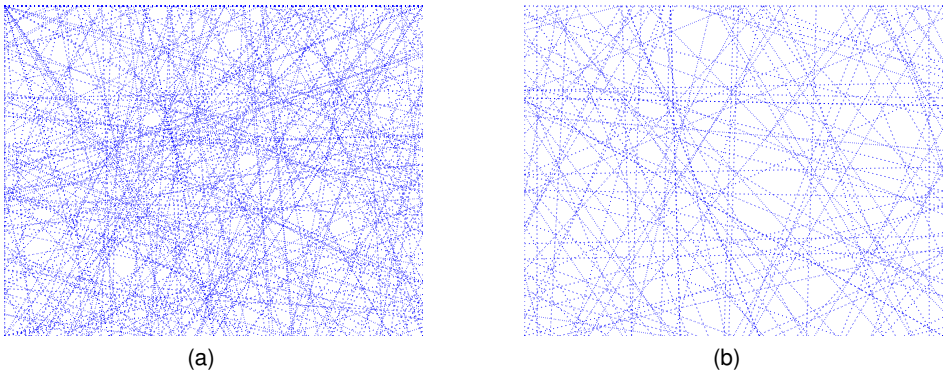
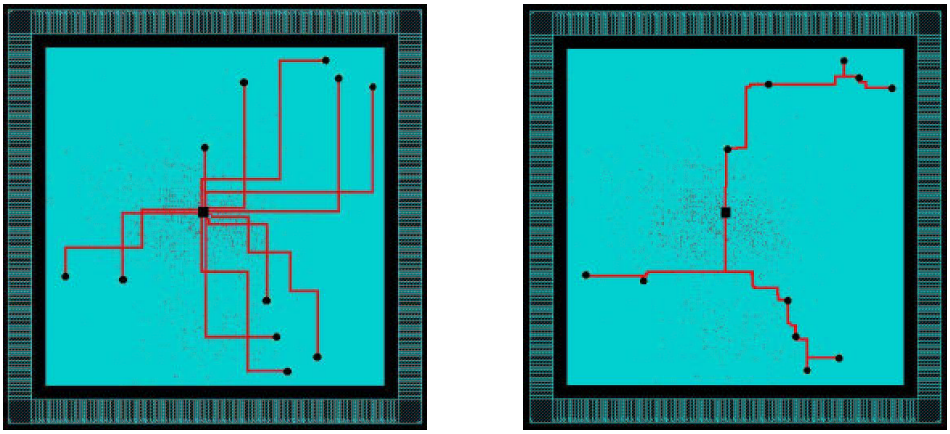


Figure 3.4: A small section of interconnects for a length-2048 LDPC code (a) before broadcast (b) after broadcast. There is 40% reduction in total wirelength.

broadcasting and once with broadcasting. In both cases the decoding algorithm was based on the Gallager's Algorithm A as described in [23].

Fig. 3.6 shows the check and variable node architecture for the non-broadcasting decoder. In Algorithm A, all the check-to-variable and variable-to-check messages are 1-bit values, or *votes*, regarding the value of the corresponding received bit. In the



(a) (b)

Figure 3.5: The routed nets for one check node output highlighted in a fully-parallel LDPC decoder layout: (a) without broadcasting and (b) with broadcasting.

check nodes, each output message is the exclusive-OR of all the other 31 check node inputs. In the variable nodes, the adders and subtracters first calculate the sum of the votes on each message. Then the unanimous voting (UV) blocks calculate the variable node outputs from the channel input, c, and the number of votes for one, s. The UV block output, u is calculated as:

$$u = \begin{cases} 1 & \text{if } s = 5 \\ c & \text{if } 5 > s > 0 \\ 0 & \text{if } s = 0 \end{cases} .$$

In other words, the variable node output on each edge is always the same as the channel input except when all the other incoming messages vote against it. The majority logic block (ML) calculates the hard decision decoder output w from the channel input, c, and the total votes, t, based on:

$$w = \begin{cases} 1 & \text{if } t > 3 \\ c & \text{if } t = 3 \\ 0 & \text{if } t < 3 \end{cases} .$$

The `new-frame` signal specifies the end of one set of iterations and start of loading a new frame.

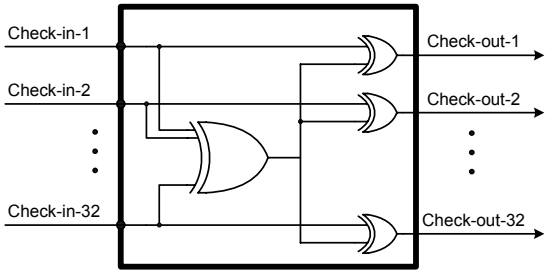
Table 3.1: The average wirelength reduction for global nets in fully-parallel LDPC decoders.

Code	P&R tool	Predicted
(992,829) RS-LDPC	-	27%
(2048,1723) RS-LDPC	26%	26%
(4096,3403) RS-LDPC	-	27%

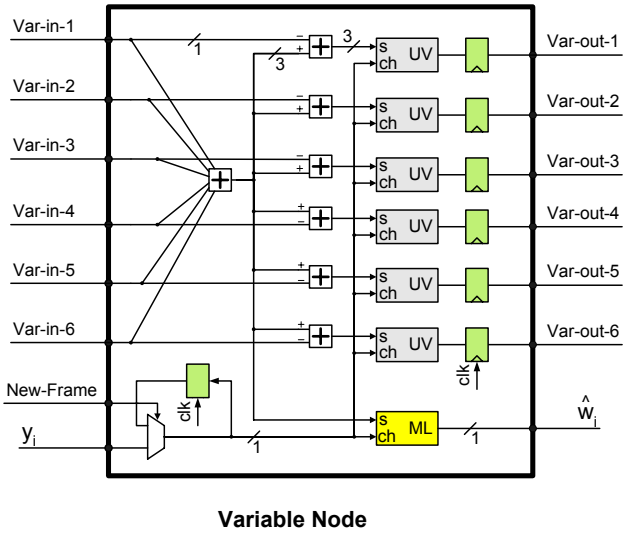
Fig. 3.7 shows the variable node and check node for the decoder with half-broadcast scheme. In this decoder the result of the 32-bit exclusive-OR is broadcast from each check node to all of its neighboring variable nodes. The correct message is then reconstructed in the variable nodes using the variable-to-check messages from previous iteration without affecting the functionality of the decoder.

As explained in this section, half broadcasting results in reduced global wirelength in fully-parallel decoders. For the two 2048-bit decoder designs presented in this chapter, the broadcasting scheme reduced the average node-to-node wirelength by 26% from 1.88 mm to 1.40 mm. The timing and BER performance of the implemented decoder will be presented in more detail in Chapter 4.

Table 3.1 lists the percentage wirelength reduction obtained from half-broadcasting compared with the conventional case where no broadcasting is applied. The technique is applied to LDPC codes with various lengths. The values in the first column are obtained from automated P&R tools. The values in the second column are obtained from a prediction algorithm which approximates a Steiner tree [49] solution to predict the wirelength savings by using the actual floorplan of the fully-parallel decoder and the silicon area dedicated to each variable and check-update unit in the floorplan. Table 3.1 shows that for code lengths of a few thousand bits assuming half-broadcasting yields similar savings in global wirelength.

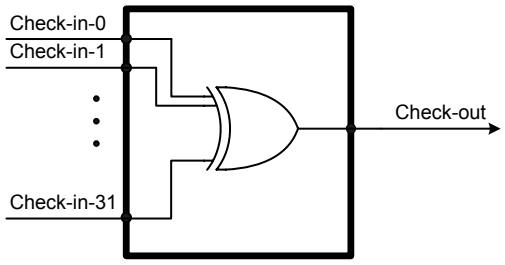


(a)



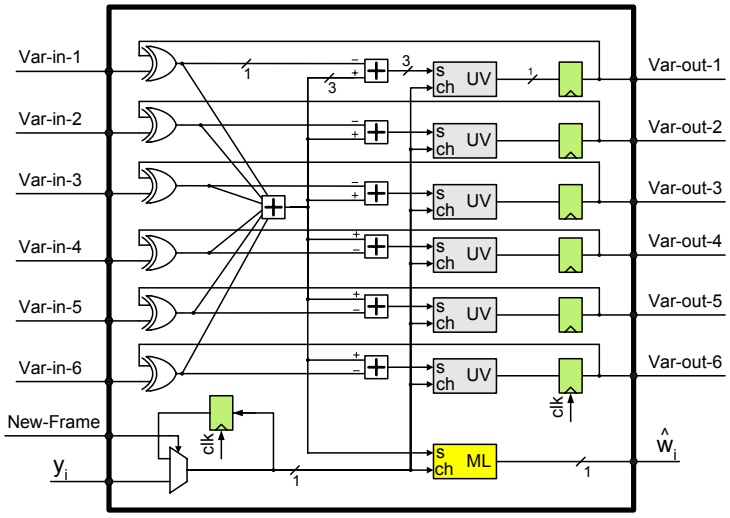
(b)

Figure 3.6: (a) The CNU and (b) the VNU architectures for a conventional hard decision message-passing decoder with no broadcasting.



Check Node

(a)



Variable Node

(b)

Figure 3.7: (a) The CNU and (b) the VNU architectures for a hard decision message-passing decoder with half broadcasting.

3.1.2 Half-broadcasting for partially-parallel decoders

For partially-parallel LDPC decoders, broadcasting reduces the number of shared memory write accesses. This is because in a conventional partially-parallel decoder, each check-processing unit (CPU) reads the extrinsic messages, z_{ij}, generated in the previous iteration from the memory and writes the resulting q_{ij} messages to another shared memory to be read by variable-processing units (VPU's), and so on. Thus the CPU for a check node with degree d_c, needs to perform d_c reads and d_c writes. By using a broadcast scheme for the check nodes, the CPU still needs to read d_c input values, but since the CPU generates only one output value, just one write operation is required. In total, the number of read/write memory accesses per CPU per iteration is reduced from $2d_c$ to $d_c + 1$. Unlike fully-parallel decoders, there is some hardware overhead for half-broadcasting in partially-parallel decoders since the z_{mn}'s also need to be stored locally in the VPU for use in the next iteration. This additional local storage, however, does not add to the global node-to-node communication complexity.

3.2 Full-broadcasting

We called the architecture of Fig. 3.2 half-broadcasting because we applied the broadcasting technique only to the check-to-variable messages while the variable-to-check messages were kept unchanged. The same idea can be extended to a *full-broadcasting* scheme in which both check-to-variable and variable-to-check messages are broadcast. Fig. 3.8 shows the variable and check node architecture capable of full-broadcasting. In this figure, the inverse-check and inverse-variable operations, $\boxminus$ and $\oslash$, are duplicated in order to be able to reconstruct the individual messages from the interim variable and check totals. A memory-based version of full-broadcasting is proposed in [50]. As one can expect, full-broadcasting results in further simplification in interconnect complexity; however this comes with a relatively large logic overhead. The exact amount of this overhead depends on the exact type of variable and check update functions, but since most of the calculations are duplicated, the overhead can be as much as 2x [50]. In addition, it should be noted that the full-broadcasting technique is not applicable for decoding algorithms, such as min-sum decoding, where either the inverse-check function, $\boxminus$, or inverse variable function, $\oslash$, is not available.

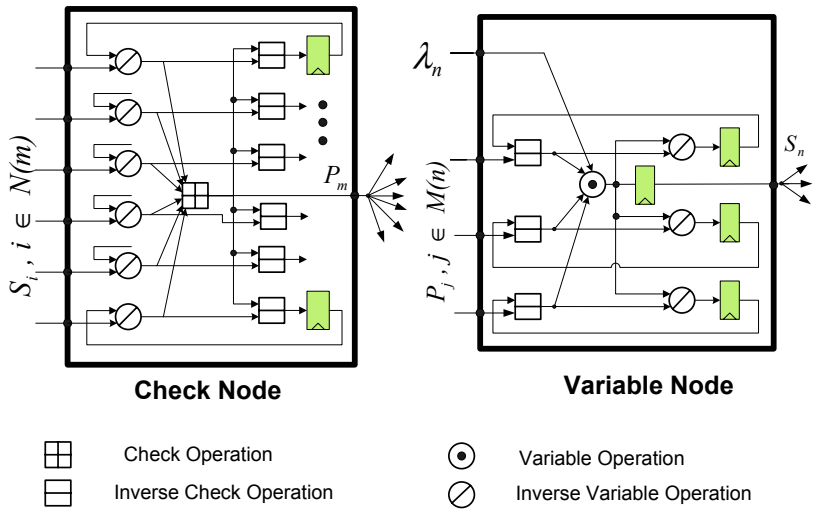


Figure 3.8: A full-broadcast architecture. The check node c_m broadcasts P_m to the neighboring variable nodes. The variable node v_n broadcasts S_n.

3.3 Comparison and Discussion

Depending on the type of update functions, the designer may need to assign a larger word length for the broadcast values (e.g., $P_m^{(i)}$ in (3.2) and $S_n^{(i)}$ in (3.4) in the case of Sum-Product message-passing) compared with the word length needed for the actual extrinsic values (e.g., $q_{mn}^{(i)}$'s and $z_{mn}^{(i)}$'s in (3.1) and (3.3)). In these cases, the effect of increased word length for broadcast messages must be taken into account. As an example, if λ_n and $q_{m'n}$'s in (3.4) are quantized with q bits, then $q + \lceil \log(d_v + 1) \rceil$ bits would be required to represent S_n. However, since the word length of the z_{mn}'s in (3.3) is generally limited to only q bits by clipping, as in [20], S_n can be represented with only $q + 1$ bits without loss of accuracy. So, the variable-to-check messages will have $0.5(1/q) \times 100\%$ additional wiring [1] due to the increased word length in S_n. For q=6, the overhead becomes 8%. A similar analysis can also be made for the check-to-variable broadcasting of Fig. 3.2 since the multiplications and divisions in (3.1) and (3.2) are usually transformed into summations and subtractions in the logarithm

[1] The 0.5 coefficient arises because in this case half-broadcasting only affects the variable-to-check messages and keeps the check-to-variable messages unchanged.

domain. One particular case is the hard-decision message passing decoding [23] in which the z_{mn}'s and q_{mn}'s in Fig. 3.1 are 1-bit messages, and $\boxplus$ and $\boxminus$ symbols both indicate exclusive-OR operations. As a result, the broadcast message, P_m, in Fig. 3.2 is also a 1-bit value, hence no word length increase is required.

To compare the effectiveness of different broadcasting schemes in a partially-parallel LDPC decoder, we define the node-to-node communication complexity as the number of unique LLR messages being read/written from/to the shared memory per iteration. [2] For an LDPC code with E edges in the graph, $2E$ global read operations are involved in each iteration: E reads in the check-update phase and E reads in the variable-update phase, independent of the type of broadcasting. The number of write operations, however, varies with the choice of broadcasting. In a conventional decoder, each variable node generates d_v unique messages, so $Nd_v = E$ write operations are needed in variable-update phase. Similarly, E write operations are needed for the check-update phase. In a half-broadcasting scheme in which the check nodes each broadcast a single LLR message, the number of variable-update phase memory writes continues to be E; however, the number of check-update phase write operations is reduced to M. Finally, in a full-broadcast scheme the number of required write operations for variable and check-update phase is reduced to N and M, respectively.

To summarize, one decoding iteration in a no-broadcast scheme requires $2E + 2E = 4E$ read/write operations. With half-broadcasting this is reduced to $2E + E + M = (3 + 1/d_c)E$. With full-broadcasting this is further reduced to $2E + N + M = (2 + 1/d_v + 1/d_c)E$. For moderately large values of d_v and d_c, half-broadcasting and full-broadcasting result in close to 25% and 50% memory access reductions, respectively, compared to a no broadcasting scheme. As an example, for a (6-32)-regular (2048,1723) LDPC code in the 10GBase-T standard ($E = 12288$), the number of global memory accesses per iteration is reduced by 24% and 45% using half-broadcasting and full-broadcasting, respectively.

The above comparisons suggest that in fully-parallel decoders, half-broadcasting provides a better trade-off between relaxing node-to-node communication complexity and logic overhead. Meanwhile, the full-broadcasting architecture can be the preferred choice in low-parallelism LDPC decoders where logic area constitutes a small portion

[2] In addition to communication complexity, one can also investigate the effectiveness of broadcasting schemes from energy consumption point of view and possibly assigning different energy costs for the read and write operations.

of the total decoder area and hence the logic overhead due to full-broadcasting can be tolerated.

4 Block-Interlaced Decoding

4.1 Background

Each iteration of LDPC message-passing decoding consists of updating check and variable node outputs based on functions similar to (3.1) and (3.3). Due to the data dependency, the computation of $z_{mn}^{(i)}$ in (3.3) cannot be started until all the $q_{m'n}^{(i)}$, $m' \in M(n)$ have been calculated from (3.1). Similarly, the check-update phase cannot be started before completion of the variable update phase of the previous iteration.

In [1], an overlapped message-passing scheduling is proposed for quasi-cyclic LDPC codes decoded in a memory-based architecture. The idea is to perform the check (variable) node update phase in an order such that the variable (check) node update phase can be started for some variable (check) nodes before all the check (variable) node updates are complete. A modified scheduling algorithm for overlapped message passing is proposed in [50] which can be applied to any LDPC code. The algorithm in [50] is based on permuting rows and columns of H so that the sub-matrices in the lower-left and upper-right corners of the permuted H are all zeros. Fig. 4.1 shows an example of a 6×12 parity check matrix before and after the row and column permutations as proposed in [50]. In Fig. 4.2 the corresponding timing diagrams are shown. The timing diagrams are based on a partially-parallel decoder architecture with two shared CPUs and three shared VPUs. In contrast to the diagram in Fig. 4.2a, the diagram in Fig. 4.2(b) shows that using the permuted H matrix of Fig. 4.1(b), some variable nodes (i.e., columns 2, 6 and 9) can be updated even before the check node update phase is complete. Similarly, some check nodes (i.e., rows 1 and 3) can be updated before the variable node update phase from previous iteration is complete. This overlapped scheme reduces the amount of time required to perform one decoding iteration, hence it results in higher decoding throughput. The maximum possible amount of overlapping in a partially-parallel message-passing LDPC decoder is directly related to the number of all-zero sub-matrices (i.e., the $m \times n$ sub-matrices,

where m and n are the number of shared CPUs and VPUs, respectively) in the lower left and upper right corners of the permuted H matrix.

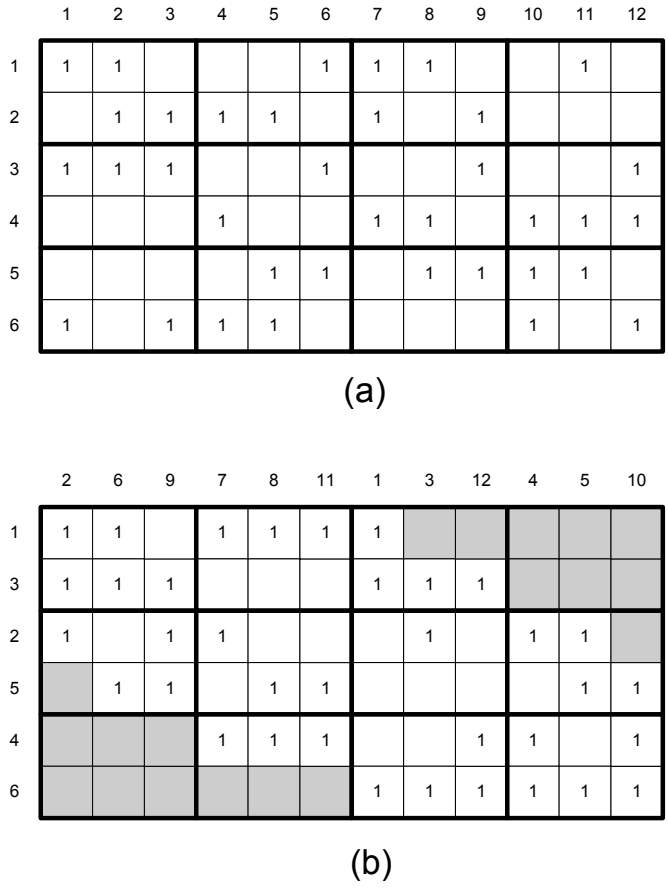


Figure 4.1: An example of row/column permutation of H matrix in overlapped message-passing [1]: (a) the original H matrix, (b) the permuted H matrix.

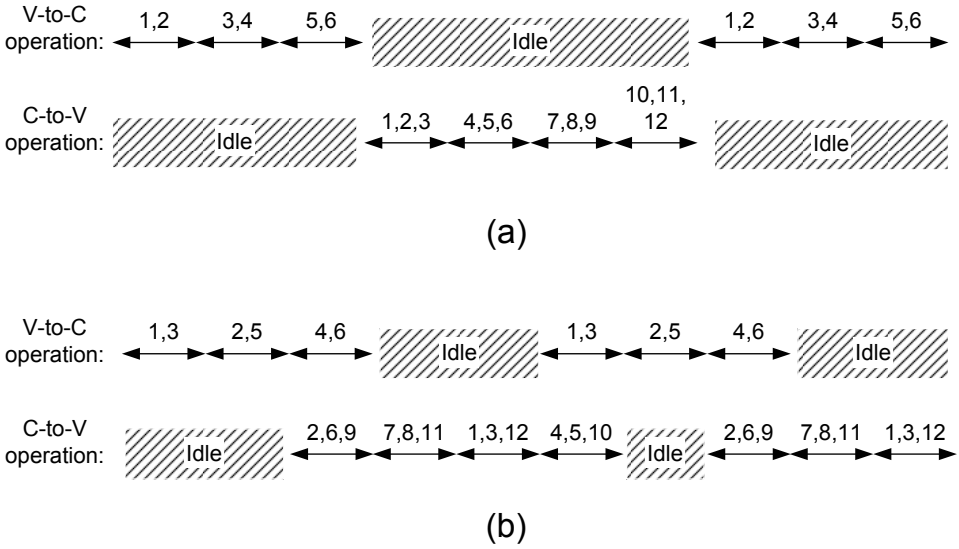


Figure 4.2: Message-passing timing diagram for (a) the original matrix of Fig. 4.1.a (b) for the permuted matrix of Fig. 4.1.b.

One drawback of the overlapped message-passing is that as the number of parallel variable and check processing units increases, the potential increase in throughput is decreased. This can be seen from Fig. 4.1. As the parallelism increases the relative size of the sub-matrices in H is also increased and as a result it becomes increasingly harder to find a permuted H matrix with all-zero sub-matrices in the lower-left and upper-right corners. (In the case of a fully-parallel architecture, the sub-matrix becomes the same size as the parity check matrix and as a result no overlapping is possible). In addition, the potential throughput gain is reduced for smaller H matrices with high variable and check node degrees.

4.2 Interlacing

The following paragraphs describe an alternative approach to increase the decoding throughput. We will show that unlike the overlapped message-passing technique, the proposed scheme is most applicable for the fully-parallel decoder architecture. We will also evaluate the throughput improvement and associated costs of this approach for two fully-parallel LDPC decoders.

