Energy-Efficient Decoding of Low-Density Parity-Check Codes

Kevin Cushon

Doctor of Philosophy

Department of Electrical and Computer Engineering

McGill University

Montreal, Quebec, Canada

August 2014

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Doctor of Philosophy

DEDICATION

For my family, whose unconditional love and support made this work possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

KEY TO ABBREVIATIONS

ASIC: Application-Specific Integrated Circuit

AWGN: Additive White Gaussian Noise

BER: Bit Error Rate

BPSK: Binary Phase-Shift Keying

BS-(O)MS: Bit-Serial (Offset) Min-Sum

CMOS: Complimentary Metal-Oxide Semiconductor

CN: Check Node

DB: Differential Binary

DC: Decoding Cycle or Down Counter

DD-BMP: Differential Decoding with Binary Message Passing

EG: Euclidean Geometry

EMD: Extrinsic Message Degree

FER: Frame Error Rate

FG: Finite Geometry

FPGA: Field-Programmable Gate Array

Gbps: Gigabits per second

GSP: Gear-Shift Pulse-width min-sum

HDL: Hardware Description Language

IDB: Improved Differential Binary

IEEE: Institute of Electrical and Electronics Engineers

IGSP: IDB with GSP

ITU: International Telecommunication Union

LDPC: Low-Density Parity Check

LLR: Log-Likelihood Ratio

MDD-BMP: Modified Differential Decoding with Binary Message Passing

MS(A): Min-Sum (Algorithm)

MTFM: Majority Tracking Forecast Memory

NMS(A): Normalized Min-Sum (Algorithm)

OMS(A): Offset Min-Sum (Algorithm)

PG: Projective Geometry

PWM-(O)MS: Pulse-Width Modulated (Offset) Min-Sum

QC: Quasi-Cyclic

RHS: Relaxed Half-Stochastic

RIS: Random Initial State

RNG: Random Number Generator

RS: Reed-Solomon

RSF: Random Sign Flip

SAPTL: Sense Amplifier with Pass Transistor Logic

SBF: Soft Bit Flipping

SNR: Signal-to-Noise Ratio

SoC: System on Chip

SP(A): Sum-Product (Algorithm)

TCL: Tool Command Language

TSMC: Taiwan Semiconductor Manufacturing Company

UDC: Up-Down Counter

VFS: Voltage and Frequency Scaling

VLSI: Very Large Scale Integration

VN: Variable Node

WPAN: Wireless Personal Area Network

ABSTRACT

Low-density parity-check (LDPC) codes are a type of error correcting code that are frequently used in high-performance communications systems, due to their ability to approach the theoretical limits of error correction. However, their iterative soft-decision decoding algorithms suffer from high computational complexity, energy consumption, and auxiliary circuit implementation difficulties. It is of particular interest to develop energy-efficient LDPC decoders in order to decrease cost of operation, increase battery life in portable devices, lessen environmental impact, and increase the range of applications for these powerful codes.

In this dissertation, we propose four new LDPC decoder designs with the primary goal of improving energy efficiency over previous designs. First, we present a bidirectional interleaver based on transmission gates, which reduces wiring complexity and associated parasitic energy losses. Second, we present an iterative decoder design based on pulse-width modulated min-sum (PWM-MS). We demonstrate that the pulse width message format reduces switching activity, computational complexity, and energy consumption compared to other recent LDPC decoder designs. Third, we present decoders based on differential binary (DB) algorithms. We also propose an improved differential binary (IDB) decoding algorithm, which greatly increases throughput and reduces energy consumption compared to recent decoders of similar error correction capability. Finally, we present decoders based on gear-shift algorithms, which use multiple decoding rules to minimize energy consumption. We propose gear-shift pulse-width (GSP) and IDB with GSP (IGSP) algorithms, and

demonstrate that they achieve superior energy efficiency without compromising error correction performance.

ABRÉGÉ

Les codes LDPC sont un type de code correcteur d'erreurs qui sont fréquemments utilisés dans les systèmes de communications à haute performance. Cependant, leurs algorithmes de décodage iteratifs à décisions souples souffrent d'une complexité de calcul et d'une consommation d'énergie élevée, ainsi que des difficultés auxiliaires d'implementation en circuit électronique. Il est d'intérêt particulier de mettre au point des décodeurs des codes LDPC à basse consommation d'énergie afin de diminuer le coût de l'opération, augmenter l'autonomie des appareils portables, réduire l'impact sur l'environnement, et augmenter le nombre d'applcations pour ces codes puissants.