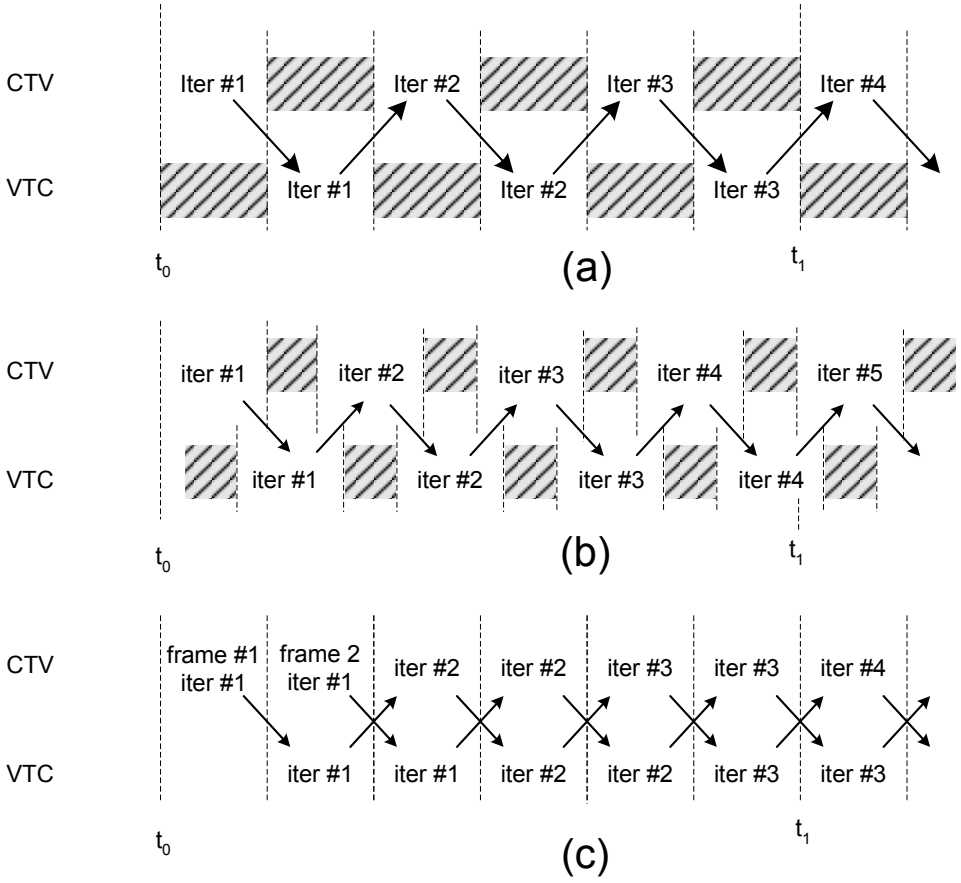


Figure 4.3: A timing diagram for the message-passing algorithm: (a) conventional (b) overlapped message passing [1] (c) block interlacing.

In this new approach, instead of overlapping the variable-update phase and check-update phases of the same frame, we *interlace* the decoding of two consecutive frames such that when variable (check) node update phase is being applied to one frame, another frame is in the check (variable) node update phase and vice versa. Fig. 4.3 compares the conventional and overlapped message-passing timing diagram with that of the block interlaced timing diagram. It can be seen that from t_0 to t_1 the conventional scheme in Fig. 4.3(a) has finished 3 iterations and the overlapped message passing has completed 4 iterations (or it is ready to start 5th iteration) whereas in the same time period the block interlaced scheme in Fig. 4.3(c) has finished 3 iterations for frame 1 and 3 iterations for frame 2, i.e., 6 iterations in total. This improvement in decoder throughput results from eliminating the idle times in the timing diagram, or equivalently, increasing the hardware utilization of the decoder.

The block-interlacing is similar to the fully-pipelined architecture in [48] where logic utilization is increased by having one set of VPUs and CPUs for each iteration. Here, a single set of VPUs and CPUs exchange information back and forth while performing the iterations. As a result the memory and the logic size does not increase with number of iterations. The block interlacing technique can be used both in fully-parallel and partially-parallel LDPC decoders and, unlike the overlapped message-passing technique of [50], the throughput improvement is independent of the parity-check matrix.

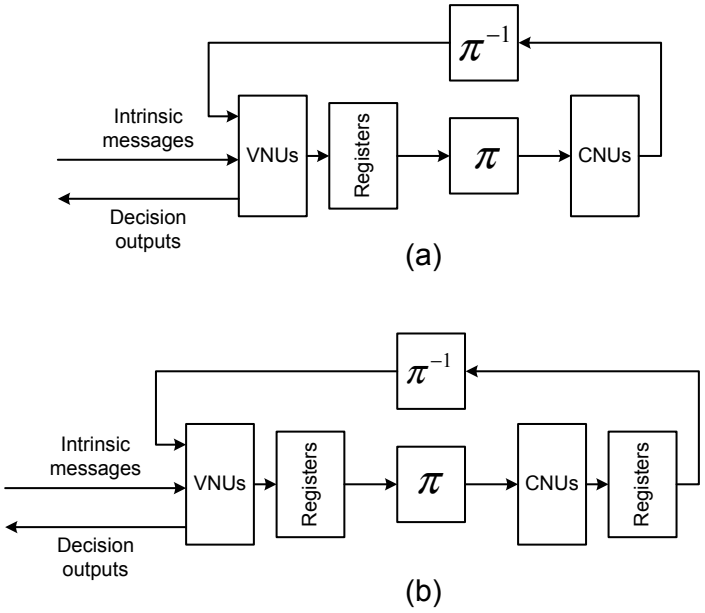


Figure 4.4: (a) conventional, and (b) block-interlaced architecture for fully-parallel LDPC decoders.

For fully-parallel decoders, the only additional requirement for block interlacing are the pipeline registers at the node outputs (Fig.4.4). In these decoders, the gate-count overhead due to the pipeline registers is relatively small compared with the large logic gate count. As an example, for the two decoders presented below, the hardware overhead is less than 10%. This overhead is easily justified by the improvement in decoder throughput. The exact amount of throughput improvement depends on the reduction of the critical path as the result of adding pipeline registers. Typically, the critical path in the non-interlaced scheme of Fig.4.4.a starts from the registers at the output of the VNUs and goes through the combinational logic in CNUs and VNUs

before arriving at the VNU output registers. In the ideal case where the critical path in Fig. 4.4.b is broken to two paths with half length, the throughput is doubled. However, the critical path is not always exactly reduced to half. As an example, for the two decoders that will be presented in this chapter, the improvement is about 1.6X and 1.7X.

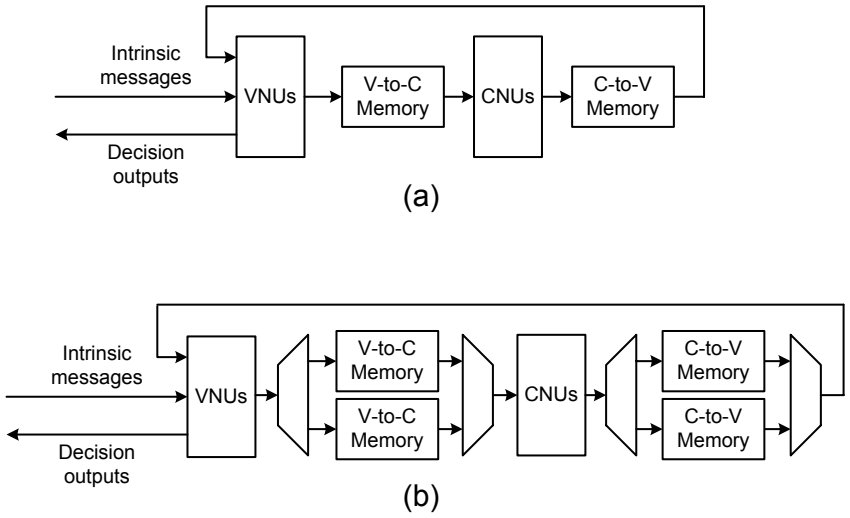


Figure 4.5: (a) conventional, and (b) block-interlaced architecture for partially-parallel LDPC decoders.

For partially-parallel decoders, block interlacing keeps the CPU and VPU logic unchanged but requires doubling the size of the LLR memories so that the CPUs and VPUs can switch between two memory banks as they switch between the two different frames (Fig. 4.5). In decoders with a small number of CPUs and VPUs (i.e., low parallelism) memory is the dominant portion of the decoder, so block interlacing will nearly double the power and area but, in high-parallelism decoders, the logic size becomes dominant and overhead is reduced.

4.3 Implementations

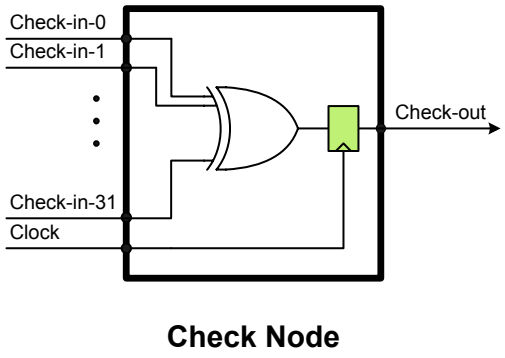
To demonstrate the block interlacing, we compare the characteristics of two implemented fully-parallel LDPC decoders. Each of these decoders was implemented twice: once without block interlacing and once with interlacing.

4.3.1 Design 1: (2048, 1723) RS-based LDPC decoders

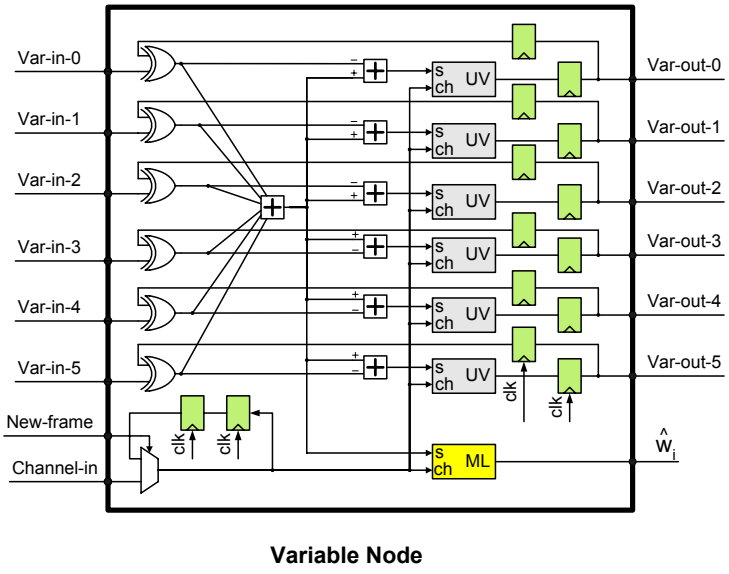
The first two decoders are rate-0.84 (2048, 1723) RS-based (6,32)-regular fully-parallel LDPC decoders in a 0.18-μm CMOS-6M technology. This particular code is now adopted for the IEEE 802.3an 10GBase-T (10 Gigabit Ethernet on twisted pair) standard [51]. Both decoders performs 32 iterations of hard-decision message-passing decoding [23]. Their top-level block diagram is similar to the fully-parallel architecture in Fig. 2.5 with 2048 VNUs and 1723 CNUs.

For the first design (Design 1A), the variable and check nodes are the same as the half-broadcast hard-decision message passing architecture that was shown in Fig. 3.7. The second design (design 1B) is also based on the half-broadcast hard-decision architecture of Fig. 3.7, but it also includes the additional flip flops to permit block interlacing. Fig. 4.6 shows the variable and check nodes for design 1B.

The comparison between the timing diagrams of conventional decoder in Designs 1A and the block interlaced decoder in Design 1B is shown in Fig. 4.7. Although each decoding iteration in Design 1B requires double the number of clock cycles compared to Design 1A, the overlapped decoding schedule and also the higher clock frequency in 1B results in about 1.6X throughput increase, as will be shown in the results section.



(a)



(b)

Figure 4.6: (a) CNU, and (b) VNU for the block-interlaced LDPC decoder in Design 1B.

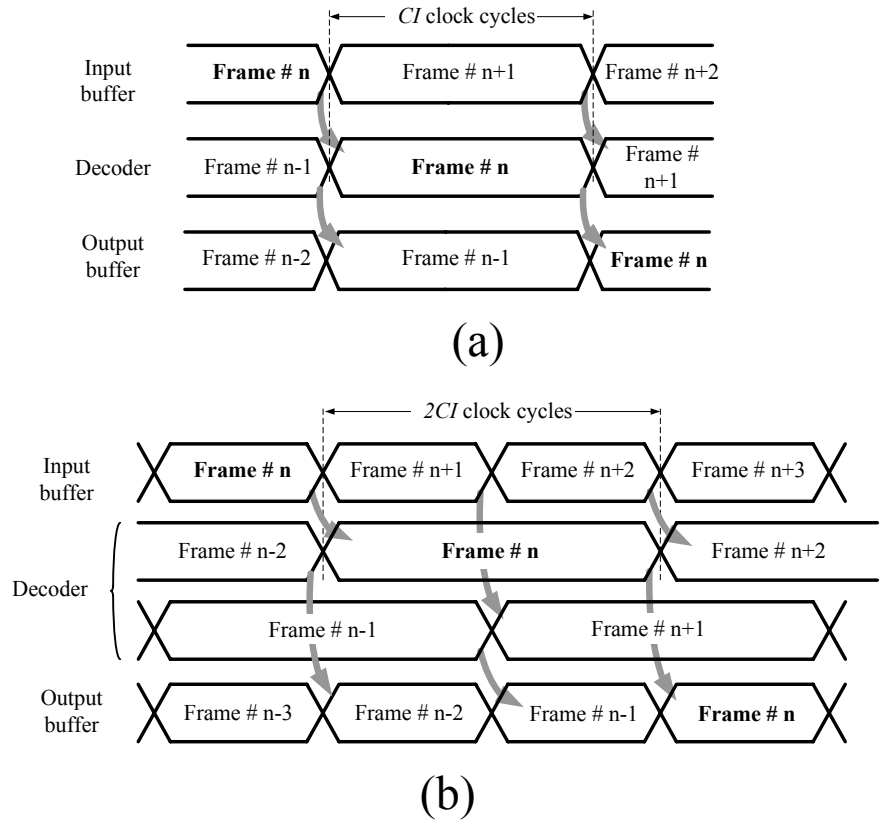


Figure 4.7: Timing diagram for (a) Design 1A, (b) Design 1B (with block interlacing).

4.3.2 Design 2: (660, 484) PEG LDPC decoders

The next two decoders also use a similar top-level diagram as in Fig. 2.5 but are based on a (660,484) (4,15)-regular LDPC code generated using the progressive edge growth (PEG) algorithm [28]. They perform 16 iterations of min-sum decoding (as described in Chapter 2) with 4-bit quantized LLR values.

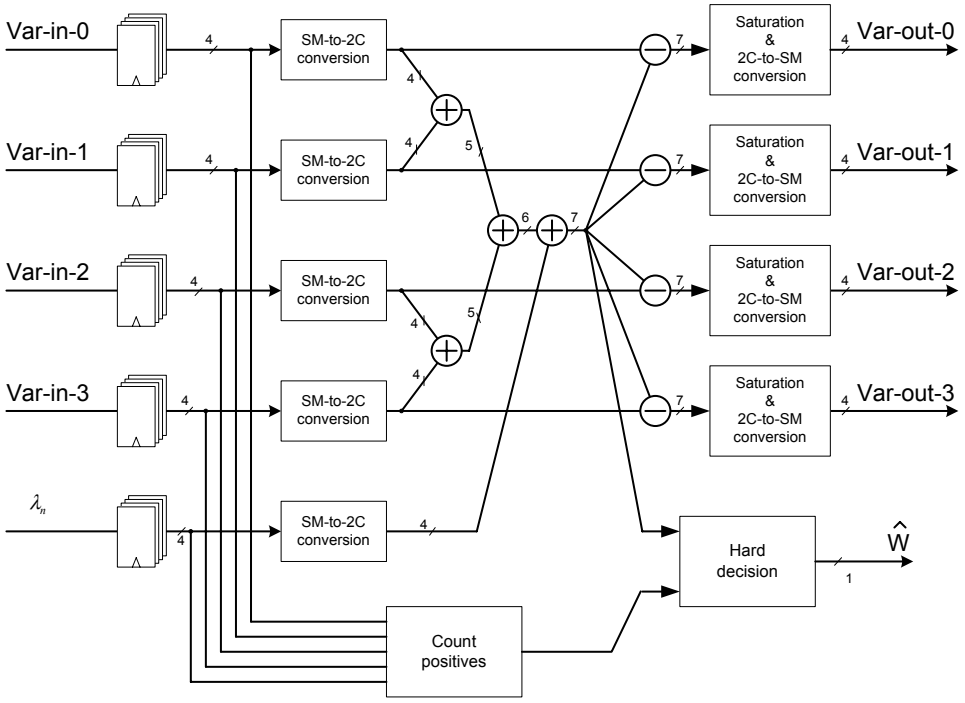


Figure 4.8: VNU for the fully-parallel LDPC decoders in Designs 2A and 2B.

The first design (Design 2A) performs conventional 4-bit min-sum decoding with no block-interlacing included. Fig. 4.8 shows the VNU architecture. The 4-bit input LLRs are first converted from sign-magnitude format to 2's complement format which is more efficient for performing the additions and subtractions. The 7-bit outputs of the subtracters are saturated to generate 4-bit results and are then converted to the sign-magnitude format to be processed by the CNUs. The hard decision block in the VNU generates the 1 bit output of the decoder corresponding to that variable node in the final iteration based on the sign of the sum of all messages. If the sum is zero, the output is determined based on the majority of the input signs.

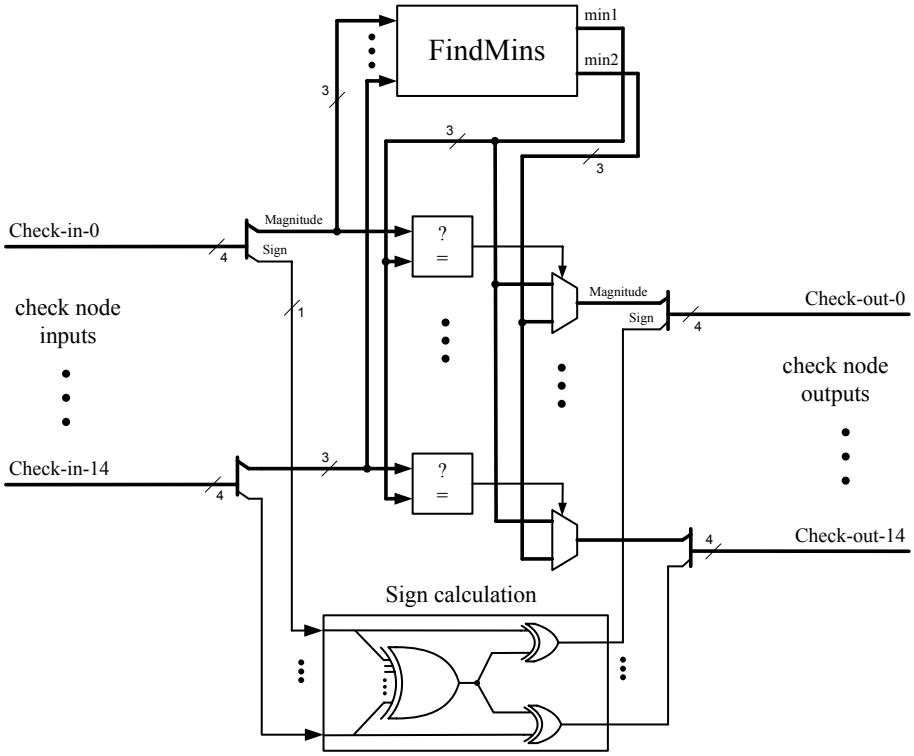


Figure 4.9: CNU for the fully-parallel LDPC decoder in Designs 2A.

Fig. 4.9 shows the internal architectures of the CNUs for Design 2A. According to the min-sum update rule, the magnitude of each CNU output is the minimum of the magnitudes from all other inputs. These output magnitudes can be efficiently calculated by finding the first and second minimums among all input magnitudes. These minimums are calculated in the `FindMins` block as shown in Fig. 4.10.

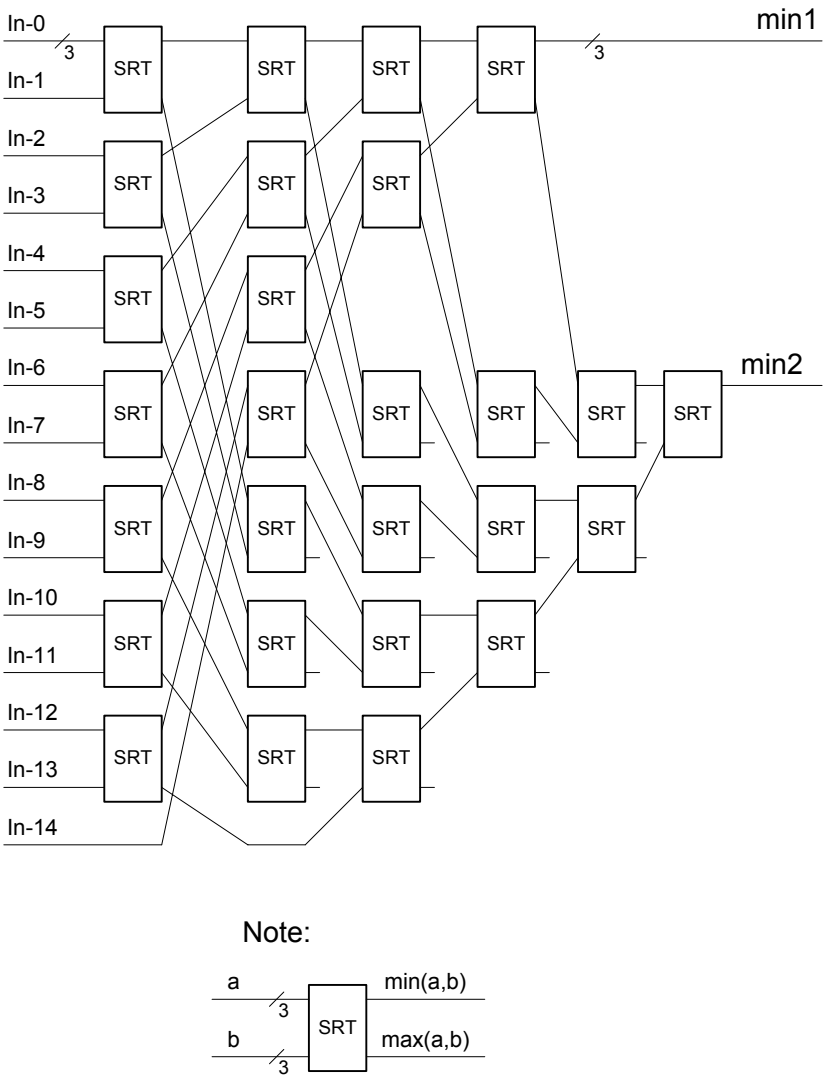


Figure 4.10: The `FindMins` block to calculate the first and second minimums (i.e., `min1` and `min2`) in Fig. 4.9.

The second design (Design 2B) has the same VNUs as Design 2A. The CNUs are also similar to the CNUs in Design 2A with the exception that the inputs at the CNUs are registered.

4.3.3 Results

Table 4.1 summarizes post-layout simulation results (after timing-driven place and route) for all the decoders. The results show that for the Designs 1A and 1B, the block interlacing increases the total throughput by 62% at the cost of just 5.5% more core area. For the Designs 2A and 2B throughput is increased by 71%, with a 9.5% area overhead. Notice that block interlacing has not exactly doubled the throughput as one would ideally expect from Fig. 4.3. The reason is that the pipeline registers do not exactly break the critical path into two identical length paths. So, the maximum clock frequency is limited by the delay in the longer path.

Table 4.1: Summary of LDPC decoder characteristics.

	Design 1A	Design 1B	Design 2A	Design 2B
CMOS process	0.18 μm	0.18 μm	90 nm	90 nm
Architecture	parallel	parallel	parallel	parallel
Block interlacing	no	yes	no	yes
Code length (bits)	2,048	2,048	660	660
Code rate	0.84	0.84	0.73	0.73
No. of Edges	12288	12288	2640	2640
Iter. per frame	32	32	16	16
Clock cycles per iteration	1	2	1	2
quantization (bits)	1	1	4	4
Max freq. (MHz)	60	96	83	142
Core area (mm^2)	9.8	11.4	2.72	3.06
Min. size NAND gate count (k)	522	551	685	750
Total throughput (Gbps)	3.84	6.14	3.42	5.86

5 Power-Saving Techniques for LDPC Decoders

This chapter investigates power saving techniques for decoding LDPC codes. First, we analyze the advantage of the fully-parallel decoding architecture from an energy efficiency point of view. Then, we will discuss the achievable power savings by applying an early termination architecture without degrading the coding performance. Finally, we will explore the power vs. BER vs. throughput design space.

5.1 Parallelism and Supply-Voltage Scaling

5.1.1 Background

A generic LDPC decoder architecture is shown in Fig. 5.1. It comprises K_v shared variable node update units (VNUs), K_c shared check node update units (CNUs), and a shared memory fabric used to communicate messages between the VNUs and CNUs. Inputs to each CNU are the outputs of VNUs fetched from memory. After performing some computation (e.g., the MIN operation for the magnitude and parity calculation for the signs in min-sum decoding), the CNU's outputs are written back into the extrinsic memory. Similarly, inputs to each VNU arrive from the channel and several CNUs via memory. After performing the message update (e.g., the SUM operation in min-sum decoding), the VNU's outputs are written back into the extrinsic memory for use by the CNUs in the next decoding iteration. Decoding proceeds with all CNUs and VNUs alternately performing their computations for a fixed number of iterations, after which the decoded bits are obtained from one final computation performed by the VNUs.

By increasing the number of VNUs and CNUs, K_v and K_c, the decoder performs more computations in parallel. When the decoder is operated from a fixed supply voltage, such increased parallelism may be used to achieve higher throughput, with

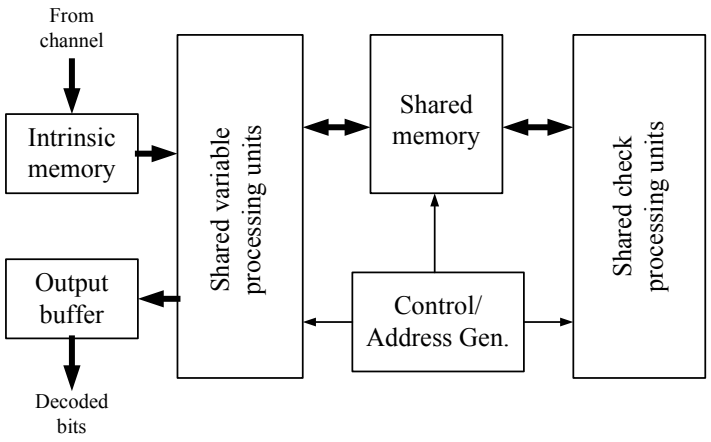


Figure 5.1: A partially-parallel LDPC decoder.

attendant increases in power and area. However, it is well known that increased parallelism can also permit a digital system to operate from a lower supply voltage with constant throughput, resulting in greatly decreased power consumption [52]. In general, the power advantages offered by parallelism are mitigated by the overhead associated with multiplexing and demultiplexing the system's inputs and outputs amongst several parallel computing units. However, in the case of an LDPC decoder, all of the signals required for each iteration are already available in parallel in the extrinsic memory (Fig. 5.1). The inherent parallelism of LDPC iterative decoding with long block lengths is, therefore, well suited to implementation with a low supply voltage. Until now, this property has not been fully exploited to design a low-voltage, low-power LDPC decoder.

5.1.2 Analysis

The reduced supply voltage obtainable using increased parallelism is described qualitatively in Fig. 5.2. There is a practical limit to the decoder's parallelism power savings when the numbers of VNUs and CNUs equal the total number of variable and check node computations required in each iteration. Further increases in K_v or K_c are not straightforwardly possible since the required input messages are not available in memory. As shown in Fig. 5.2, unless the targeted throughput is low, the supply voltage will remain significantly higher than the MOS threshold voltage. Al-

though sub-threshold circuits have been shown to be energy-efficient, they are mostly suitable for low-to-mid performance systems [53] with relaxed constraints on throughput. Since many present and future applications of LDPC codes target a multi-Gbps throughput, our analysis will proceed assuming a square-law MOS model.

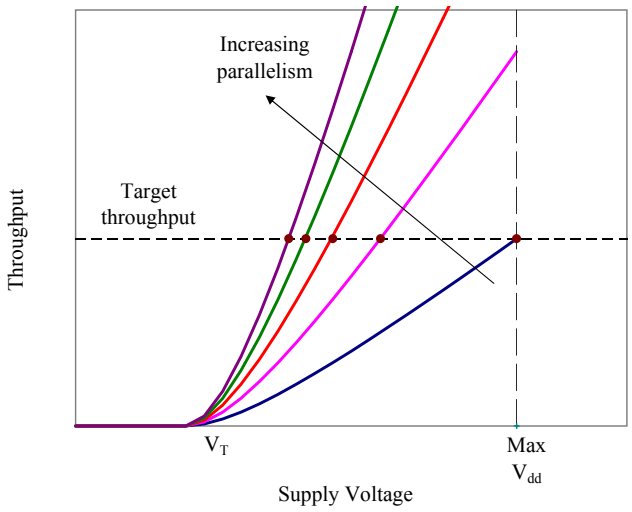


Figure 5.2: Increased parallelism allows reduced supply voltage.

To quantify the power reduction that can be offered by highly-parallel LDPC decoding architectures, let us compare two decoders: a reference design with K_v VNUs and K_c CNUs; and a design with increased parallelism having $(k \cdot K_v)$ VNUs and $(k \cdot K_c)$ CNUs, $k > 1$. The dynamic power consumption of these decoders, operated at a clock frequency f from a supply voltage V_{dd} is $fC_{\text{eff}}V_{dd}^2$, where C_{eff} is the effective capacitance of each decoder including an activity factor. The effective capacitance in a decoder is the sum of the capacitance due to the logic gates in the VNUs and CNUs and the capacitance due to the memory cells. Note that the memory size is the same for both decoders since the total number of messages stored in each iteration is constant. Therefore, only the effective capacitances of the VNUs and CNUs are scaled by increasing parallelism. Let β be the fraction of the reference decoder's total effective capacitance that scales with k. Hence, if C_m is the effective capacitance associated with the memory and C_1 is the total effective capacitance of

the reference design, then $\beta = (C_1 - C_m)/C_1$. Since C_m does not scale with k, the effective capacitance of the parallel design is

$$C_2 = (1 + \beta(k-1))\, C_1.$$

The parallel decoder can operate at a clock frequency f_2 that is k times lower than the reference design clock frequency, f_1, while maintaining the same throughput: $f_2 = f_1/k$. Since we are striving for low-power operation, each decoder operates from the lowest supply voltage that will support its targeted clock frequency. Hence, the parallel design can be operated from a lower supply voltage (V_{dd2}) than the reference design (V_{dd1}).

Using the first-order derivation for the propagation delay, T_d, of a gate with a load capacitance, C_L, we obtain

$$T_d = \frac{C_L V_{dd}}{I} = \frac{C_L V_{dd}}{\mu C_{ox}(W/L)(V_{dd} - V_t)^2}. \tag{5.1}$$

Following an analysis similar to [54], we find that

$$V_{dd2} = \frac{V_{dd1}}{2}\left(2m + \frac{1}{k}(1-m)^2 + \sqrt{\left(2m + \frac{1}{k}(1-m)^2\right)^2 - 4m^2}\right) \equiv v_{sc} V_{dd1}, \tag{5.2}$$

where $m = \frac{V_t}{V_{dd1}}$. Therefore, the dynamic power dissipation for the parallel design is

$$P_2 = \frac{v_{sc}^2}{k}\,(1 + \beta(k-1))\, P_1 \equiv \eta P_1, \tag{5.3}$$

Fig. 5.3(a) shows the normalized supply voltage, v_{sc}, required for different values of k to maintain a constant throughput based on (5.2) for a typical 0.13-μm CMOS process where $V_t = 300mV$ and $V_{dd1} = 1.2V$. Fig. 5.3(b) shows the normalized power, η, versus the parallelism factor, k, for different values of β based on (5.3). It can be seen that the power reduction is greatest for small values of β since in those cases C_m is the dominant portion of C_1, so the added parallelism implies a negligible increase in the system's effective capacitance. However, significant power savings can be achieved even with high values of β. For example, with $\beta = 0.95$ and $k = 4$ the supply voltage can be reduced by about 50%, resulting in a 74% power saving compared with the reference design.

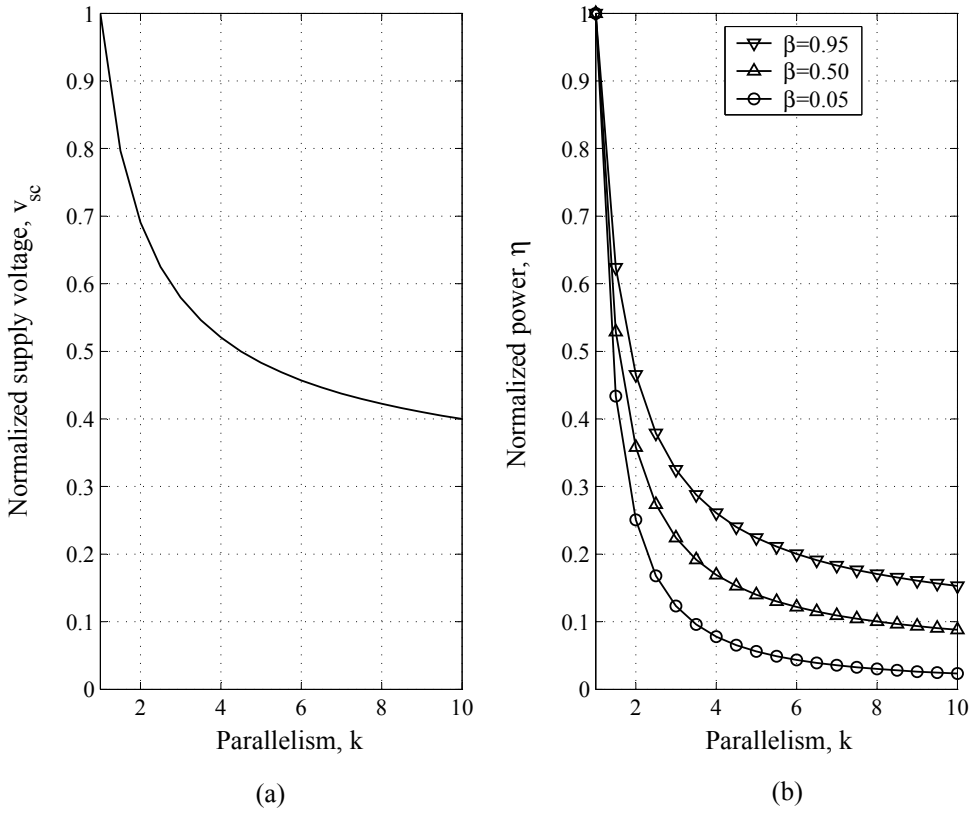


Figure 5.3: Power reduction as a result of a parallel architecture: a) Reduction in supply voltage. b) Reduction in dynamic power dissipation.

The preceding analysis makes two assumptions that have not yet been discussed:

a) Power consumption is dominated by dynamic power dissipation. Our measurements for the 0.13-μm CMOS decoder presented in Chapter 6 suggest that leakage power constitutes less than 1% of the total power dissipation when operating at the maximum clock frequency and with typical supply voltage values. This is also consistent with the power measurements reported in [15].

b) The overhead associated with the increased parallelism is negligible. This is the focus of Chapter 6.

Following the power efficiency discussion above, we have adopted a fully-parallel

architecture where a separate VNU or CNU is designated for each variable node or check node in the code Tanner graph. Another advantage of fully-parallel decoder architecture is that unlike most partially-parallel decoders that are based on a particular code construction (such as the (3, k)-regular construction in [41], or the Architecture-Aware code construction in [42]), the fully-parallel architecture can be applied to irregular codes with no constraints on the code structure. This is done simply by instantiating VNUs and CNUs of the desired degree and connecting them based on the code graph. The only consideration is that the timing performance of the decoder for irregular codes will be typically limited by a critical path through the nodes with highest degree.

The major challenge in implementing highly-parallel decoders [15] is the large area and the overhead effects such as the routing complexity that are not modeled in the discussion in this section. To reduce the effect of routing complexity, we have used a bit-serial message-passing scheme in this work where multi-bit messages are communicated between the nodes over multiple clock cycles [55]. In addition to reducing the routing complexity, the bit-serial message-passing requires less logic to perform min-sum LDPC decoding because both the MIN and SUM operations are inherently bit-serial. As a result bit-serial VNUs and CNUs can be efficiently implemented to generate only partial 1-bit extrinsic messages every clock cycle. Although bit-serial message-passing reduces the amount of global wiring, the routing complexity will eventually limit the maximum length of the LDPC codes that can be implemented in a bit-serial fully-parallel decoder. However, the important point is that the bit-serial scheme pushes the practical code length limit to higher values, making it feasible to implement fully-parallel decoders for emerging high-speed standards such as 10GBase-T or Mobile WiMAX, which specify maximum code lengths of 2048 and 2304, respectively. The bit-serial decoder developed in this work is presented in Chapter 6.

5.2 LDPC Decoding With Early Termination

5.2.1 Background

LDPC decoders generally correct most bit errors within the first few decoding iterations. Subsequent iterations provide diminishing incremental improvements in decoder performance. The number of iterations performed by the decoder, I_M, is

usually determined a priori and hard-coded based on worst-case simulations. Therefore, the decoder performs I_M iterations even though it will usually converge to its final output much sooner. We propose an energy-efficient decoder architecture that efficiently detects when it has converged to its final output and shuts off all VNUs and CNUs for the remaining iterations to save power.

Earlier work in this area has focused on identifying particular bits within each frame that appear likely to have converged [56, 57]. They have suggested that one can stop updating extrinsic messages for those reliable bits while other unreliable bits are still being decoded. The resulting power savings depends on the specific criteria used to identify the reliable bits. Unfortunately, these bits are sometimes incorrectly identified, so the decoder's performance suffers. In [58], an additional post-processing decoder is introduced to mitigate this performance degradation. Naturally, there is overhead associated with identifying the reliable bits and with the post-processing decoder. The overhead reduces the potential power savings of this approach.

In this work, instead of trying to identify individual bits that appear to have converged early, we monitor the entire frame to determine when the decoder has converged to a valid codeword. We then deactivate the entire decoder for the remaining iterations to save power. The remainder of this section describes a hardware-efficient implementation of this technique with significant power savings and no performance degradation.

5.2.2 Early termination

Although EXIT charts can be used to determine the average number of iterations required for the convergence of an LDPC decoder operating on very long block lengths [24], for practical block lengths of 1000 to 10,000 bits the estimates so obtained are inaccurate. Instead, we have used extensive simulations to investigate the convergence behavior of two practical LDPC codes.

Fig. 5.4 shows the BER versus input SNR for two different LDPC codes under 4-bit-quantized min-sum decoding. The code in Fig. 5.4(a) is the Reed-Solomon based (6,32)-regular 2048-bit LDPC code as specified for the 10Gbase-T Ethernet standard [51]. The code in Fig. 5.4(b) is the one employed in the hardware prototype that will be described in Chapter 6. Each code is simulated under a range of different I_M's. These simulations indicate that little performance improvement is observed for either code as the number of iterations is increased from $I_M = 12$ to $I_M = 16$. Therefore,

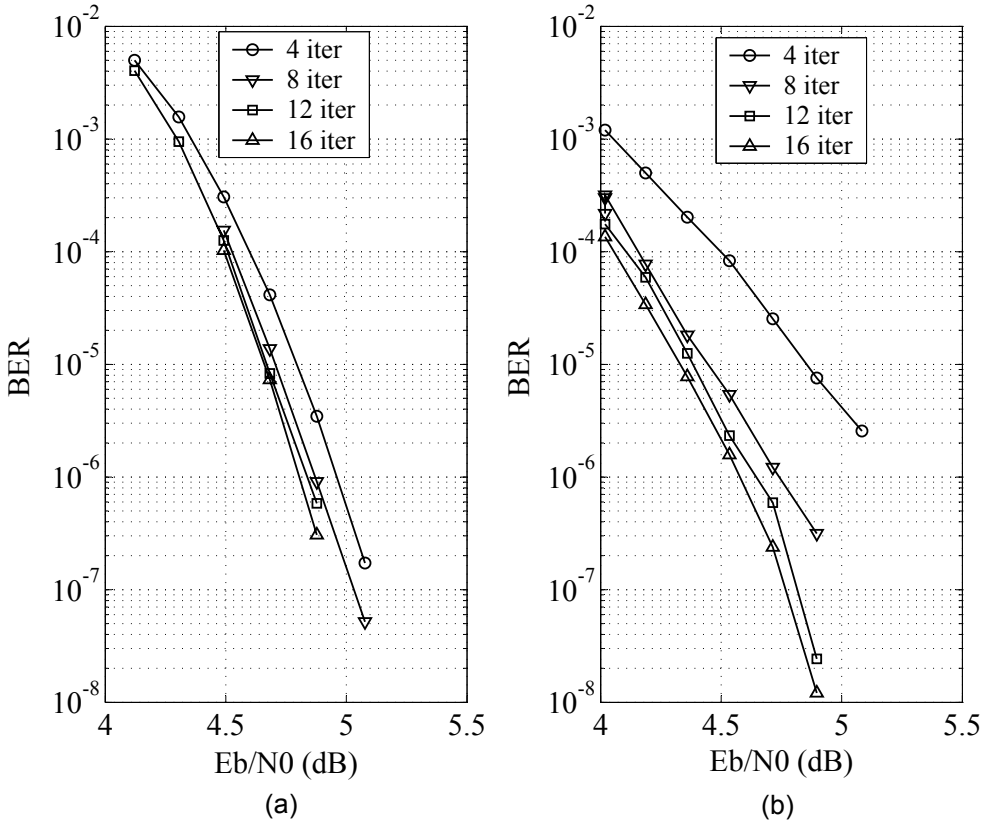


Figure 5.4: BER vs. maximum number of iterations under 4-bit quantized min-sum decoding: (a) Reed-Solomon based (6,32)-regular 2048-bit code and (b) PEG (4,15)-regular 660-bit code.

no more than $I_M = 16$ iterations are required for either code.

The convergence behavior of the same two codes is shown in Fig. 5.5, which plots the average fraction of uncorrected frames versus the iteration number. These two figures show that the vast majority of frames are correctly decoded in the first few iterations. For example, for the code in Fig. 5.5(a), at an E_b/N_0 of 5.1 dB, more than 99.99% of all frames have been successfully decoded during the first five iterations.

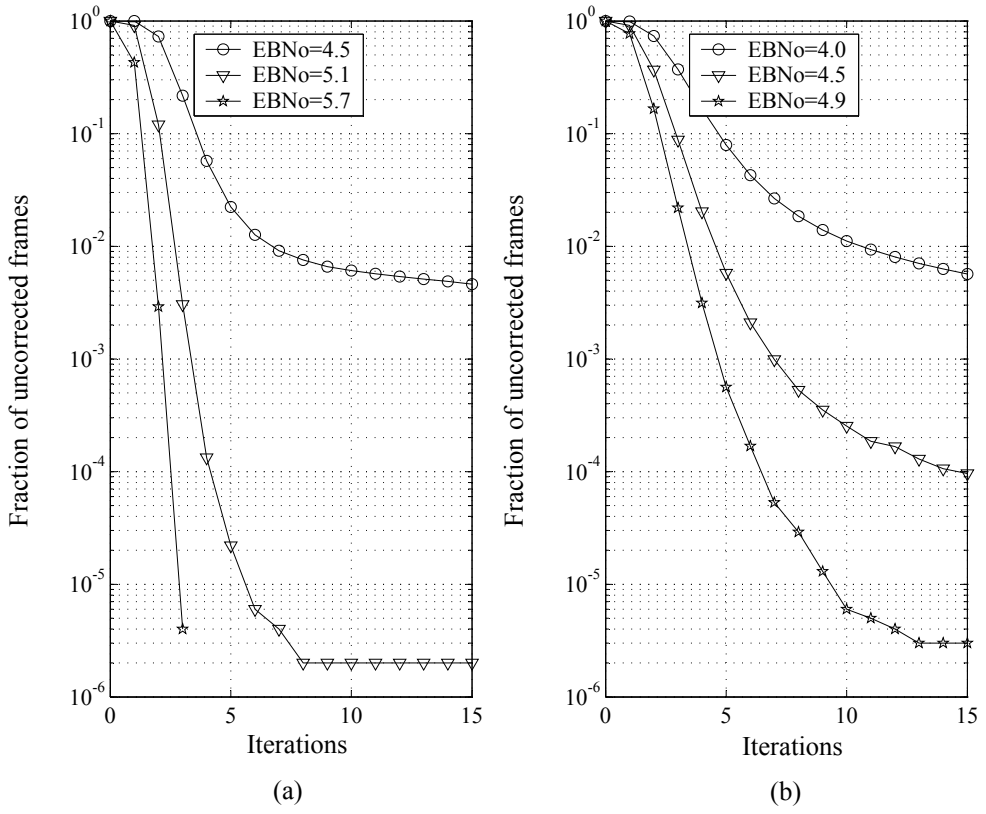


Figure 5.5: The fraction of uncorrected frames vs. iteration number for (a) a Reed-Solomon based (6,32)-regular 2048-bit code. (b) a PEG (4,15)-regular 660-bit code.

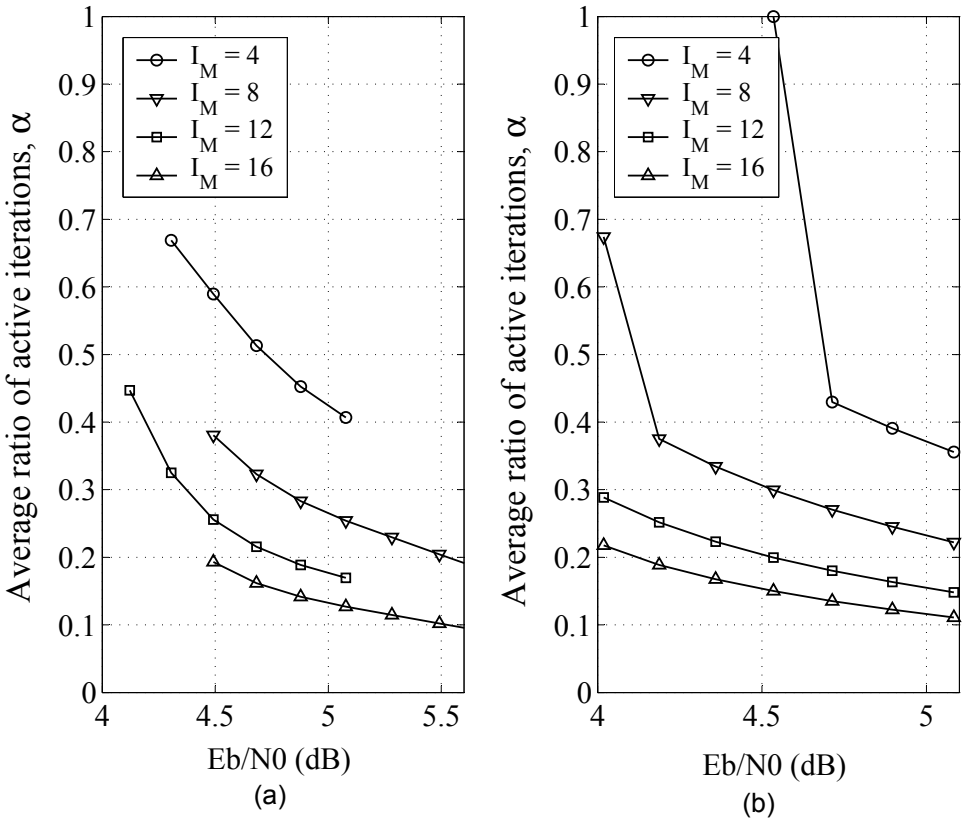


Figure 5.6: Fraction of active iterations (a) Reed-Solomon based (6,32)-regular 2048-bit code and (b) PEG (4,15)-regular 660-bit code.

Fig. 5.6 plots the ratio of the average number of required iterations to I_M, α, versus input SNR for the same two codes as in Fig. 5.5. The figure shows the graphs for I_M =4, 8, 12 and 16. For example, based on Fig. 5.6(b), for the code implemented in this work with $I_M = 15$, on average less than 3 iterations are needed per frame at SNR=4.3dB (corresponding to BER=10^{-5}). As will be shown in the results section, by exploiting this behavior and turning off the decoder in the remaining un-needed iterations, the total dynamic power is reduced by 65%.

5.2.3 Hardware implementation

The remaining task is to efficiently implement early termination in hardware, i.e., to detect that the decoder has converged to a correct codeword. A standard approach is to make final decisions in each VNU at the end of each iteration and then check if all parity constraints are satisfied. This is referred to as syndrome checking. Although straightforward, this approach has a considerable hardware cost. First, it requires additional hardware in the VNUs to make the hard decisions at the end of each iteration. Second, the hard decisions must be distributed to the destination check nodes in every iteration where syndrome checking can be performed. This distribution can be done either by dedicating extra hard wires from VNUs to the neighboring CNUs, or by sharing the same wires used for transferring extrinsic messages in a bit-serial time multiplexed fashion. Neither of these approaches is efficient because they either increase the routing complexity by adding global wires or they decrease the decoding throughput by increasing the number of clock cycles per iteration.

Alternatively, in this work we check the parity of the sign of the normal variable-to-check messages that are already required by the decoding iterations. If the parities of all these signs are satisfied, we stop decoding for that frame and compute the final hard decision at the beginning of next iteration. Although not mathematically equivalent to the standard syndrome checking, we have simulated the two LDPC codes of Fig. 5.4 with the same set of 10^6 frames both without and with early termination at E_b/N_0 ranging from 4 dB to 5.1 dB. The simulations show identical performance between the two approaches for these codes.

For the two codes discussed in this chapter, our method on average needs one extra iteration to terminate compared with the conventional syndrome checking method. This difference reduces the amount of power savings achieved compared to the conventional syndrome checking. For example, in the 660-bit decoder that will be presented in Chapter 6, conventional syndrome checking could have improved the percentage of power savings from 49% to 51% for low-SNR inputs ($E_b/N_0 \approx 3$ dB) and from 66% to 72% for high-SNR inputs ($E_b/N_0 \approx 6$ dB). In spite of the reduced power savings, we have adopted this new termination method for two reasons. First, in contrast to conventional early termination, our termination method does not increase the number of VNU-to-CNU wires, nor does it require extra clock cycles per iteration to distribute the hard decision results to the CNUs. Second, this approach requires minimal hardware overhead since most of the calculations are already part of the

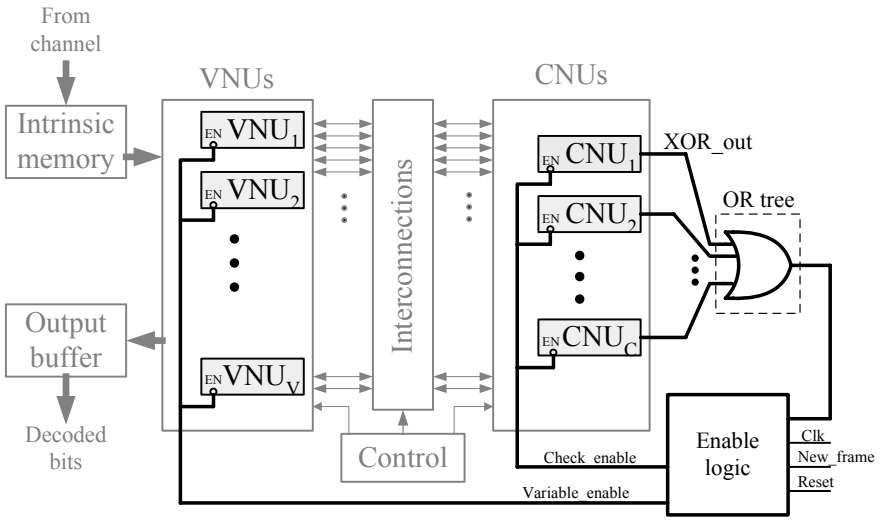


Figure 5.7: The fully-parallel iterative LDPC decoder with early termination functionality.

normal VNU and CNU operations.

Fig. 5.7 shows the block diagram of a decoder with early termination logic. It is similar to the one in Fig. 2.5 with a few added blocks: First, all the parity results are ORed. The output of the OR tree is zero only when all the parities are satisfied. Second, a termination logic block generates the proper disable/enable signals for the VNUs and CNUs depending on the value of the OR tree output. If the output of the OR tree is zero, it keeps the VNUs and CNUs disabled for the remaining iterations. Fig. 5.8 shows the timing diagrams of the decoder, with and without early termination. It shows that the decoding throughput is the same in both cases since the start time for decoding the frames is identical. However the power consumption is reduced in Fig. 5.8 because the decoder is turned off as soon as a correct codeword is detected.

The synthesis results show that the added OR tree and the enable/disable functionality required in CNUs and VNUs adds only less than 0.1% and 0.5% to the total decoder gate count, respectively. It should also be noted that no additional logic is required inside the CNUs to generate the XOR-out signals as this value is already available from the sign-calculation block inside the CNUs (Fig. 4.9).

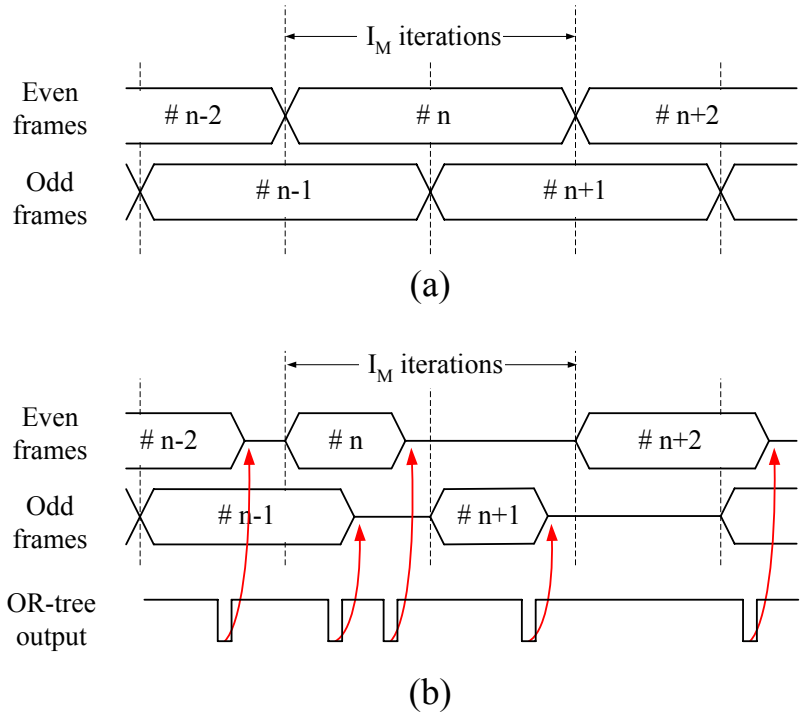


Figure 5.8: Block-interlaced decoding timing diagram (a) without early termination, (b) with early termination.

5.3 Power vs. BER Performance

When designing an LDPC decoder, the value of I_M (i.e., the total number of decoding iterations per frame) should be determined such that for a given decoding throughput the target BER performance is maintained and at the same time the power consumption is kept low.

Although higher values of I_M are desirable for good BER performance, unnecessarily large values of I_M result in insignificant BER improvement. At the same time, for a given decoding throughput, a higher value of I_M directly translates to higher required clock frequency and as a result a higher power consumption. In this section, we show the effect of I_M on BER and power performance. We also show the effect of early termination in reducing power consumption under different values of I_M.

Fig. 5.9 summarizes the effect of I_M on BER performance and normalized power consumption for the (660, 484) PEG LDPC code. Fig. 5.10 shows similar results for the RS-based (2048, 1723) LDPC code. As expected, in both cases increasing I_M

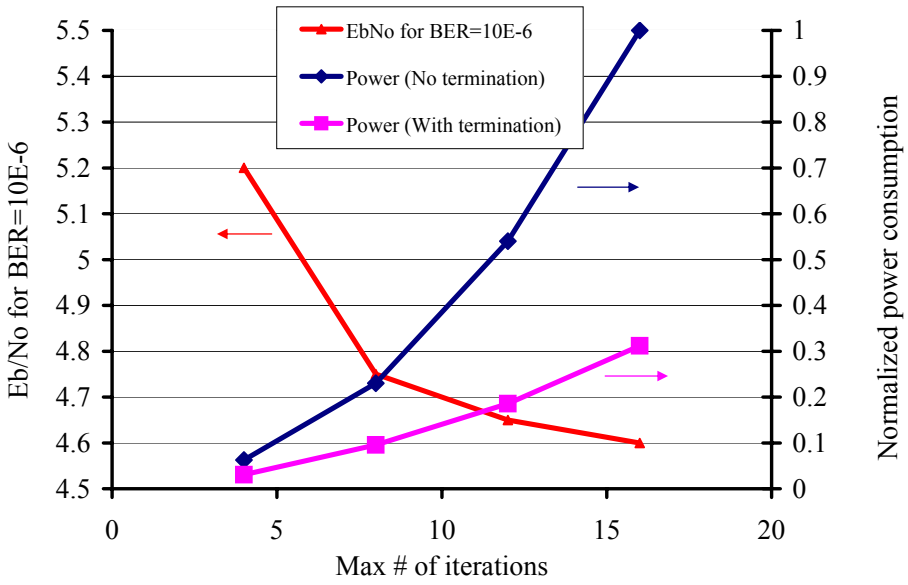


Figure 5.9: The effect of I_M on BER and power performance under a fixed decoding throughput for the (660, 484) PEG LDPC code.

from 4 to 16 improves the BER performance and the distance from the Shannon limit is reduced. It can be seen that for the code in Fig. 5.9, the BER improvement is more than for the code in Fig. 5.10. Although not illustrated here, it is also known that irregular codes typically require on average a larger number of decoding iterations before converging to a valid codeword.

The power consumption curves in Fig. 5.9 and Fig. 5.10 are shown for two cases: once without and once with early termination. The power values in these curves are normalized with respect to the highest power consumption, which is when I_M=16 and also no early termination is applied.

For the curves with no early termination, the power values are obtained from the $P = fCV_{dd}^2$ formula and based on the following facts:

a) Since the same parallelism level is maintained, the value of C stays unchanged.

b) For constant decoding throughput and constant parallelism level, the required operating clock frequency, f, is proportional to I_M.

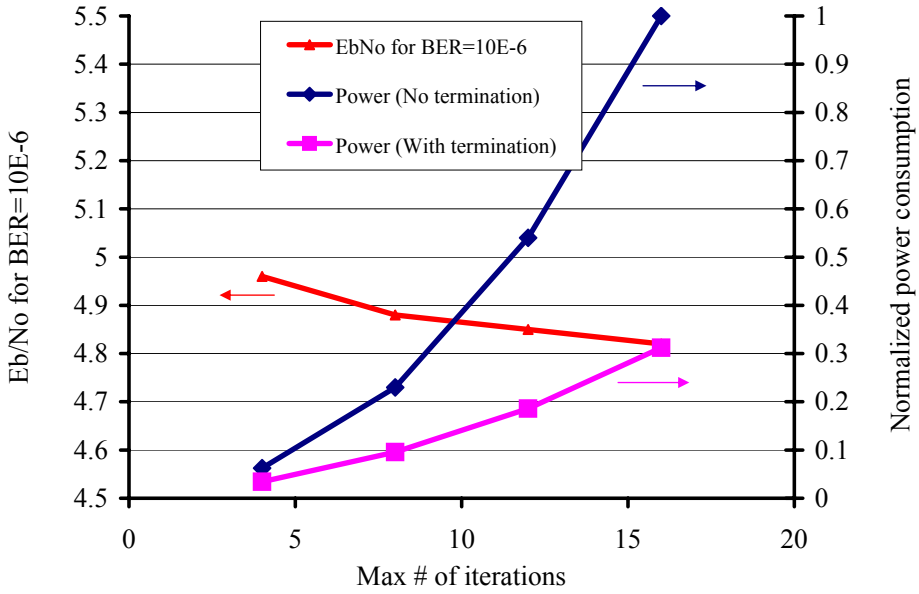


Figure 5.10: The effect of I_M on BER and power performance under a fixed decoding throughput for the RS-based (2048, 1723) LDPC code.

c) For the new f, the value of the V_{dd} can be scaled using (5.2). (For the curves in Fig. 5.9 and Fig. 5.10 we have used $m = V_t/V_{dd1} = 0.3/1.2$.)

For the curves with early termination, the power values P'_D are calculated from the corresponding P_D values from no-termination curves and using

$$P'_D = (1 + \gamma)(p_c + \alpha(1 - p_c))P_D, \tag{5.4}$$

where γ accounts for the overhead of the early termination logic, p_c is the fraction of dynamic power attributable to the clock tree, and α is the ratio of active iterations similar to the values plotted in Fig. 5.6(b). This expression accounts for the fact that early termination does not decrease the dynamic power in the clock tree. From the power measurements for the decoder reported in Chapter 6, the value of p_c is approximately 0.2. Also, from the synthesis results the value of γ is estimated to be less than 0.006.

Both figures show that in comparison to no early termination, the early termina-

tion reduces the power consumption by about 69% at $I_m = 16$ and by about 50% at $I_M = 4$. The figures also show that in both cases the value of I_M=8 seems to provide a good trade off between power consumption and coding performance.

6 Bit-Serial Message-Passing

6.1 Motivation

Following the discussions in the previous chapters, we have concluded that the fully-parallel LDPC decoder architecture is preferable both in terms of decoding throughput and energy efficiency. The main challenge in implementing a fully-parallel decoder is the congestion that arises when routing the interconnections between the processing nodes [15]. This routing congestion results in a large decoder with low area utilization, and degrades the timing performance due to the presence of long interconnects across the chip. These problems are exacerbated because word lengths of 4 to 8 bits are generally required to represent each extrinsic message. In addition, in most fully-parallel decoders, separate sets of wires are used for each direction of messages on each edge. As an example, a conventional fully-parallel LDPC decoder for the 2048-bit LDPC code from the 10GBase-T standard with variable node degree of 6 and 6-bit quantized LLR messages will require 147,456 global wires to connect the variable and check nodes.

To alleviate the routing problem caused by the large number of global wires in a fully-parallel LDPC decoder, in this section we describe a bit-serial message-passing scheme. Fig. 6.1(a) shows a conventional decoder where q-bit messages are calculated and communicated between VNUs and CNUs in parallel. Alternatively, Fig. 6.1(b) shows the proposed decoder in which messages are generated and transferred between nodes bit-serially over one wire in q clock cycles.

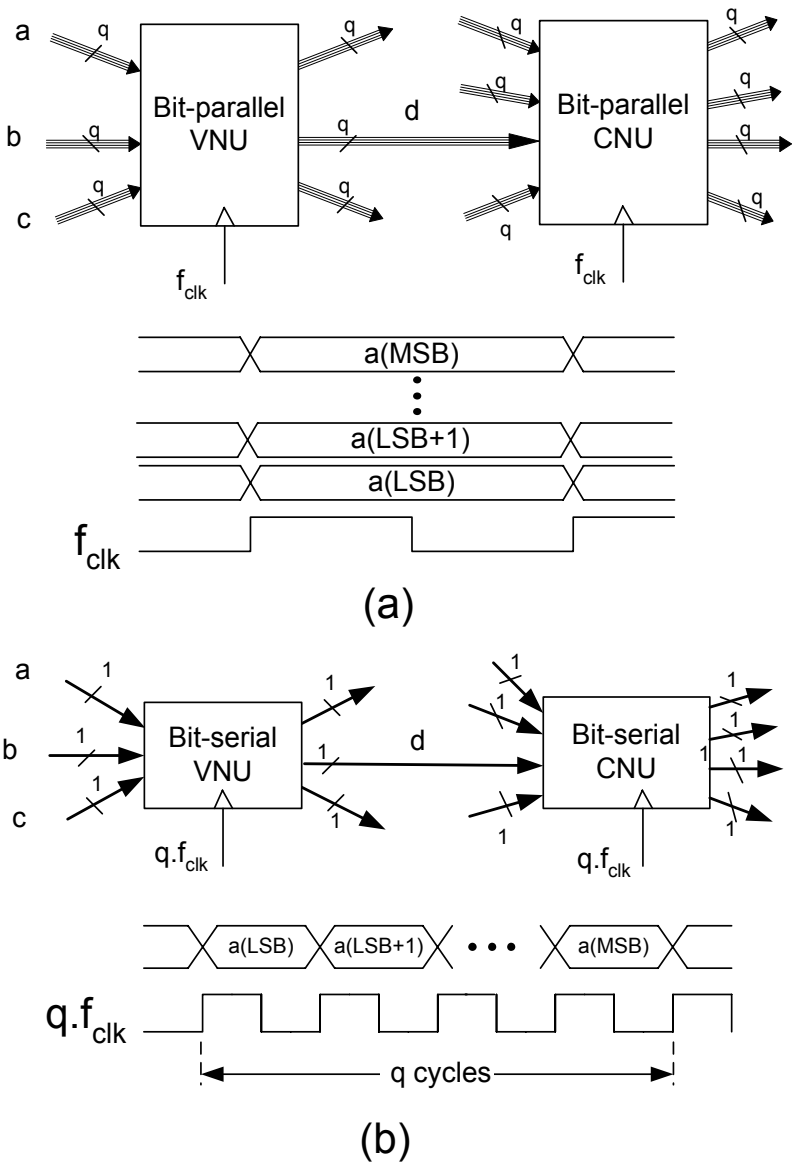


Figure 6.1: Bit-serial vs. bit-parallel message passing.

The bit-serial computation is of particular interest in min-sum decoding because both the Min and Sum operations inside the CNUs and VNUs can be naturally performed bit-serially. In addition to simplifying the node-to-node interconnections, the bit-serial approach has several other advantages for fully-parallel LDPC decoders. In a bit serial scheme, the word length of computations can be increased simply by increasing the number of clock cycles allocated for transmitting the messages. Using this property, the precision of the decoder can be made programmable just by re-timing the node-to-node message transfers without the need for extra routing channels. Programmability of the decoder word length allows one to efficiently trade-off complexity for error correction performance. This in turn allows efficient implementation of gear-shift decoding [59]. Gear-shift decoding is based on the idea of changing the decoding update rule used in different iterations to simultaneously optimize hardware complexity and error correction performance. For instance, gear shift decoding often suggests applying a complex powerful update rule in the first few iterations followed by simpler update functions in later iterations. Bit-serial computation allows efficient *shifting* between update rules by changing the computations word length.

Bit-serial decoding, however, imposes some challenges. The immediate effect is that it reduces the decoder throughput compared with fully-parallel implementations, as multiple clock cycles are required for transmitting a single message. Also some common check and variable update functions cannot be efficiently implemented bit-serially. Although bit-serial fully-parallel LDPC decoders have a lower throughput compared with bit-parallel fully-parallel LDPC decoders, we will show in this work that their throughput can still be higher than hardware-sharing decoder schemes.

Stochastic computation [34] is similar to bit-serial computation in that it communicates extrinsic messages over single wires. It has a very simple check and variable node architecture but needs a significant amount of hardware overhead in order to translate the stochastic messages at the decoder inputs and outputs. In addition, the stochastic computation uses a redundant number representation which limits the decoder throughput.

6.2 Bit-serial Blocks for Conventional Min-Sum Decoding

This section illustrates how bit-serial VNUs and CNUs can be implemented for a fully-parallel min-sum LDPC decoder with update rules as specified by (2.8) and (2.7).

6.2.1 CNU architecture

Fig. 6.2 shows the top level architecture of a bit-serial CNU for min-sum decoding. Each input is an n-bit sign-magnitude binary number which is received bit-serially. Each output is a bit-serial n-bit number calculated based on (2.8). Similar to the architecture in Fig. 4.9, each output sign is calculated by XORing all the other input signs. In contrast to Fig. 4.9, here the output magnitudes are calculated from the

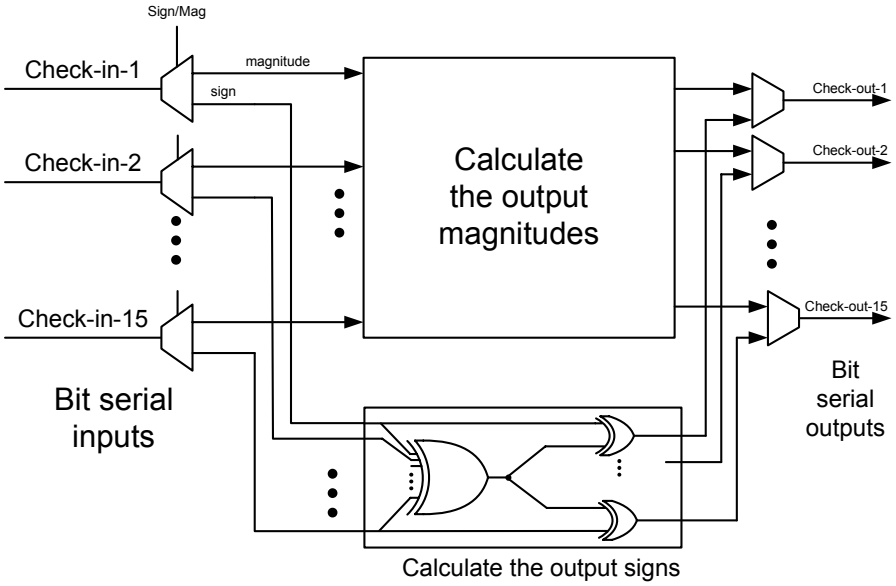
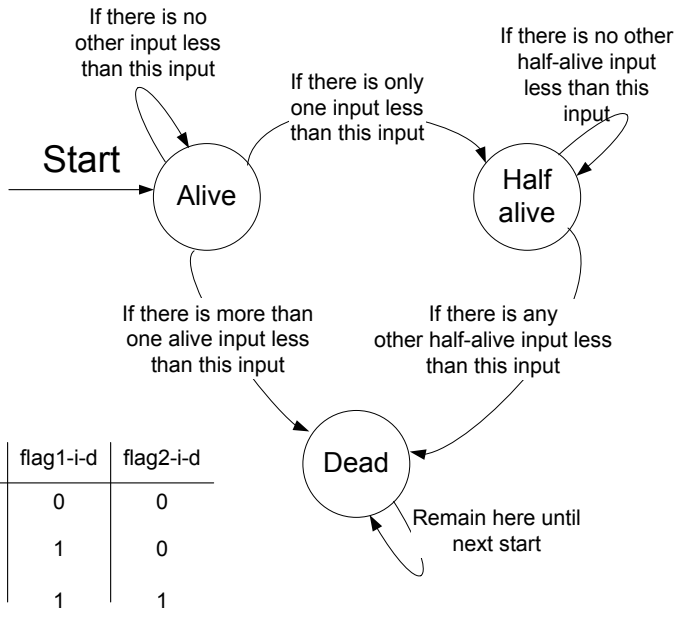


Figure 6.2: Top-level diagram for a bit-serial CNU.

bit-serial input magnitudes arriving MSB first. To perform the calculations in (2.8) efficiently, this module finds the first and second distinct minimums at each cycle and also keeps track of the number of inputs that share the same magnitude as the first distinct minimum. Each output is then decided based on these values.

Since input magnitudes arrive one bit per cycle, there needs to be a finite state machine associated with each input/output, as shown in Fig 6.3, to keep track of the history of each input in comparison with other inputs. In the process for finding the first and second minimums, each input can be in one of the following three states.

- **Alive**: The input is a potential candidate for being the first distinct minimum.
- **Half-alive**: The input cannot be the first minimum anymore but is a potential candidate for being the second minimum.
- **Dead**: The input can be neither the first nor the second minimum.



State	flag1-i-d	flag2-i-d
Alive	0	0
Half Alive	1	0
Dead	1	1

Figure 6.3: The finite-state machine for bit-serial calculation of CNU output magnitudes in a conventional min-sum decoder.

At the beginning of the Min calculation, all of the FSMs are set to be in the `Alive` state since all inputs can potentially be a first minimum. As the new bits arrive, the input states change depending on their current state and the number of other inputs with smaller magnitude.

Fig. 6.4 shows the bit-serial implementation of the magnitude calculation module for a degree-15 CNU (d_c=15). The state of the i-th input ($1 \leq i \leq 15$) is stored in two flip flops (i.e., `flag1_i_d` and `flag2_i_d` flip flops) based on the state assignment values as shown in Fig. 6.3. The logic before the `flag1_i_d` flip flops determines which edges are still in the `Alive` state and passes the `Alive` inputs to a 15-input AND gate in order to find the value of the `1st_min` signal. Similarly, the logic before the `flag2_i_d` flip flops determines which inputs are `Half-alive` and passes them to a 15-input AND gate to generate the `2nd_min` signal. The output value for the i-th edge, `mag_out_i`, is selected either from `1st_min` or `2nd_min` signals. By default, the `mag_out_i` is `1st_min` unless the i-th edge is the only one edge that is still `Alive`, in which case the value of `2nd_min` signal is passed as the `mag_out_i`.

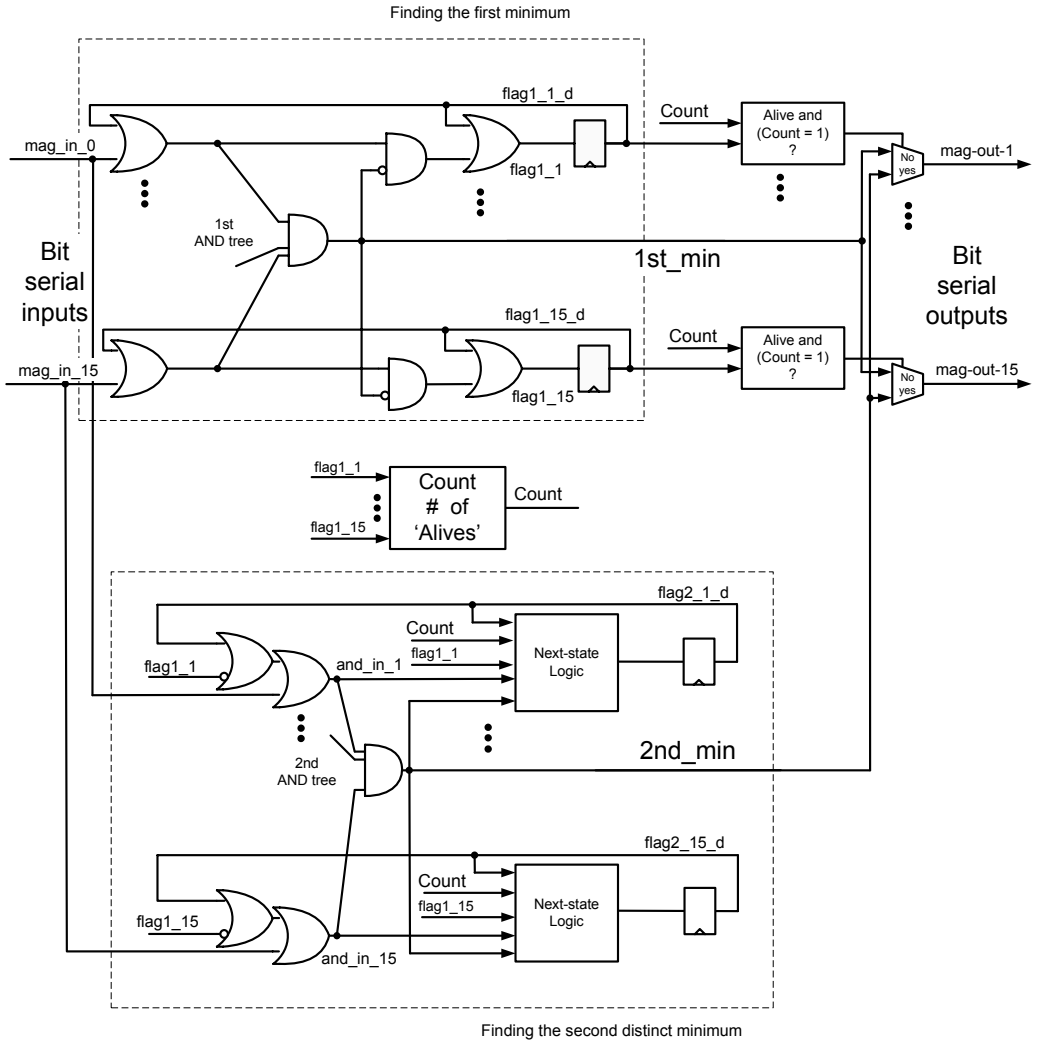


Figure 6.4: The magnitude calculation module for the CNU in Fig. 6.2.

6.2.2 VNU architecture

To find an efficient bit-serial variable node architecture, we investigated two alternatives. The first architecture, shown in Fig. 6.5, is based on a forward-backward computation [2]. The main difference between our approach and [2] is that here all the inputs and outputs are bit-serial. The main problem with the forward-backward architecture of Fig. 6.5 is that for a variable node of degree d_v the critical path consists of a chain of $(d_v - 2)$ two-input adders. For LDPC codes with relatively high d_v, this can limit the timing performance of the decoder.

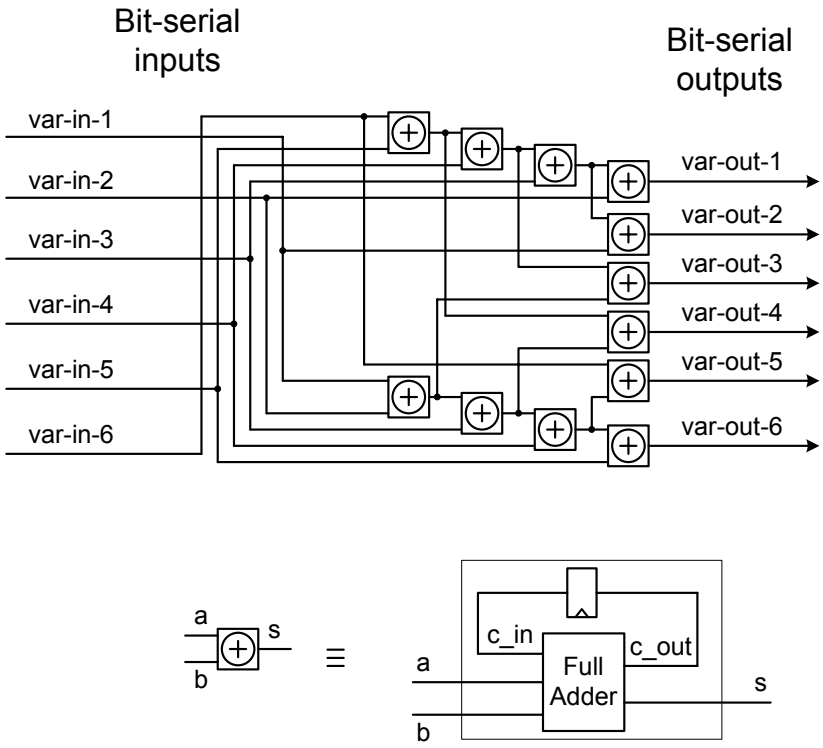


Figure 6.5: A degree-6 VNU for computing (2.7) with a forward-backward architecture [2]. Each adder box consists of a full-adder and a flip-flop to store the carry from the previous cycle.

The second variable node architecture investigated in this work is shown in Fig. 6.6. In this architecture, the bit-serial inputs are first converted to parallel inputs and then the additions are performed in one cycle using parallel adders/subtracters.

The parallel outputs are finally converted back to bit-serial format before being sent to check nodes.

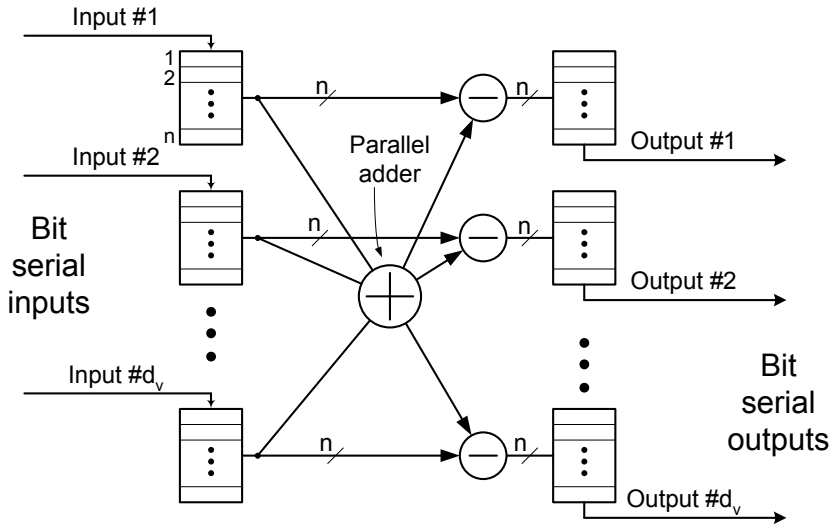


Figure 6.6: A VNU architecture for computing (2.7) with parallel adders and parallel-serial converters at the inputs and outputs.

Table 6.1 summarizes the VLSI hardware cost and timing performance of two degree-6 variable nodes corresponding to the two above alternatives. The parameters in this table are based on the synthesis results using a 90-nm CMOS standard cell library and with 3-bit quantization. Based on Table 6.1, we have used the variable node architecture of Fig. 6.6 in the design presented in this work since it is superior both in terms of timing and area.

Table 6.1: Comparison between the variable node architectures of Fig. 6.5 (forward-backward) and Fig. 6.6 (parallel adder/subtracters) with $d_v = 6$ and 3-bit quantization synthesized with CMOS 90-nm library cells.

Architecture	Forward-backward	Parallel adder
Combinational area (μm^2)	2484	2099
Flip flops area (μm^2)	623	405
Total area (μm^2)	3107	2504
Minimum clock period (nsec)	3	2.20

6.3 Bit-Serial Blocks for Approximate Min-Sum Decoding

Although the conventional MS decoding algorithm can be implemented in a bit-serial fashion as described in Section 6.2, in this section we introduce an approximation to the MS algorithm that reduces the hardware complexity of CNUs while causing minimal degradation in code performance. We then show the corresponding bit-serial CNU architecture for the simplified algorithm. It should also be noted that this approximation is also applicable to conventional bit-parallel hardware decoders.

6.3.1 Approximate Min-Sum decoding

The first step is to replace the min-sum check update rule,

$$\epsilon_{mn}^{(i)} = \min_{n' \in N(m) \backslash n} |z_{mn'}^{(i)}| \prod_{n' \in N(m) \backslash n} \mathrm{sgn}(\mathrm{z}_{\mathrm{mn'}}^{(\mathrm{i})}), \tag{6.1}$$

with

$$\epsilon_{mn}^{(i)} = \min_{n' \in N(m)} |z_{mn'}^{(i)}| \prod_{n' \in N(m)} \mathrm{sgn}(\mathrm{z}_{\mathrm{mn'}}^{(\mathrm{i})}). \tag{6.2}$$

In other words, the sign of the check node outputs are calculated exactly the same way as before but now the output magnitude is the minimum of magnitudes of *all* input messages.

Fig. 6.7 compares the BER performance of original MS decoding algorithm with that of the modified MS based on (6.2) for two RS-based LDPC codes [10] using full-precision computations. This graph shows that with full-precision computations, the two algorithms perform almost identically. It is clear that a check update rule as in (6.2) significantly reduces the hardware complexity. The reason is that once the minimum among all input magnitudes is found it can be sent out as the magnitude of all the outgoing messages, $\epsilon_{mn}^{(i)}$, for all $n \in N(m)$.

It is observed that although the above modification to MS results in almost no performance loss under full-precision operations, it introduces a considerable loss when performed in finite-precision. Fig. 6.8 shows the effect of the MS approximation when applied to quantized messages. In the following paragraphs we introduce a further change to the modified MS decoding algorithm that reduces the performance gap shown above.

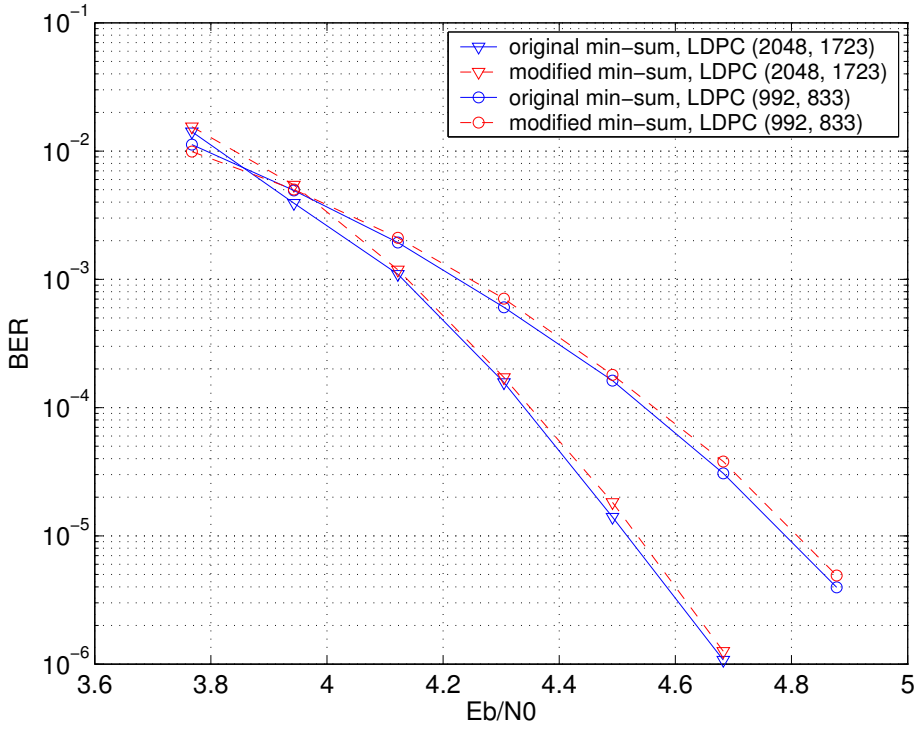


Figure 6.7: Comparison between original min-sum and modified min-sum under full-precision operations for (2048, 1723) and (992, 833) LDPC codes.

The sign of the output messages in the new check update rule is the same as in (6.2). The magnitude of the output message is calculated as follows. First, for check node c_m, in the i-th iteration, we define $M_m^{(i)} = \min_{n' \in N(m)} |z_{mn'}^{(i)}|$. We also define $1 \leq T_m^{(i)} \leq d_c$ as the number of inputs $z_{mj}^{(i)}$ to check node c_m that satisfy $|z_{mj}^{(i)}| = M_m^{(i)}$. The magnitudes of the check node outputs are then calculated as

$$|\epsilon_{mn}^{(i)}| = \begin{cases} M_m^{(i)} + 1 & \text{if } T_m^{(i)} = 1 \text{ and } |z_{mn}^{(i)}| = M_m^{(i)} \\ M_m^{(i)} & \text{otherwise.} \end{cases} \tag{6.3}$$

In other words, the output magnitudes in (6.3) are calculated in the same way as in (6.2) with only one exception: if there is one unique input to the check node m with a magnitude equal to $M_m^{(i)}$, the outgoing message on that edge is set to $M_m^{(i)} + 1$ instead of $M_m^{(i)}$. We add this one LSB correction factor to reflect the fact that based on the original update rule in (6.1), the magnitude on this edge should be larger than $M_m^{(i)}$.

Simulation results plotted in Fig. 6.8 show that with the new check update rule

using 4-bit quantization the BER performance gap with respect to the original 4-bit MS algorithm is reduced from 0.7 dB to less that 0.1 dB at BER of 10^{-6}. More importantly, the error floor effect is also avoided.

The simulation results in Fig. 6.8 are for the same (2048, 1723) LDPC code as in Fig. 6.7. We expect that the approximate min-sum, as explained above, would have a similar effect on BER performance for other LDPC codes with relatively large check node degree. This is because in large-degree check nodes there is a high probability that the first and second minimums differ by a small amount, and, as a result, the performance gap between approximate min-sum and the original min-sum be reduced. A detailed and more vigorous analysis of approximate min-sum decoding performance on general families of LDPC codes is however subject to further research.

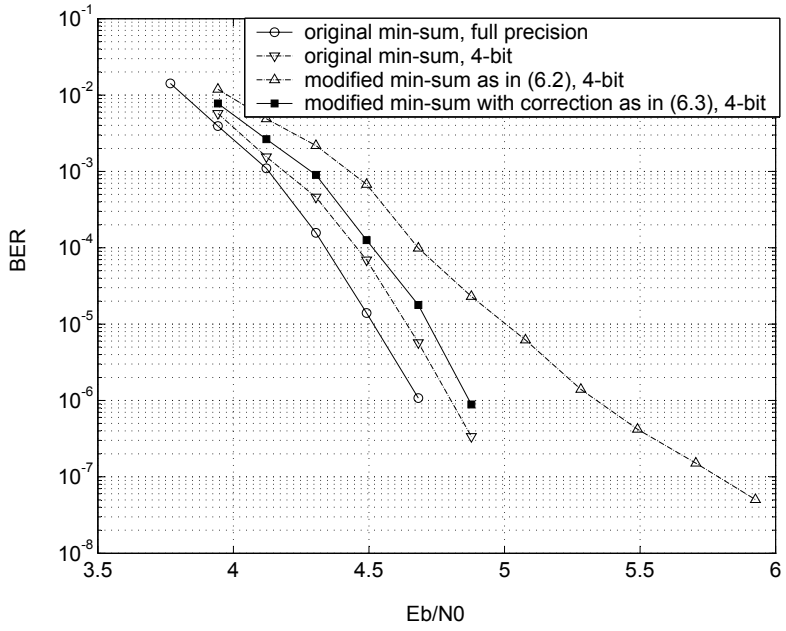


Figure 6.8: Comparison between the original min-sum and modified min-sum as in (6.2) and (6.3) under fixed-point operations for (2048, 1723) LDPC code.

6.3.2 CNU architecture for approximate Min-Sum decoding

This section describes an internal architecture for the bit-serial CNU based on (6.3). As discussed in Section 6.3.1, in the modified MS algorithm only the smallest magni-

tude among all check inputs needs to be found. Fig. 6.9 shows the pipelined bit-serial module that finds the minimum of the check inputs. This module receives d_c inputs. Associated with each input there is a flip flop acting as status flag which indicates whether that input is still a candidate for being the minimum. At the beginning, the status flags are all reset to zero. As the MSB bits are received some flags become '1' indicating that the corresponding input is out of competition.

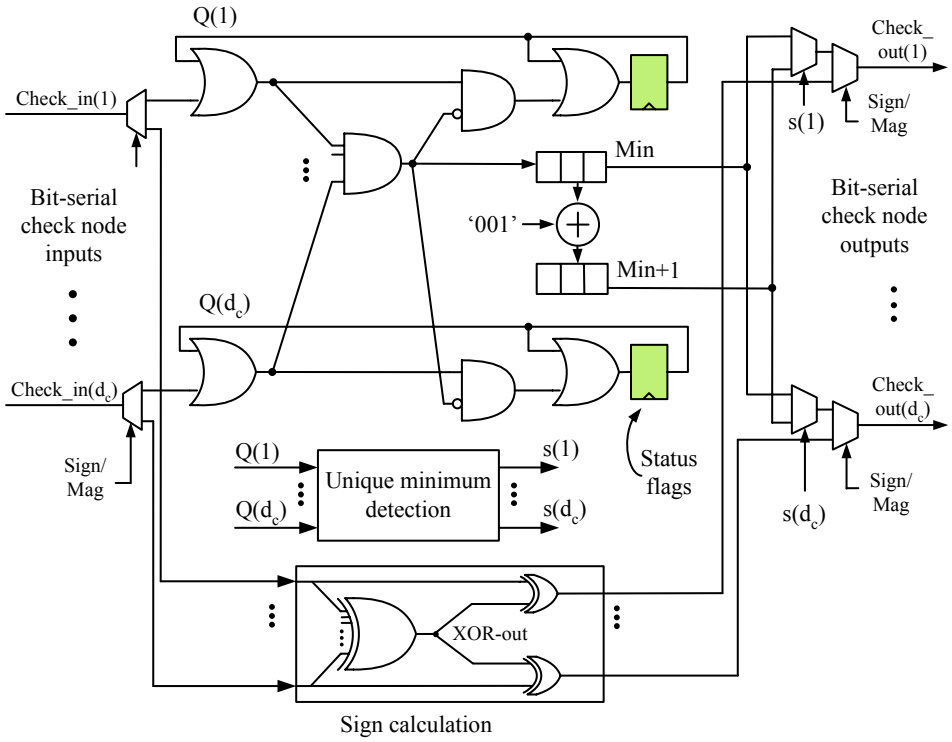


Figure 6.9: A bit-serial module for detecting the minimum magnitude of the check node inputs.

In spite of the extra hardware needed for the correction term, the VLSI implementation of a degree-15 check node using a 90-nm CMOS cell library shows that a check node based on (6.3) is 40% smaller than a check node based on (2.8). This is because there is no need to calculate the second minimum which is needed in conventional min-sum algorithm.

6.4 Implementations

We illustrate two implemented decoders in this section to demonstrate the timing performance and power efficiency of bit-serial fully-parallel LDPC decoding. The first decoder is developed on an Altera Stratix EP1S80 FPGA device. The second decoders is fabricated in a 0.13-μm CMOS process using standard cell libraries.

6.4.1 An FPGA 480-bit LDPC decoder

This decoder is based on a (480, 355) (4, 15)-regular RS-based LDPC decoder and is implemented on a single Altera Stratix EP1S80 FPGA device using a configurable prototyping board called the Transmogrifier-4 (TM-4)[60].

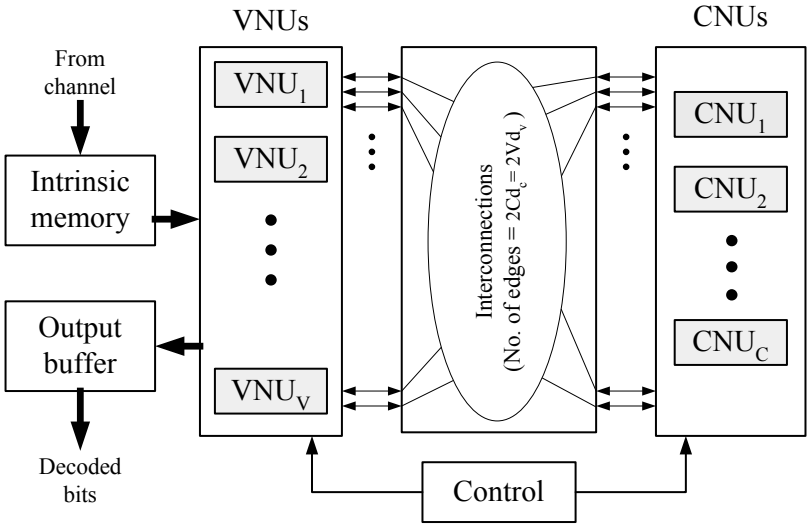


Figure 6.10: A fully-parallel LDPC decoder.

The high-level architecture of the decoder is shown in Fig. 6.10. All message updates and message transfers are performed bit-serially. The decoder updates the extrinsic messages using the node architectures of Fig. 6.9 and Fig. 6.6. Since the updated messages are carried bit-serially over single wires, the complexity of node-to-node interconnections is less than that of conventional bit-parallel fully-parallel decoders [15].

Both the check and variable nodes in this design are pipelined. For n-bit quantized input messages they generate n-bit output messages in n clock cycles. Each

iteration of LDPC message-passing decoding consists of one check and one variable node update. As a result, using a conventional scheme, $2n$ clock cycles are needed to complete one iteration. However, in this design we adopt a block-interlaced scheme [55] as described in Chapter 4 where two frames are processed in the decoder simultaneously in an interlaced fashion; while the check nodes process one frame, the variable nodes are processing the neighboring frame. So, in effect it takes only n cycles to complete one iteration, hence doubling the throughput.

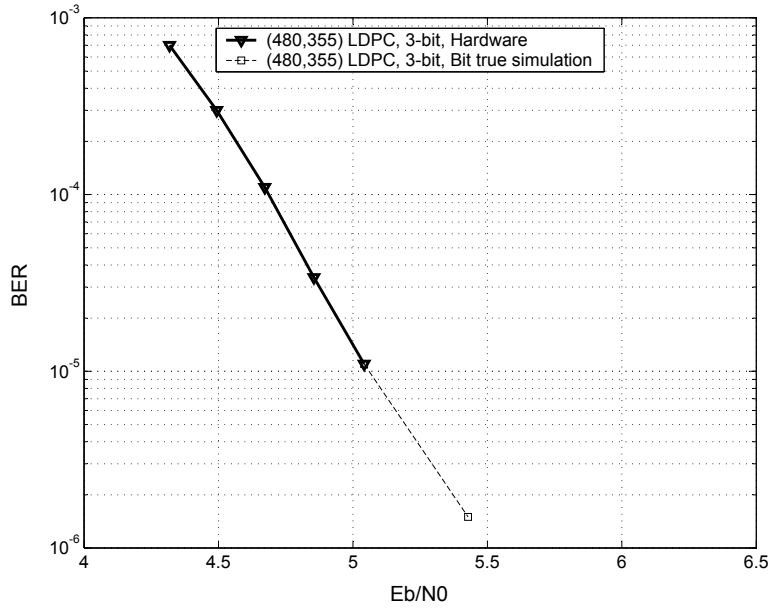


Figure 6.11: FPGA hardware BER results and bit-true software simulation.

The functionality of this decoder was verified by passing input vectors with very low SNR and then comparing the decoder outputs with the expected values from bit-true simulations.

The BER performance of the decoder was measured as follows.

1. For a given SNR additive white Gaussian noise with the desired variance was generated in software and added to all-zero codewords.

2. The LLR value for each noisy bit in the frame was calculated and quantized into $q = 4$ bits.

Table 6.2: (480, 355) RS-based LDPC decoder implementation results.

	This work	L. Yang et al. [61]
FPGA device	Stratix EP1S80	Xilinx XC2V8000
Architecture	Fully-parallel	Partially-parallel
Decoding Algorithm	Approx. min-sum	Min-sum
Logic elements (LUTs)	66,588 (84%)	53,000 (57%)
Shared memory	0	102 × 18kb
Max clock frequency (MHz)	65	100
Decoder throughput (Mbps)	610	up to 40
Code length	480	9000
Code type	(4,15)-regular RS-based	multi-rate
Iterations per frame	17	24
Wordlength (bits)	3	-

3. The quantized LLRs were passed into the hardware decoder through the internal memory modules available on the FPGA device.

4. The decoder outputs were compared against the desired all-zero codewords to produce an error count.

Fig. 6.11 shows the measured BER performance from decoder hardware as well as the bit-true simulation. The decoder operates at a clock frequency of 65 MHz and performs 17 iterations per frame. Using the block-interlacing technique and a word length of 3 bits, each iteration effectively takes 3 clock cycles to complete which results in 610 Mbps total throughput. Table 6.2 summarizes the FPGA implementation results and compares them with the recently-published FPGA decoder in [61].

6.4.2 A 0.13-μm CMOS 660-bit bit-serial decoder

The second decoder is based on a 660-bit (4,15)-regular LDPC code which was constructed using a progressive edge-growth (PEG) algorithm [62] to minimize the number of short cycles in the code's Tanner graph. The block length was limited to 660 by the silicon area available for prototyping. The decoder employs the approximate

min-sum algorithm as described in Section 6.3.1 with 4-bit quantized LLR messages. It also employs the block-interlacing scheme of Chapter 4.

The decoder was fabricated in a 0.13-μm CMOS8M process and was tested using an Agilent 93000 high-speed digital tester. The decoder block diagram is shown in Fig. 6.12. The die photo is also shown in Fig. 6.13. The 4-bit channel LLR values are received via 44 input pins and the decoded outputs are read out via 44 output pins. With a 1.2-V supply, the measured maximum clock frequency is 300 MHz. The decoder performs 15 decoding iterations per frame (beyond which decoding performance increases negligibly) resulting in a total encoded data throughput of 3.3 Gbps. This throughput could be further increased by reducing the number of iterations, at the cost of slightly reduced decoding performance.

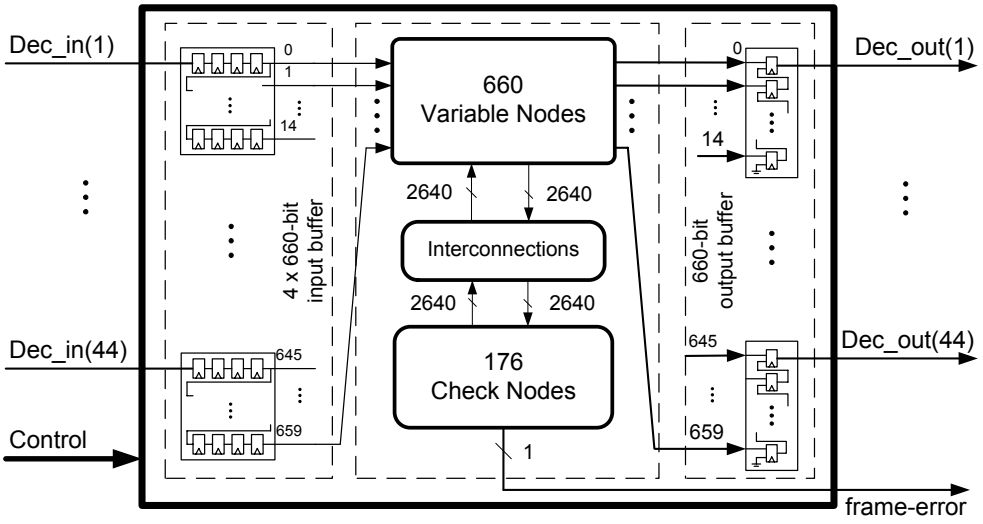


Figure 6.12: The top-level block diagram for the (660, 484) LDPC decoder.

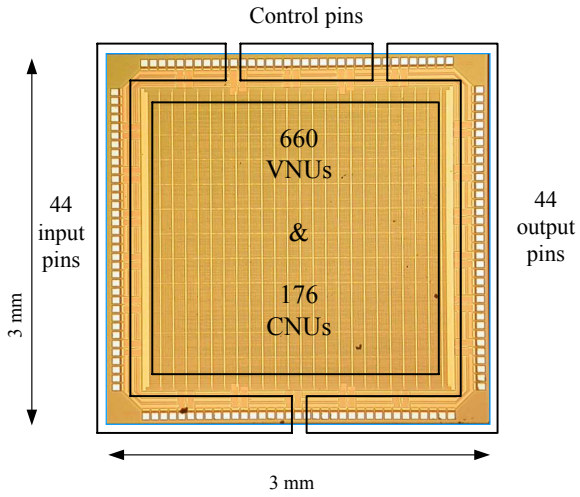


Figure 6.13: Die photo of the (660, 484) LDPC decoder.

The functionality and BER tests for this decoder were performed using a procedure similar to Section 6.4.1 with the exception that the decoder was interfaced through the Agilent 93000 high-speed tester. Fig. 6.15 shows the measured BER performance obtained from decoding 10^7 frames.

Fig. 6.15 also shows the decoder's power consumption with a 1.2-V supply and 300-MHz clock frequency as a function of input SNR. Unlike the BER test in which all-zero codewords were used, for the power measurements the decoder was run with randomly-generated codewords. This ensures that the toggling rates in the internal signals are realistic. During the decoder measurements it was observed that the measured power was about 2X higher than the power consumption estimated by the CAD tools. We believe this is mainly because of the inaccurate RC extraction and to a second degree the inaccurate estimation of activity factors in the internal nodes.

The plot in Fig. 6.15 show that the power consumption peaks at SNRs where the iterative decoder struggles to converge, thus resulting in a slightly higher activity factor. However, the correlation between power consumption and input SNR is lower than in conventional fully-parallel decoders because block-interlacing and bit-serial message-passing techniques tend to maintain high switching activity on all nodes, even at high SNRs.

Measurements show that approximately 20% of the total power dissipation is due to the clock tree. The static power consumption due to leakage current accounts for

less than 1% of the total decoder power. The rest of the power is due to the logic in the CNUs, VUNs and input/output buffers. Since the same supply was used for all the core logic, further breakdown on measured power was not feasible. Fig. 6.14(a) and 6.14(b) show the area breakdown of the decoder core logic for different cell types and different module types, respectively, from the post-synthesis area reports.

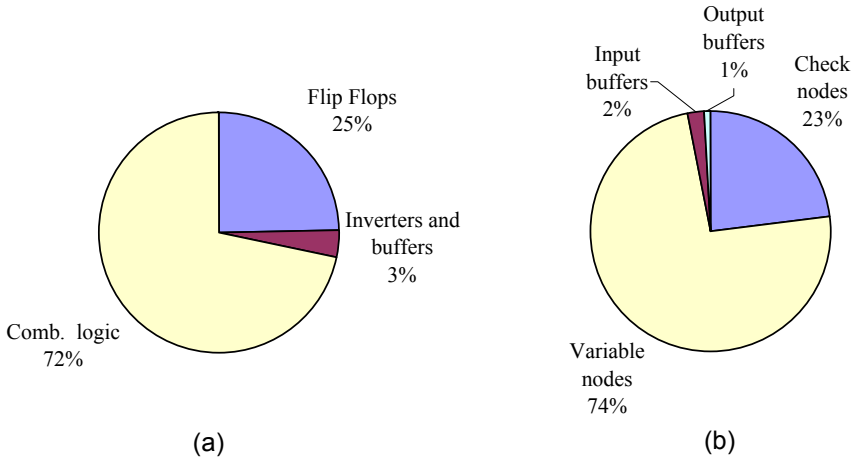


Figure 6.14: Gate area breakdown for (a) cell types, (b) module types.

Fig. 6.16 shows the effect of scaling the decoder's supply voltage on its maximum clock frequency and the corresponding power consumption. The dotted lines are the predicted values based on the MOS square-law equation with $V_t = 300mV$. It can be seen that the measured results closely follow the extrapolated results both for maximum operating frequency and for the power consumption. The plot suggests that for lower-throughput applications, energy efficiency can be improved using the same bit-serial decoder by operating at a lower supply voltage [52]. For example, with a 0.6-V supply, the design has a measured energy consumption of 0.148 nJ/bit – better than previously reported designs specifically targeting low power [16].

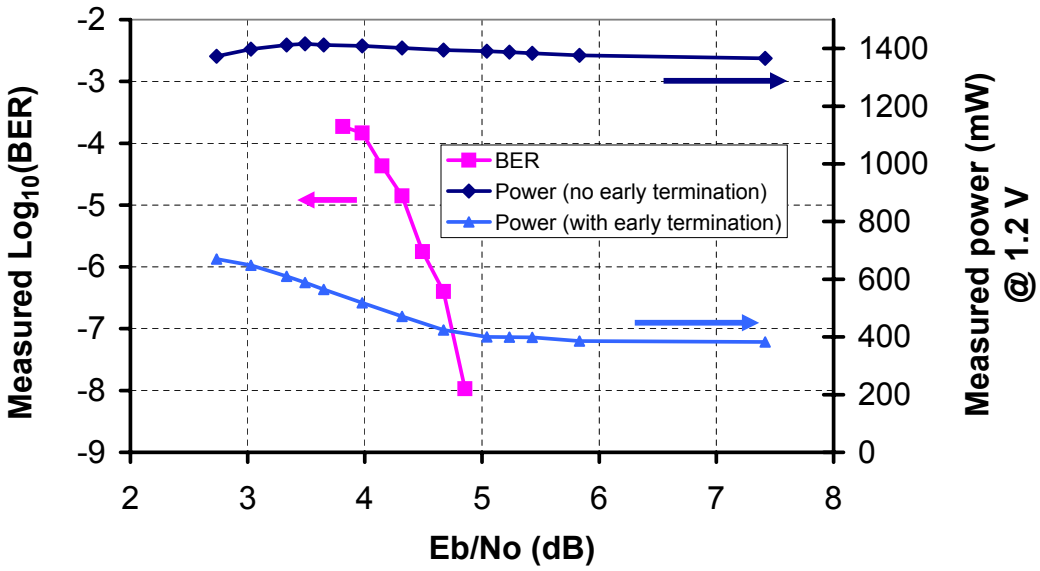


Figure 6.15: Measured power and BER for the fabricated 660-bit decoder.

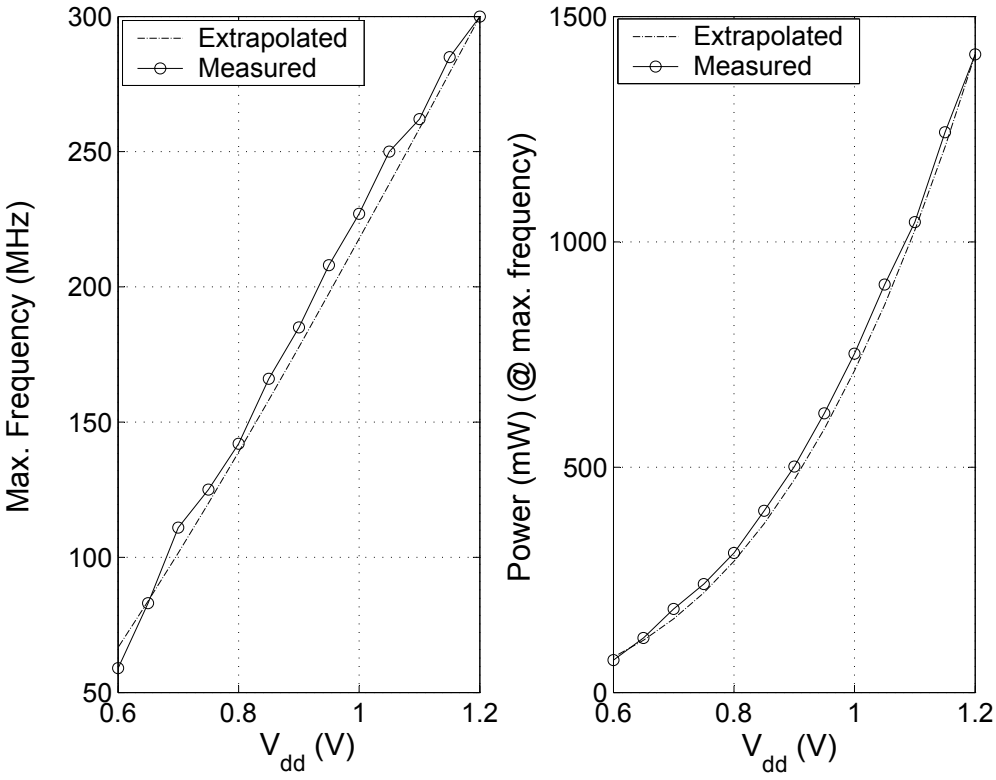


Figure 6.16: Maximum operating frequency and the corresponding power dissipation vs. supply voltage.

Table 6.3: Characteristics summary and measured results.

Process	0.13-μm CMOS			
Architecture	Fully parallel			
LDPC Code	(4, 15)-regular 660-bit			
Code rate	0.74			
Decoding algorithm	Modified min-sum			
Core area (mm^2)	7.3			
Total area (mm^2)	9			
Gate count	690 k			
Core area utilization	72%			
Iterations per frame	15			
Message word length	4 bits			
Package	QFP160			
Supply	1.2 V		0.6 V	
Maximum frequency (MHz)	300		59	
Total throughput (Mbps)	3300		648	
Information throughput (Mbps)	2440		480	
Early termination	No	Yes	No	Yes
Power @ (E_b/N_0=4dB) (mW)	1408	518	72	26.5
Power @ (E_b/N_0=5.5dB) (mW)	1383	398	71	20.5
Energy consumption @ (E_b/N_0=4dB) (pJ/bit/iter)	28.5	10.4	7.4	2.7
Energy consumption @ (E_b/N_0=5.5dB) (pJ/bit/iter)	27.9	8.0	7.3	2.1

Although a maximum of 15 iterations per frame is required for excellent low BER performance, the vast majority of frames need only few decoding iterations to be correctly decoded. As a result, a significant power saving can be achieved by detecting early convergence in the decoder and turning it off for the remaining iterations as described in Chapter 5. The early termination feature is not included in the currently fabricated decoder but, synthesis results show that it would add less than 1% overhead in logic area while providing a large power saving as shown in Fig. 6.15. The values in this graph are obtained from laboratory power measurements of the prototype decoder (without early termination) combined with simulation results of the average required number of iterations per frame at different input SNRs. The plot shows that for the 660-bit LDPC decoder, early termination results in more than 60% power savings at low input SNRs and more than 70% savings at high input SNRs. There is no change in the decoder's BER performance. Table 6.3 summarizes the characteristics of the fabricated decoder.

Table 6.4: Comparison with other works.

	[15]	[63]	[35]	[38]
CMOS Process	0.16-μm	0.18-μm	0.18-μm	0.18-μm
Fully parallel	yes	no	no	yes
Bit serial	no	no	no	yes
Code type	irregular	irregular	regular	regular
Code length	1024	600	2048	256
Code rate	0.5	0.75	programmable	0.5
Total area (mm^2)	52.5 (18.1)[a]	21.9 (12.6)[a]	14.3 (2.4)[a]	10.8 (14.5)[a]
Iter. per frame	64	8	10	32
Supply voltage (V)	1.5	1.8	1.8	1.8
Max. frequency (MHz)	64	80	125	250
Total throughput (Mbps)	1000	640	640	500
Info. throughput (Mbps)	500	480	320	250
Power (mW)	690	192[b]	787	1260 [c]
Energy (pJ/bit/iter)	10.9 (5.8)[d]	37.5 (14.1)[d]	123 (46.3)[d]	118 (44.5)[d]

[a] Scaled linearly to 660-bit code length and quadratically to 0.13-μm process
[b] The power consumption due to clock tree is not included
[c] At 166.7 MHz clock frequency
[d] Scaled cubically to the 0.13-μm process

Another bit-serial LDPC decoder has been independently developed and reported in [38]. It has a pipelined and block-interlaced architecture similar to the bit-serial architecture presented in this thesis. This decoder performs 32 iterations of min-sum decoding on a (256, 128) regular-(3, 6) LDPC code. It is fabricated in a 0.18-μm CMOS process and has a maximum clock frequency of 250 MHz, corresponding to a total throughput of 500 Mbps, or a decoded information throughput of 250 Mbps. When operating at 166.7 MHz, the decoder core is reported to consume 1260 mW from a 1.8-V supply.

Table 6.4 lists the measurement results from LDPC decoders reported in [63, 35, 15,]. The power and throughput performance comparison between these works is shown in Fig. 6.17. To take into account the varying number of iterations per frame and the different code rates in the different decoders, the throughput on the vertical axis is the information throughput normalized to a maximum of 15 iterations per frame, which is used in our decoder. The horizontal axis is the energy consumption of the decoder in pJ per bit per iteration. This value is obtained by dividing the

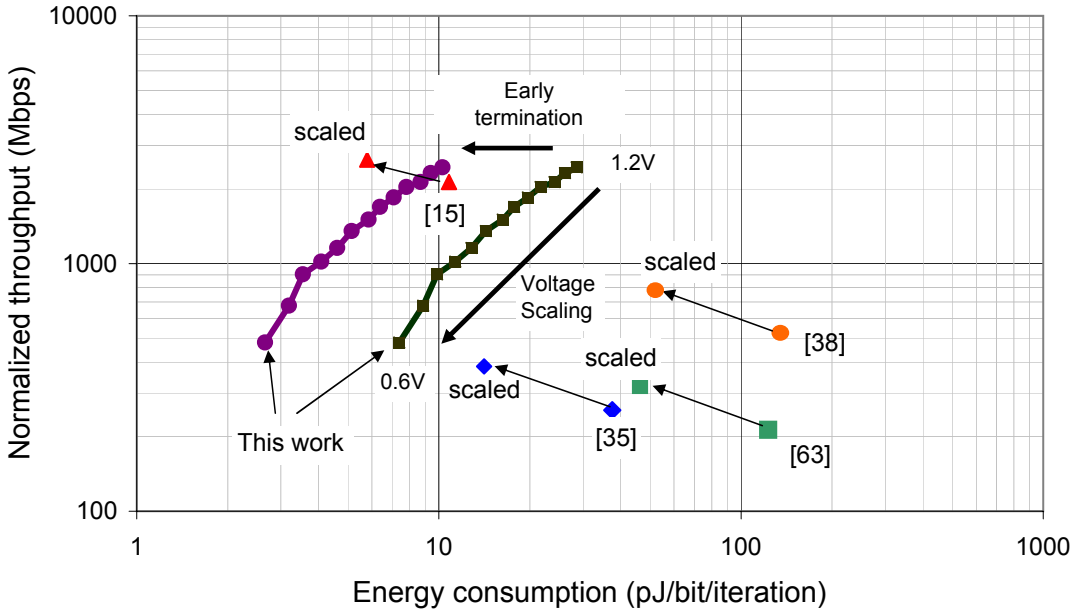


Figure 6.17: Comparison with other works. The effect of early shut-down and supply voltage scaling on power consumption is illustrated.

decoder power consumption by the total decoder throughput and the number of iterations per frame. For comparison purposes, we have also included scaled values for area, throughput and energy consumption in Table 6.4.2 and Fig. 6.17. The area entries in the brackets in Table 6.4 are scaled down quadratically to a 0.13-μm CMOS process and also scaled linearly to a block length of 660 bits. The throughputs and energy efficiencies are scaled linearly and cubically to 0.13-μm CMOS process, respectively ([64], Chapter 16). The comparison graph confirms that fully-parallel decoders provide better energy efficiency and decoding throughput.

The high energy efficiency in [15] can be attributed to its high level of parallelism as predicted in our analysis in Section 5.1.2. It can also be explained by the fact that even though the decoder performs 64 iterations on each block, the vast majority

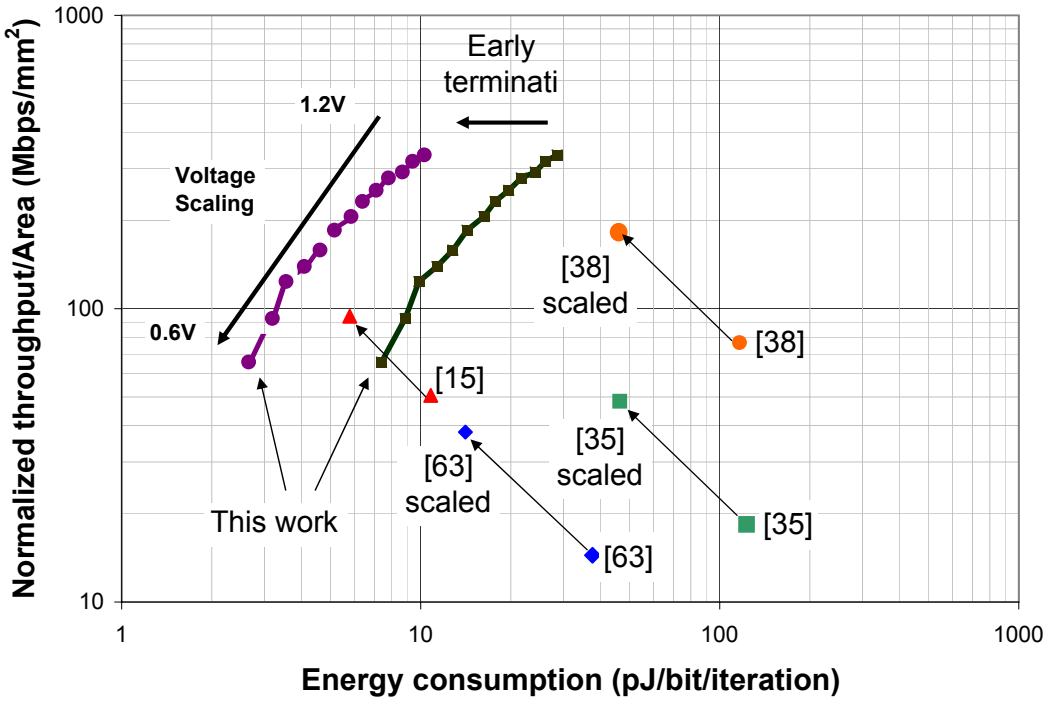


Figure 6.18: Comparison with other works with decoder area also reflected on the vertical axis.

of blocks converge in the first few iterations, resulting in minimal switching activity for the remaining iterations. This is in contrast with the bit-serial block-interlaced decoder presented in our work where the switching activity does not scale down with decoder convergence unless an early termination method is applied. Finally, the average variable node degree in [15] is 3.25 compared to an average degree of 4 in our decoder. For two decoders with the same code length and the same code rate, the decoder with lower average node degree computes fewer messages in each iteration, and hence, consumes less power.

One important dimension which is missing from Fig. 6.17 is the decoder total silicon area and its routing complexity. For example, although the fully-parallel decoder in [15] has good power and throughput performance, its large area makes it

very costly in practice. The bit-serial fully-parallel scheme demonstrated in this work combined with the early termination scheme reduces routing complexity and area while maintaining the throughput and energy efficiency advantages of fully-parallel decoders. Compared to conventional fully-parallel decoders, the logic area is reduced in bit-serial fully-parallel decoders because only 1-bit partial results are generated in each clock cycle. In addition, the reduced routing congestion allows for higher area utilization. This can be observed from the 52.5 mm^2 total area (18.1 mm^2, if scaled for process and code length) with about 50% area utilization in [15] compared to the 9mm^2 total area with 72% area utilization in our design. This comparison is also demonstrated in Fig. 6.18 which is similar to Fig. 6.18 except the throughput on vertical axis is now normalized with respect to the core area.

With the power reduction achievable by early termination, the decoder consumes only 10.4 pJ/bit/iteration from a 1.2V supply voltage and has an encoded throughput of 3.3 Gbps. The projected lines in the graph show that even further power reductions are achievable if supply voltage scaling is combined with early termination. A minimum of 2.7 pJ/bit/iteration is predicted with a 0.6-V supply voltage operating at 59 MHz clock frequency and providing 648 Mbps encoded throughput. These energy efficiency results even compare favorably with analog decoders which are aimed for energy efficiency. For example, the analog LDPC decoder reported in [16] consumes 0.83 nJ/bit and has a throughput of only 6 Mbps. With 15 iterations per frame the digital decoder presented here consume 45% less energy per bit with 500x greater throughput, even without early termination.

Using the same bit-serial architecture, a decoder was synthesized for the (6, 32)-regular (2048, 1723) LDPC code in the 10GBase-T 10-Gbps Ethernet standard. Again, q=4 bit quantization was used for the LLR messages. In this decoder, the maximum number of iterations per frame is eight since our simulations indicate that additional iterations provide less than 0.1-dB performance improvement.

Synthesis in a 90-nm CMOS library using Synopsys Design Compiler results in 9.8-mm^2 logic area (2.23M equivalent minimum size NAND gates) and 250 MHz maximum clock frequency corresponding to 16-Gbps maximum decoding throughput. This throughput is significantly higher than that required by the 10GBase-T standard.

7 Conclusions and Future Work

7.1 Summary

This work investigated VLSI architectures for LDPC decoders with multi-Gbps throughput and high energy efficiency.

To reduce the node-to-node communication complexity, which is a major challenge in most LDPC decoders, half- and full-broadcasting techniques were proposed. It was shown that half-broadcasting provides a better trade-off between node-to-node communication complexity and logic overhead in fully-parallel decoders. Half-broadcasting resulted in 26% global wirelength reduction in fully-parallel decoders and 24% memory access reduction in partially-parallel decoders [39].

A block interlacing scheme was described that maximizes logic utilization and increases the decoder throughput compared with conventional schemes. This method requires no change in the structure of the code or the decoding algorithm. This technique was demonstrated in two fully-parallel LDPC decoder designs. Post-layout simulations show that the throughput was improved by 60% and 71% at the cost of only 5.5% and 9.5% more gates, respectively [65].

Two techniques were discussed to improve the energy efficiency of LDPC decoders. First, an analysis was given on how increased hardware parallelism coupled with a reduced supply voltage is a particularly effective technique to reduce the power consumption of LDPC decoders due to the algorithm's inherent parallelism. Second, an efficient early termination scheme was proposed to further reduce the power consumption [66].

A bit-serial architecture was presented to reduce the routing complexity in fully-parallel LDPC decoders. A new approximation to the check node update function in min-sum decoding algorithm was also proposed that reduces the area of the CNUs by more than 40% compared with conventional min-sum decoding with only a 0.1 dB performance penalty at BER=10^{-6} [55].

The proposed techniques were demonstrated in two bit-serial LDPC decoder implementations. First, a 610-Mbps bit-serial fully-parallel (480, 355) LDPC decoder on a single Altera Stratix EP1S80 device was presented. To our knowledge, this is the fastest FPGA-based LDPC decoder reported in the literature [55]. Second, a fabricated 0.13-μm CMOS bit-serial (660, 484) LDPC decoder was reported. The decoder has a 300 MHz maximum clock frequency and a 3.3 Gbps throughput with a nominal 1.2-V supply and performs within 3 dB of the Shannon limit at a BER of 10^{-5}. With more than 60% power saving achievable by early termination, the decoder consumes 10.4 pJ/bit/iteration at E_b/N_0=4 dB. Coupling early termination with supply voltage scaling results in an even lower energy consumption of 2.7 pJ/bit/iteration with 648 Mbps decoding throughput. These energy efficiency results even compare favorably with analog decoders which are aimed for energy efficiency [67].

7.2 Conclusions

The experience obtained through this research suggests that different layers of the design space need to be investigated in order to achieve a high-throughput and energy efficient LDPC decoder.

First, for a given LDPC code, a decoding algorithm has to be selected that provides the target BER with minimal hardware and time complexity. For hardware complexity, the most important aspects are the computational complexity of the update functions in the variable and check nodes and also the robustness of the algorithm against quantization effects. With regards to the time complexity, the main issue is the algorithm's convergence speed in terms of the average and maximum number of required decoding iterations per frame. As shown in this work, the choice of the maximum number of iterations per frame and the choice of message word length directly affect the decoding throughput and the decoder's energy efficiency. In particular, it was shown that the approximate min-sum decoding as described in Chapter 6 provides a good trade-off between the BER performance and hardware complexity.

Second, depending on the code length, the required decoding throughput and the available silicon area, the designer needs to select an appropriate decoder architecture. The decoders presented in this work suggest that for the 2048-bit code specified in 10GBase-T standard, a fully-parallel decoder in a 90-nm CMOS process has a core area of less than 10 mm^2 core area and provides the required decoding throughput.

The routing complexity of the fully-parallel architecture can be reduced with the bit-serial scheme described in this work. Block interlacing will be also an effective technique to increase decoding throughput with minimal hardware overhead. In addition, such a decoder would also benefit from the power efficiency advantages of the fully-parallel decoders as described in Chapter 5.

Meanwhile, for codes of a considerably longer length, such as the 16k and 64k-bit codes specified in Digital Video Broadcasting standard, a partially-parallel decoder architecture is preferred. This is because a fully-parallel implementation would require an impractical amount of logic and also because the target throughput is achievable with a much lower level of parallelism (e.g., 135 Mbps throughput with 360 parallel processors in [36]). The early termination scheme should be applied both in partially parallel and fully-parallel decoders since it improves the energy efficiency of either architectures with minimal overhead and no effect on BER performance.

7.3 Future Work

7.3.1 Technology scaling

In the 0.13-μm CMOS decoder presented in this work it was observed that the static power due to the leakage current constitutes only less than 1% of the total power. As a result, in the power analysis performed in Chapter 5, the main focus was on the dynamic power consumption. However, it is well know that as the CMOS process technology scales to smaller geometries, the leakage power becomes a more dominant portion of the total power. Consequently, architectural and physical level leakage management techniques such as power-gating and employing multiple voltage domains and multiple threshold devices will be needed.

At the same time, CMOS technology scaling will facilitate direct mapping of longer LDPC codes into hardware, resulting in fully-parallel decoders with higher throughput and better BER performance. Furthermore, it is known that the ratio of global interconnect delays to the logic delays will grow as the CMOS technology scales down [68]. Since LDPC decoders intrinsically have a large number of global connections between VNUs and CNU, addressing the global wire delay problem will be a key to successfully increasing code lengths of fully-parallel LDPC decoders in future technologies.

7.3.2 Multi-rate and multi-length decoders

The fully-parallel decoder architectures presented in this work directly map the LDPC code graph into hardware. This implies that changing the LDPC code in general requires re-synthesizing the decoder based on the new parity check matrix. Although this is acceptable for applications such as 10GBase-T which specify only one fixed code in the standard, other applications such as WiMAX need to be able to decode multiple LDPC codes with different lengths and rates, depending on the channel condition and desired BER.

Developing flexible fully-parallel LDPC decoders that can support multiple codes needs further investigation. One possible direction is to implement the hardware decoder for a Tanner graph that contains all the individual codes as its subgraphs. In such a decoder, different nodes and edges need to be activated or deactivated depending on the specific code. In the case of WiMAX standard for example, there are multiple codes specified in the standard with rates ranging from 1/2 to 5/6 and lengths ranging from 576 to 2304. It can be seen that all these codes can be generated by puncturing and shortening the rate-1/2 2304-bit code (i.e., by removing some rows and columns from the parity check matrix of the rate-1/2 2304-bit code). As a result, a fully-parallel LDPC decoder compliant with WiMAX standard can be realized by implementing this code and adding the control logic to disable some VNUs, CNUs and edges depending on the target LDPC code.

7.3.3 Custom layout for sub-blocks

A fully-parallel LDPC decoder contains a large number of identical sub-blocks (i.e., check nodes and variable nodes). One possible alternative to the digital standard cell design flow is to develop an optimized CNU and VNU with custom logic circuit and layout, and then instantiate them multiple times in the top-level of the decoder design.

Although the single CNUs and VNUs can be developed manually, the large size of decoder still necessitates the use of automated CAD tools for top-level design tasks such as placement, clock tree generation, power routing and global routing. In addition, although the CNUs and VNUs are similar across the chip, they will have different output loads to drive due to the difference in the length of the global check-to-variable and variable-to-check wires. This again requires automated timing

analysis and optimization. To integrate the custom designed sub-blocks with the standard digital design flow, one needs to characterize them in the proper timing and physical library formats usable by the CAD tools.

Another similar design approach is to customize not a complete VNU or CNU but only portions of the nodes that are instantiated frequently in each VNU or CNU. For example, the `Next-state-logic` block in the bit-serial CNU design of Fig. 6.4 is a purely combinational logic block that is repeated d_c times in each CNU. Therefore, a more manageable solution might be to create a custom layout with optimized timing, area and power for this block and then instantiate it as a standard cell in the top-level digital design flow.

7.3.4 Hardware-based Gaussian noise generation

The BER measurements in Chapter 6 were reported only down to 10^{-8}. This was because the speed of the frame tests was limited by the software-based Gaussian noise generation and the required data transfer between the workstation and the test board. This rate could increase by several orders of magnitude if a high-speed noise generation module (such as the FPGA-based Gaussian noise generator in [69]) was integrated with the test setup providing an incoming frame rate closer to the maximum potential decoder throughput. A hardware-based Gaussian noise generator along with an FPGA decoder can also be used as a fast LDPC code simulation engine for evaluating and exploring the performance of different LDPC codes at BER's lower than 10^{-9} or 10^{-10} [70].

7.3.5 Adjustable message word length

For the bit-serial decoder architecture reported in this work, the decoding throughput is directly affected by the value of message word length, q. This is because each decoding iteration consists of $2q$ clock cycles: q cycles for the CNU update and q cycles for the VNU update.

One advantage of bit-serial decoding is that it allows for efficient switching between different word lengths on the fly. This property can be utilized to improve the decoder throughput and power dissipation by adjusting the word length of the LLR messages according to the channel conditions and the target BER. By adjusting the value of q based on the input SNR and choosing the smallest q needed for the target BER

one can increase the throughput during the high input SNR periods of operation as less number of clock cycles are needed per iteration. Alternatively, one can keep the throughput constant in those periods but reduce the operating frequency and the supply voltage, hence reduce the power consumption.

Another possibility is to adopt a decoding strategy similar to the gear-shift decoding [59] where the decoding parameters and update rules change as decoding iterations proceed for each frame. In this technique, the first few decoding iterations use more powerful and computationally-intensive update rules such as Sum-Product to be able to correct more errors. As the number of remaining errors decreases, the later iterations use simpler update rules such hard-decision message passing or bit-flipping resulting in reduced overall decoding complexity.

In the context of the bit-serial decoders in this thesis, the gear-shift decoding can be realized by keeping the update rules unchanged but reduce the word length, q, as the iterations proceed for each frame. This technique can potentially reduce the total number of clock cycles for decoding each frame, hence increase the throughput, or alternatively, reduce the power consumption for a fixed throughput.

A Parity-Check Matrix for FPGA-Based Decoder

Table A.1 specifies the RS-based LDPC code used for the the FPGA-based decoder reported in Section 6.4.1. The first entry in each line of the table is the row number of the H matrix and the rest of the entries indicate the indices of the non-zero elements in that row.

Table A.1: The (4, 15)-regular (480, 355) LDPC code.

1	1	33	65	97	129	161	193	225	257	289	321	353	385	417	449
2	2	56	70	99	157	167	221	244	280	318	345	374	398	424	458
3	3	44	75	101	158	173	222	228	268	320	342	368	411	431	467
4	4	61	80	103	130	171	194	241	285	291	334	379	408	426	476
5	5	55	85	105	160	185	224	231	279	316	336	383	402	445	450
6	6	34	82	107	132	191	196	246	258	295	344	364	413	444	457
7	7	62	95	109	131	181	195	230	286	293	347	370	396	435	468
8	8	43	92	111	159	179	223	247	267	314	323	357	391	438	475
9	9	42	78	113	156	182	220	237	266	308	351	378	392	446	451
10	10	63	73	115	136	180	200	256	287	303	327	365	395	443	460
11	11	35	72	117	135	186	199	240	259	301	332	375	414	436	465
12	12	54	67	119	155	192	219	253	278	306	340	356	401	437	474
13	13	64	90	121	133	174	197	235	288	297	338	360	407	418	452
14	14	41	93	123	153	172	217	250	265	310	330	371	412	423	459
15	15	53	84	125	154	162	218	234	277	312	325	361	397	432	466
16	16	36	87	127	134	168	198	251	260	299	349	382	386	425	473
17	17	51	91	102	148	176	212	249	275	292	346	376	399	448	453
18	18	38	96	104	144	170	208	236	262	319	322	355	388	441	462
19	19	58	81	98	143	164	207	252	282	317	333	377	405	434	471

20	20	47	86	100	147	166	211	233	271	290	341	366	410	439	480
21	21	37	79	110	141	184	205	255	261	313	343	362	416	420	454
22	22	52	76	112	145	178	209	238	276	294	335	381	403	421	461
23	23	48	69	106	146	188	210	254	272	296	324	359	390	430	472
24	24	57	66	108	142	190	206	239	281	315	348	372	393	427	479
25	25	60	88	118	137	187	201	245	284	305	328	367	394	419	455
26	26	45	83	120	149	189	213	232	269	302	352	380	389	422	464
27	27	49	94	114	150	183	214	248	273	304	339	354	404	429	469
28	28	40	89	116	138	177	202	229	264	307	331	373	415	428	478
29	29	46	68	126	152	163	216	243	270	300	329	369	409	447	456
30	30	59	71	128	140	165	204	226	283	311	337	358	406	442	463
31	31	39	74	122	139	175	203	242	263	309	350	384	387	433	470
32	32	50	77	124	151	169	215	227	274	298	326	363	400	440	477
33	23	57	76	104	149	180	197	230	282	298	344	362	410	442	458
34	24	48	79	102	137	182	217	247	271	309	336	381	405	447	449
35	21	52	66	100	138	192	218	231	275	311	323	359	388	440	476
36	22	37	69	98	150	186	198	246	262	300	347	372	399	433	467
37	19	47	96	112	140	172	220	228	272	307	345	376	393	422	457
38	20	58	91	110	152	174	200	241	281	304	321	355	390	419	450
39	17	38	86	108	151	168	199	225	261	302	334	377	403	428	475
40	18	51	81	106	139	162	219	244	276	305	342	366	416	429	468
41	31	50	71	120	144	167	224	234	273	315	330	369	415	421	460
42	32	39	68	118	148	161	196	251	264	296	338	358	404	420	451
43	29	59	77	116	147	171	195	235	284	294	349	384	389	427	474
44	30	46	74	114	143	173	223	250	269	313	325	363	394	430	465
45	27	40	83	128	145	191	193	240	263	290	327	367	400	441	459
46	28	49	88	126	141	185	221	253	274	317	351	380	387	448	452
47	25	45	89	124	142	179	222	237	270	319	340	354	406	439	473
48	26	60	94	122	146	181	194	256	283	292	332	373	409	434	466
49	7	43	82	99	136	189	216	254	268	299	335	383	408	423	462
50	8	62	85	97	156	187	204	239	285	312	343	364	411	418	453
51	5	34	92	103	155	177	203	255	257	310	348	370	398	425	480
52	6	55	95	101	135	183	215	238	280	297	324	357	385	432	471
53	3	61	70	107	153	165	201	252	286	306	322	353	391	443	461

54	4	44	65	105	133	163	213	233	267	301	346	374	396	446	454
55	1	56	80	111	134	169	214	249	279	303	341	368	413	437	479
56	2	33	75	109	154	175	202	236	258	308	333	379	402	436	472
57	15	36	93	115	157	170	205	242	259	314	337	360	401	444	464
58	16	53	90	113	129	176	209	227	278	293	329	371	414	445	455
59	13	41	87	119	130	166	210	243	266	295	326	361	395	438	478
60	14	64	84	117	158	164	206	226	287	316	350	382	392	435	469
61	11	54	73	123	132	178	212	248	277	291	352	378	386	424	463
62	12	35	78	121	160	184	208	229	260	320	328	365	397	417	456
63	9	63	67	127	159	190	207	245	288	318	331	375	412	426	477
64	10	42	72	125	131	188	211	232	265	289	339	356	407	431	470
65	10	54	87	111	142	164	201	235	280	307	332	371	408	440	467
66	9	35	84	109	146	166	213	250	257	304	340	360	411	433	476
67	12	63	93	107	145	176	214	234	285	302	351	382	398	442	449
68	11	42	90	105	141	170	202	251	268	305	327	361	385	447	458
69	14	36	67	103	147	188	216	237	258	298	325	365	391	428	468
70	13	53	72	101	143	190	204	256	279	309	349	378	396	429	475
71	16	41	73	99	144	184	203	240	267	311	338	356	413	422	450
72	15	64	78	97	148	178	215	253	286	300	330	375	402	419	457
73	2	61	92	127	151	183	212	231	287	290	342	364	401	427	465
74	1	44	95	125	139	177	208	246	266	317	334	383	414	430	474
75	4	56	82	123	140	187	207	230	278	319	321	357	395	421	451
76	3	33	85	121	152	189	211	247	259	292	345	370	392	420	460
77	6	43	80	119	138	175	205	225	265	315	347	374	386	439	466
78	5	62	75	117	150	169	209	244	288	296	323	353	397	434	473
79	8	34	70	115	149	163	210	228	260	294	336	379	412	441	452
80	7	55	65	113	137	165	206	241	277	313	344	368	407	448	459
81	26	40	77	108	159	173	220	243	262	306	339	358	410	425	471
82	25	49	74	106	131	171	200	226	275	301	331	369	405	432	480
83	28	45	71	112	132	161	199	242	271	303	328	363	388	423	453
84	27	60	68	110	160	167	219	227	282	308	352	384	399	418	462
85	30	50	89	100	130	181	197	245	276	299	350	380	393	437	472
86	29	39	94	98	158	179	217	232	261	312	326	367	390	436	479
87	32	59	83	104	157	185	218	248	281	310	329	373	403	443	454

88	31	46	88	102	129	191	198	229	272	297	337	354	416	446	461
89	18	47	66	124	134	186	193	255	269	291	333	381	415	438	469
90	17	58	69	122	154	192	221	238	284	320	341	362	404	435	478
91	20	38	76	128	153	182	222	254	264	318	346	372	389	444	455
92	19	51	79	126	133	180	194	239	273	289	322	359	394	445	464
93	22	57	86	116	155	162	224	249	283	314	324	355	400	426	470
94	21	48	81	114	135	168	196	236	270	293	348	376	387	431	477
95	24	52	96	120	136	174	195	252	274	295	343	366	406	424	456
96	23	37	91	118	156	172	223	233	263	316	335	377	409	417	463
97	32	46	94	106	154	177	205	240	271	316	349	380	399	431	476
98	31	59	89	108	134	183	209	253	282	295	325	367	388	426	467
99	30	39	88	110	133	189	210	237	262	293	330	373	405	417	458
100	29	50	83	112	153	187	206	256	275	314	338	354	410	424	449
101	28	60	74	98	135	169	212	234	281	289	340	358	416	435	475
102	27	45	77	100	155	175	208	251	272	318	332	369	403	438	468
103	26	49	68	102	156	165	207	235	276	320	327	363	390	445	457
104	25	40	71	104	136	163	211	250	261	291	351	384	393	444	450
105	24	37	81	122	131	166	216	228	264	297	323	355	394	436	474
106	23	52	86	124	159	164	204	241	273	310	347	376	389	437	465
107	22	48	91	126	160	170	203	225	269	312	344	366	404	446	460
108	21	57	96	128	132	176	215	244	284	299	336	377	415	443	451
109	20	51	69	114	158	190	201	230	274	308	334	381	409	432	473
110	19	38	66	116	130	188	213	247	263	303	342	362	406	425	466
111	18	58	79	118	129	178	214	231	283	301	345	372	387	418	459
112	17	47	76	120	157	184	202	246	270	306	321	359	400	423	452
113	16	64	72	109	139	192	224	248	285	313	326	365	385	434	480
114	15	41	67	111	151	186	196	229	268	294	350	378	398	439	471
115	14	53	78	105	152	180	195	245	280	296	337	356	411	448	462
116	13	36	73	107	140	182	223	232	257	315	329	375	408	441	453
117	12	42	84	101	150	168	193	242	267	292	331	371	402	430	479
118	11	63	87	103	138	162	221	227	286	319	339	360	413	427	472
119	10	35	90	97	137	172	222	243	258	317	352	382	396	420	461
120	9	54	93	99	149	174	194	226	279	290	328	361	391	421	454
121	8	55	75	125	146	171	197	252	278	300	348	374	392	429	478

122	7	34	80	127	142	173	217	233	259	311	324	353	395	428	469
123	6	62	65	121	141	167	218	249	287	309	335	379	414	419	464
124	5	43	70	123	145	161	198	236	266	298	343	368	401	422	455
125	4	33	95	117	143	179	220	254	260	305	341	364	407	433	477
126	3	56	92	119	147	181	200	239	277	302	333	383	412	440	470
127	2	44	85	113	148	191	199	255	265	304	322	357	397	447	463
128	1	61	82	115	144	185	219	238	288	307	346	370	386	442	456

B Parity-Check Matrix for 0.13-μm CMOS Decoder

Table B.1 specifies the PEG-based LDPC code used for the the 0.13-μm CMOS decoder reported in Section 6.4.2. The first entry in each line of the table is the row number of the H matrix and the rest of the entries indicate the indices of the non-zero elements in that row.

Table B.1: The (4, 15)-regular (660, 484) LDPC code.

1	1	45	91	135	179	224	270	312	355	401	442	486	530	573	618
2	2	46	92	136	180	225	271	313	356	402	443	487	531	574	619
3	3	47	93	137	178	226	272	314	357	403	444	488	532	575	620
4	4	48	94	138	181	227	273	310	358	404	442	489	533	576	621
5	5	49	92	139	182	228	267	315	359	405	445	490	534	577	622
6	6	50	95	140	183	229	274	316	360	406	446	491	535	578	623
7	7	51	96	141	183	230	275	317	352	407	447	485	536	579	621
8	2	52	97	142	184	226	276	318	361	408	442	492	537	580	624
9	8	53	98	143	185	231	277	319	362	398	448	493	538	581	625
10	9	54	99	125	176	228	275	320	363	409	449	494	532	573	626
11	9	55	100	144	186	232	278	293	364	410	450	495	539	582	627
12	10	52	101	145	187	233	279	313	349	411	447	495	530	583	628
13	11	56	93	143	186	222	236	270	365	412	445	496	540	584	629
14	7	57	92	146	188	224	280	321	366	413	451	497	541	585	630
15	12	58	102	141	189	234	278	320	367	369	452	498	542	586	631
16	13	59	103	147	190	235	281	322	368	414	453	499	533	579	632
17	14	59	104	148	191	236	282	316	369	415	454	500	543	587	630
18	13	60	105	149	192	237	283	323	370	416	455	501	528	578	629
19	15	54	106	146	193	221	284	318	355	417	441	502	544	588	621

20	16	61	107	145	194	238	275	324	355	399	453	503	545	589	622
21	17	62	101	146	185	239	285	325	371	418	456	504	546	590	619
22	18	63	108	150	195	220	286	326	367	406	457	505	530	591	633
23	19	61	104	151	196	240	287	323	372	419	458	506	529	590	634
24	20	49	109	152	197	241	288	327	373	420	459	507	547	592	635
25	19	63	110	146	178	242	289	319	374	421	460	495	548	593	623
26	21	47	111	140	198	243	290	328	375	401	461	508	549	594	636
27	22	64	106	153	199	244	291	329	376	420	453	508	550	587	637
28	19	55	112	153	197	233	281	330	377	418	445	509	551	595	621
29	22	59	90	154	194	227	286	331	359	401	448	510	531	596	638
30	22	65	113	136	200	237	292	312	378	410	462	511	542	580	639
31	23	66	114	155	201	245	293	332	379	421	455	503	552	591	640
32	24	64	115	152	202	246	270	297	371	422	463	512	553	577	641
33	14	67	91	156	201	247	285	333	357	423	464	513	549	592	642
34	25	68	116	157	203	225	286	334	363	424	465	514	554	597	643
35	8	69	116	158	184	248	294	321	380	425	450	488	534	576	644
36	26	70	95	159	199	232	295	323	381	426	459	494	540	586	632
37	12	71	113	151	179	241	296	335	382	422	451	515	555	598	637
38	13	47	88	159	202	249	280	333	383	393	466	502	550	597	640
39	27	53	109	144	204	227	284	328	384	427	447	516	551	599	643
40	15	56	112	136	205	250	287	308	385	426	467	492	556	592	645
41	28	66	88	135	191	239	286	336	353	419	468	491	555	600	646
42	14	72	117	137	206	251	297	337	374	428	469	516	529	601	646
43	29	73	118	160	207	243	282	301	386	429	455	489	541	602	647
44	29	74	91	161	190	242	290	314	373	411	470	487	544	590	617
45	26	53	119	145	182	247	298	338	361	430	443	507	539	601	616
46	22	50	120	162	178	218	283	337	363	431	471	496	557	603	648
47	30	48	100	163	190	239	299	305	383	429	472	511	552	596	620
48	11	75	121	134	207	220	279	315	360	407	460	517	558	580	648
49	1	60	94	164	193	252	300	339	365	403	473	497	559	579	631
50	31	76	122	144	208	229	301	329	354	357	457	484	540	573	634
51	28	76	115	156	196	228	302	319	382	430	472	517	560	602	649
52	15	77	96	147	209	247	258	340	379	404	472	510	541	604	624
53	32	55	97	150	188	253	290	341	385	432	465	489	538	605	646

54	32	78	123	155	193	254	303	331	387	433	450	498	561	606	624
55	33	79	123	157	196	255	274	342	366	386	474	500	540	607	650
56	34	78	108	161	186	256	304	338	381	404	467	518	543	577	620
57	27	78	104	165	200	255	299	314	382	417	455	513	562	608	646
58	35	80	124	156	180	233	273	343	388	415	475	488	555	591	651
59	36	77	111	139	195	236	305	344	378	424	474	491	563	609	652
60	2	71	119	166	210	257	305	334	389	425	453	489	564	610	630
61	37	80	125	140	200	224	276	338	356	428	447	498	560	593	627
62	4	81	126	162	191	254	306	345	379	434	476	519	559	609	629
63	21	75	106	167	211	241	293	333	380	432	473	520	537	611	653
64	31	82	111	135	212	244	304	326	390	414	448	520	542	608	627
65	38	46	126	168	197	222	307	342	391	435	448	486	541	582	644
66	17	83	122	169	198	240	267	310	370	395	460	513	552	612	651
67	39	84	122	170	213	248	305	332	371	405	443	515	565	603	642
68	25	72	114	153	208	258	299	315	362	433	449	521	544	585	627
69	6	71	118	130	214	254	279	346	368	431	477	500	554	595	617
70	16	69	108	140	201	259	303	308	373	396	465	522	550	601	625
71	13	85	123	167	185	259	271	346	360	403	461	523	529	600	639
72	34	66	127	147	177	238	280	345	380	411	465	507	535	613	654
73	17	86	128	152	200	216	298	347	392	434	477	508	566	614	633
74	15	76	116	161	214	260	262	348	393	436	478	512	566	584	653
75	28	58	110	171	213	235	265	349	364	416	459	518	567	589	652
76	37	48	110	155	182	259	297	329	384	437	446	515	538	608	655
77	6	49	93	138	215	261	292	348	372	433	479	510	539	609	619
78	40	87	104	164	180	262	295	315	394	438	452	514	538	598	635
79	41	54	128	171	195	263	289	350	368	439	443	501	564	606	650
80	33	81	90	137	212	248	298	319	391	440	478	524	568	592	628
81	16	58	106	168	215	257	304	340	362	409	480	509	546	575	642
82	5	73	102	133	192	246	308	313	358	421	458	524	569	579	651
83	23	81	129	172	204	260	283	331	343	426	444	520	543	590	626
84	28	65	109	160	184	260	266	324	395	422	464	525	559	573	632
85	35	58	117	142	198	264	306	341	365	402	470	521	550	604	634
86	41	85	115	162	187	245	282	335	376	436	481	506	570	581	655
87	42	72	118	128	180	259	278	325	372	390	482	499	563	613	641

88	24	55	129	138	185	230	307	335	363	439	478	526	562	596	652
89	39	67	127	166	181	244	250	351	366	431	483	501	546	581	638
90	18	52	96	168	206	243	287	347	370	430	484	525	570	615	652
91	43	77	110	173	203	252	285	332	375	441	483	520	530	599	641
92	31	59	130	172	216	264	295	352	389	435	449	488	558	615	622
93	24	79	131	174	211	262	298	352	396	417	475	496	554	586	639
94	35	57	120	155	202	250	261	350	369	416	484	504	565	612	635
95	26	63	105	169	214	234	288	312	376	434	475	499	556	615	656
96	29	57	98	164	197	265	273	337	389	407	458	494	554	578	649
97	30	68	89	134	198	261	269	339	385	440	462	527	547	610	657
98	11	65	132	172	217	253	284	347	373	439	452	505	539	574	638
99	38	83	131	149	218	256	276	327	371	440	457	495	567	609	618
100	5	61	124	131	193	229	307	353	380	428	452	518	548	595	656
101	40	86	133	175	187	221	280	336	386	439	471	485	531	605	639
102	3	51	89	167	182	260	307	311	396	425	459	497	560	612	658
103	10	73	94	167	188	256	264	347	395	409	485	486	548	610	648
104	20	45	108	163	196	221	272	316	397	427	482	493	559	583	658
105	39	51	109	149	187	236	303	343	374	400	476	511	571	614	645
106	30	85	127	148	215	252	302	328	378	437	471	525	534	595	618
107	20	82	117	147	219	232	308	330	391	423	449	491	571	611	647
108	44	69	113	159	181	228	282	318	397	424	468	503	557	599	642
109	17	50	112	174	210	249	302	332	359	427	442	527	545	616	659
110	9	71	107	148	209	246	289	323	388	428	470	492	547	588	659
111	42	53	127	151	220	226	306	312	356	435	484	527	569	611	660
112	38	88	118	139	221	253	309	354	395	411	473	512	532	593	656
113	20	68	125	145	191	250	291	318	384	410	470	485	549	603	660
114	36	80	114	159	205	230	300	346	375	425	460	506	566	605	659
115	27	77	134	170	179	233	303	334	383	423	446	487	532	589	633
116	30	65	131	176	189	247	289	325	398	405	483	497	568	583	649
117	23	74	122	136	220	255	275	345	390	403	466	512	567	594	632
118	26	74	124	165	188	266	296	344	357	421	461	505	558	589	626
119	33	63	133	144	210	261	300	324	368	420	476	509	557	601	638
120	42	81	96	163	202	267	309	321	377	412	480	523	561	599	649
121	3	79	101	150	209	244	309	337	369	400	479	509	536	615	640

122	32	68	95	175	212	237	273	340	392	402	479	493	553	613	630
123	29	62	115	141	177	237	281	326	361	413	473	519	556	604	626
124	10	66	95	176	222	240	294	343	393	432	469	519	551	574	618
125	33	56	107	165	219	245	310	333	396	429	483	494	565	575	619
126	4	76	119	173	206	224	263	309	360	405	471	502	571	596	631
127	7	64	132	171	214	268	272	320	379	438	464	490	568	581	654
128	6	87	134	154	184	245	251	338	398	409	474	490	536	585	647
129	40	69	100	177	217	263	288	342	388	441	481	521	561	575	616
130	34	86	99	178	207	264	271	322	382	438	482	516	553	606	631
131	12	79	114	173	216	223	284	336	399	438	446	518	569	613	629
132	10	84	113	174	204	251	268	351	354	406	461	499	570	582	657
133	39	75	97	173	194	239	295	344	372	391	462	508	572	616	655
134	32	83	102	143	190	251	311	353	375	416	456	514	531	614	650
135	37	62	103	151	186	248	291	313	385	399	466	513	557	602	653
136	3	54	98	169	201	225	296	316	388	413	466	490	545	576	660
137	2	89	91	171	192	227	274	330	383	414	463	517	570	588	637
138	1	86	129	142	157	258	310	349	381	410	468	528	546	602	625
139	44	62	111	154	223	268	274	327	358	415	444	502	572	603	645
140	24	82	105	158	194	265	299	346	366	418	454	492	561	610	623
141	12	60	112	160	195	269	297	354	362	420	458	504	535	605	653
142	14	70	120	157	213	231	238	341	377	408	478	525	542	611	659
143	35	90	100	149	207	268	277	339	387	418	463	522	543	574	643
144	7	88	121	166	219	223	287	335	397	398	433	504	534	586	650
145	31	52	99	143	211	252	291	342	377	415	451	522	562	597	620
146	41	45	102	138	208	249	271	317	378	419	445	500	535	591	647
147	1	75	130	137	177	266	277	340	356	426	456	526	547	577	658
148	19	84	85	135	183	225	293	341	358	400	430	505	563	584	628
149	37	70	126	160	211	258	272	331	355	436	479	511	556	607	657
150	25	87	105	168	181	266	311	327	364	436	467	496	562	593	628
151	43	87	101	132	218	241	301	324	390	419	477	517	569	604	625
152	18	83	129	148	199	223	300	314	394	434	451	507	553	584	622
153	8	46	117	150	179	229	283	320	389	437	482	514	572	617	636
154	40	47	119	153	213	256	294	325	365	406	475	498	572	612	643
155	41	70	121	158	204	242	288	344	400	440	456	523	555	597	641

156	4	61	121	152	205	235	290	334	394	402	454	510	563	578	637
157	18	80	93	174	192	238	296	321	387	429	469	526	567	600	655
158	27	73	120	169	212	230	257	322	359	412	472	519	549	587	657
159	36	50	98	142	217	246	278	322	348	432	467	487	564	614	658
160	43	46	123	141	199	265	311	328	392	408	464	524	545	608	656
161	8	64	94	139	189	255	276	348	394	435	468	501	571	585	640
162	21	78	92	172	209	231	234	336	374	414	477	527	544	598	644
163	36	84	126	175	219	234	285	329	364	401	462	521	533	606	633
164	9	74	132	175	206	249	294	339	351	412	476	526	566	607	648
165	43	45	90	176	205	231	281	350	367	431	474	528	537	600	654
166	23	49	103	166	203	243	277	317	381	417	457	506	537	582	623
167	11	82	133	156	203	254	292	353	393	408	480	503	565	583	645
168	38	67	99	165	183	235	269	350	397	422	450	529	551	588	636
169	44	56	130	163	218	263	292	351	367	423	463	516	568	576	651
170	34	51	124	158	210	267	270	326	376	399	441	493	533	598	636
171	44	72	97	161	215	240	306	330	384	407	454	486	564	594	654
172	21	48	128	170	222	262	279	317	370	413	480	524	548	607	624
173	5	67	116	162	216	232	301	349	361	437	444	523	560	594	635
174	25	57	107	170	189	226	269	304	387	427	481	528	536	617	634
175	42	60	125	154	217	242	257	352	386	404	469	522	552	580	644
176	16	89	103	164	208	253	302	345	392	424	481	515	558	587	660

References

[1] Y. Chen and K. K. Parhi, "Overlapped message passing for quasi-cyclic low-density parity check codes," *IEEE Transactions on Circuits and Systems-I*, vol. 51, no. 6, pp. 1106–1113, June 2004.

[2] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementation of the sum-product algorithm for decoding LDPC codes," in *IEEE GLOBECOM*, 2001.

[3] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT press, 1963.

[4] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, July 1948.

[5] S. B. Wicker and V. K. Bhargava, *Reed-Solomon Codes and Their Applications*. New York, NY: John Wiley & Sons, 1999.

[6] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, June 1960.

[7] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding:turbo-codes," in *IEEE International Conference on Communications*, Geneva, Switzerland, 1993, pp. 1064–1070.

[8] D. J. C. MacKay and R. M. Neal, "Near Shannon limit perofrmance of low density parity check codes," *Electronics letters*, vol. 32, no. 18, pp. 1645–1646, 1996.

[9] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. on Information Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[10] I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, and S. Lin, "A class of low-density parity-check codes constructed based on Reed-Solomon codes with two information symbols," *IEEE Comm. Letters*, vol. 7, no. 7, July 2003.

[11] US Patent no. 6023783, Hybrid concatenated codes and iterative decoding.

[12] European Telecommunication Standards Institue, World Wide Web, http://www.dvb.org/documents/white-papers/wp06.DVB-S2.final.pdf.

[13] IEEE 802.3 Standard, World Wide Web, http://standards.ieee.org/getieee802/802.3.html.

[14] IEEE 802.16e Standard, available at http://standards.ieee.org/getieee802/download/802.16e-2005.pdf, Feb. 2006.

[15] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check decoder," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 3, Mar. 2002.

[16] S. Hemati, A. H. Banihashemi, and C. Plett, "A 0.18-μm CMOS analog min-sum iterative decoder for a (32,8) low-density parity-check LDPC code," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 11, pp. 2531–2540, Nov. 2006.

[17] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.

[18] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. on Information Theory*, vol. 47, pp. 498–519, Feb. 2001.

[19] N. Wiberg, *Codes and decoding on general graphs, Ph.D. thesis.* Linkoping: Linkoping University, 1996.

[20] F. Zarkeshvari and A. Banihashemi, "On implementation of min-sum algorithm for decoding low-density parity-check LDPC codes," in *IEEE GLOBECOM*, 2002.

[21] A. Anastasopoulos, "A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," in *Global Telecommunications Conference*, 2001, pp. 1021–1025.

[22] S. Howard, C. Schlegel, and V. Gaudet, "A degree-matched check node approximation for LDPC decoding," in *International Symposium on Information Theory*, Adelaide, Australia, Sept. 2005.

[23] T. R. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. on Information Theory*, vol. 47, no. 2, Feb. 2001.

[24] S. ten Brink, "Convergence of iterative decoding," *Electronic Letters*, vol. 35, no. 10, pp. 806–808, May 1999.

[25] S.-Y. Chung, G. Forney, T. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Communications Letters*, vol. 5, no. 2, pp. 58–60, Feb. 2001.

[26] M. Ardakani and F. Kschischang, "A more accurate one-dimensional analysis and design of irregular LDPC codes," *IEEE Transactions on Communications*, vol. 52, no. 12, pp. 2106–2114, Dec 2004.

[27] H. Zhang and J. Moura, "Large-girth LDPC codes based on graphical models," in *4th IEEE Workshop on Signal Processing Advances in Wireless Communications*, Jun 2003.

[28] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Progressive edge-growth tanner graphs," in *IEEE GLOBECOM*, 2001.

[29] J. Campello, D. Modha, and S. Rajagopalan, "Designing LDPC codes using bit-filling," in *IEEE International Conference on Communications*, Helsinki, Finland, Jun 2001.

[30] Y. Kou, S. Lin, and P. C. Fossorier, "Low-density parity-check codes based on finite geometries: A rediscovery and new results," *IEEE Trans. on Information Theory*, vol. 47, no. 7, pp. 2711–36, Nov. 2001.

[31] IEEE 802.3an Task Force, World Wide Web, http://http://www.ieee802.org/3/an/index.html.

[32] F. Lustenberger, *On the design of analog VLSI iterative codes, Ph.D. thesis.* Zurich: Swiss Federal Institute of Technology, 2000.

[33] V. C. Gaudet and P. G. Gulak, "A 13.3-mb/s 0.35-μm CMOS analog turbo decoder IC with a configurable interleaver," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 11, pp. 2010–2015, Nov. 2003.

[34] V. Gaudet and A. Rapley, "Iterative decoding using stochastic computation," *Electronics Letters*, vol. 39, no. 3, pp. 299–301, February 2003.

[35] M. M. Mansour and N. R. Shanbhag, "A 640-Mb/sec 2048-bit programmable LDPC decoder chip," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 684– 698, Mar. 2006.

[36] P. Urard, E. Yeo, L. Paumier, P. Georgelin, T. Michel, V. Lebars, E. Lantreibecq, and B. Gupta, "A 135-Mb/s DVB-S2 complient codec based on 64800b LDPC and BCH codes," in *IEEE Int. Solid-State Circuits Conference*, 2005.

[37] C.-C. Lin, K.-L. Lin, H.-C. Chang, and C.-Y. Lee, "A 3.33 Gb/s (1200, 720) low-density parity-check code decoder," in *Proceedings of ESSCIRC*, Grenoble, France, 2005, pp. 211–214.

[38] T. Brandon, R. Hang, G. Block, V. C. Gaudet, B. Cockburn, S. Howard, C. Giasson, K. Boyle, P. Goud, S. S. Zeinoddin, A. Rapley, S. Bates, D. Elliott, and C. Schlegel, "A scalable LDPC decoder ASIC architecture with bit-serial message exchange," *Integration VLSI Journal*, 2007, doi:10.1016/j.vlsi.2007.07.003.

[39] A. Darabiha, A. Chan Carusone, and F. R. Kschischang, "Multi-Gbit/sec low density parity check decoders with reduced interconnect complexity," in *ISCAS*, Kobe, Japan, May 2005.

[40] E. Boutillon, J. Castura, and F. R. Kschischang, "Decoder-first code design," in *Proceedings of Turbo Codes*, Brest, France, 2000, pp. 459–462.

[41] T. Zhang and K. Parhi, "VLSI implementation-oriented (3, k)-regular low-densityparity-check codes," in *IEEE Workshop on Signal Processing Systems*, 2001.

[42] M. M. Mansour and N. R. Shanbhag, "High-throughput LDPC decoders," *Trans. on VLSI Systems*, vol. 11, no. 6, pp. 976– 996, Dec. 2003.

[43] L. Chen, J. Xu, I. Djurdjevic, and S. Lin, "Near-Shannon-limit quasi-cyclic low-density parity-check codes," *IEEE Transactions on Communications*, vol. 52, no. 7, pp. 1038–1042, July 2004.

[44] Z. Li, L. Chen, L. Zeng, S. Lin, and W. Fong, "Efficient encoding of quasi-cyclic low-density parity-check codes," *IEEE Transactions on Communications*, vol. 54, no. 1, pp. 71–81, Jan 2006.

[45] A. J. Felstrom and K. S. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Trans. on Information Theory*, vol. 45, no. 6, pp. 2181–2191, Sep. 1999.

[46] Z. Chen, S. Bates, and X. Dong, "Low-density parity-check convolutional codes applied to packet based communication systems," in *IEEE Global Telecommunications Conference*, vol. 3, Dec. 2005, pp. 1250–1254.

[47] R. Swamy, S. Bates, T. L. Brandon, B. F. Cockburn, D. G. Elliott, J. C. Koob, and Z. Chen, "Design and test of a 175-Mb/s, rate-1/2 (128,3,6) low-density parity-check convolutional code encoder and decoder," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 10, pp. 2245–2256, Oct. 2007.

[48] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "High throughput low-density parity-check decoder architectures," in *IEEE GLOBECOM*, 2001.

[49] E. N. Gilbert and H. O. Pollak, "Steiner minimal trees," *SIAM Journal on Applied Mathematics*, vol. 16, no. 1, pp. 1–29, Jan. 1968.

[50] S.-H. Kang and I.-C. Park, "Loosely coupled memory-based decoding architecture for low density parity check codes," *IEEE Transactions on Circuits and Systems-I*, vol. 53, no. 5, pp. 1045–1056, May 2006.

[51] IEEE 802.3 10GBase-T Task Force, World Wide Web, http://www.ieee802.org/3/10GBT/public/novl04/ungerboeck-1-1104.pdf, Nov. 2004.

[52] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, Apr. 1992.

[53] B. H. Calhoun, A. Wang, and A. Chandrakasan, "Modeling and sizing for minimum energy operation in subthreshold circuits," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 9, pp. 1778–1786, Sept. 2005.

[54] S.-J. Lee, N. R. Shanbhag, and A. C. Singer, "A low-power VLSI architecture for turbo decoding," in *International Symposium on Low Power Electronics and Design*, Aug. 2003, pp. 366–371.

[55] A. Darabiha, A. Chan Carusone, and F. R. Kschischang, "A bit-serial approximate Min-Sum LDPC decoder and FPGA implementation," in *ISCAS*, Kos, Greece, May 2006.

[56] E. Zimmermann, G. Fettweis, P. Pattisapu, and P. K. Bo-ra, "Reduced complexity LDPC decoding using forced convergence," in *Int. Symp. on Wireless Personal Mul-timedia Communications*, 2004.

[57] L. W. A. Blad, O. Gustafsson, "An early decision decoding algorithm for LDPC codes using dynamic thresholds," in *European Conference on Circuit Theory and Design*, Aug. 2005, pp. III/285–III/288.

[58] E. Zimmermann, P. Pattisapu, and G. Fettweis, "Bit-flipping post-processing for forced convergence decoding of LDPC codes," in *European Signal Processing Conference*, Antalya, Turkey, 2005.

[59] M. Ardakani and F. R. Kschischang, "Gear-shift decoding," *IEEE Transactions on Communications*, vol. 54, no. 5, pp. 1235–1242, July 2006.

[60] J. Fender, J. Rose, and D. Galloway, "The Transmogrifier-4: an FPGA-based hardware development system with multi-gigabyte memory capacity and high host and memory bandwidth," in *IEEE International Conference on Field-Programmable Technology*, 2005.

[61] L. Yang, H. Liu, and C. J. R. Shi, "Code construction and fpga implementation of a low-error-floor multi-rate low-density parity-check code decoder," *IEEE Transactions on Circuits and Systems I*, vol. 53, no. 4, pp. 892–904, Apr. 2006.

[62] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 386–398, Jan. 2005.

[63] H.-Y. Liu, C.-C. Lin, Y.-W. Lin, C.-C. Chung, K.-L. Lin, W.-C. Chang, L.-H. Chen, H.-C. Chang, and C.-Y. Lee, "A 480 Mb/s LDPC-COFDM-Based UWB baseband transceiver," in *IEEE Int. Solid-State Circuits Conference*, 2005.

[64] B. Razavi, *Design of analog CMOS integrated circuits*. Boston, MA: McGraw-Hill, 2001.

[65] A. Darabiha, A. Chan Carusone, and F. R. Kschischang, "Block-interlaced LDPC decoders with reduced interconnect complexity," *accepted in IEEE Transactions on Circuits and Systems II*, 2007.

[66] ——, "Power reduction techniques for LDPC decoders," *submitted to IEEE Journal of Solid-State Circuits*, 2007.

[67] ——, "A 3.3-Gbps bit-serial block-interlaced min-sum LDPC decoder in 0.13-μm CMOS," in *The IEEE Custom Integrated Circuits Conference*, San Jose, CA, September 2007.

[68] J. A. Davis and J. D. Meindl, *Interconnect Technology and Design for Gigascale Integration*. Norwell, MA: Kluwer Academic, 2003.

[69] D.-U. Lee, W. Luk, J. Villasenor, and P. Cheung, "A hardware gaussian noise generator for channel code evaluation," in *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2003, pp. 69–78.

[70] B. Levine, R. R. Taylor, and H. Schmit, "Implementation of near Shannon limit error-correcting codes using reconfigurable hardware," in *IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, 2000, pp. 217–226.