Dans ce manuscrit, nous proposons quatre conceptions nouvelles de décodeur de codes LDPC pour lesquelles le but primaire est la réduction de la consommation d'énergie par rapport aux designs précédents. Premièrement, nous présentons un entrelaceur bidirectionnel basé sur les portes de transmission, qui réduit la complexité de filage et les pertes d'énergie parasitaires associées. Deuxièmement, nous présentons une conception de décodeur iteratif basée sur l'algorithme min-somme avec modulation de largeur d'impulsion. Nous démontrons que le format des messages de largeur d'impulsion réduit l'activité de commutation, la complexité informatique, et la consommation d'énergie par rapport aux autres conceptions de décodeur les plus récentes. Troisièmement, nous présentons des décodeurs basés sur les algorithmes binaires différentiels. Nous préposons aussi un algorithme binaire

différentiel amélioré (IDB), qui augmente grandement le débit et réduit la consommation d'énergie par rapport aux décodeurs récents avec une capacité de correction d'erreurs similiaire. Finalement, nous présentons des conceptions de décodeur basées sur les algorithmes de changement de braquet, qui utilisent les règles de décodage multiples pour minimiser la consommation d'énergie. Nous proposons les algorithmes de largeur d'impulsion avec changement de braquet (GSP) et IDB avec GSP (IGSP), et démontrons qu'ils atteignent une efficacité d'énergie supérieure sans compromettre la capacité de correction d'erreurs.

ACKNOWLEDGEMENTS

Scientific progress is made in steps, with the new discoveries growing from the old ones. I would like to express my gratitude towards the following people for laying the foundations of this work:

My supervisors Warren Gross and Shie Mannor, for their sage guidance and instruction.

Saied Hemati, for his kind and patient mentorship.

Camille Leroux, Naoya Onizawa, Saeed Sharifi Tehrani, Ali Naderi, and Guy-Armand Kamendje, for helping me out with the technical side of things.

Fabrice Labeau and Zeljko Zilic, for taking the time to serve on my advisory committee.

And last but not least, my family and friends, for their relentless moral and material support.

PREFACE

This dissertation presents several novel designs that advance the state of the art of LDPC decoding.

The first design we present is a bidirectional interleaver based on transmission gates for multi-mode LDPC codes. This design uses transmission gates to implement a multi-mode circular shift network, which allows a decoder to support multiple LDPC codes. The transmission gates both save silicon area relative to unidirectional multiplexing, and allow the processing nodes to exchange messages over the same wires on successive clock cycles, thus reducing the number of interleaver wires by half. While this design is not greatly successful in reducing energy consumption (due to the need for inefficient tristate buffers), it reduces silicon area by 28% relative to a reference unidirectional interleaver.

In PWM-MS, we propose encoding belief messages in a sign-magnitude pulse width format. Doing so offers four major advantages in the energy domain: low switching activity, reduced routing congestion, simple computational units, and offset min-sum (OMS) implementable for negligible cost over standard min-sum. A PWM-MS decoder designed in 0.13µm CMOS for a (660, 484) LDPC code has an area of 5.76 mm^2. At an SNR of 5.5 dB, this decoder has an average throughput of 5.71 Gbps, average power consumption of 376 mW, and energy consumption of 65.8 pJ per information bit. Compared to bit-serial min-sum (BS-MS), a similar decoder architecture implementing the same code with the same technology, this represents a 19% improvement in energy efficiency. Notably, this design improves

energy efficiency without using any algorithmic heuristics, or otherwise sacrificing error correction performance.

We also present circuit layout designs of LDPC decoders based on differential binary (DB) algorithms. The first of these algorithms, called differential decoding with binary message passing (DD-BMP), and its modified variant (MDD-BMP), were described in prior works. When used with a certain class of LDPC code called *finite geometric* (FG) codes, these algorithms have been shown to produce good error correction performance, despite their low computational complexity - within 0.75 to 1 dB of floating point SPA, and up to 0.5 dB better than the standard min-sum algorithm (MSA). This dissertation presents the first hardware designs of these algorithms. We designed decoders of (273, 191), (1023, 781), and (4095, 3367) FG-LDPC codes in 65 nm CMOS, which achieve respective areas of 0.28 mm^2, 1.38 mm^2, and 15.37 mm^2, average throughputs of 37 Gbps, 75 Gbps, and 141 Gbps, and energy efficiencies of 4.9 pJ/bit, 13.2 pJ/bit, and 37.9 pJ/bit. These designs challenge the common belief that FG codes are too complex for efficient hardware implementation.

We also present a new DB algorithm called IDB, for "improved differential binary", which has improved error correction performance for non-FG codes, and created a circuit design for the (2048, 1723) LDPC code specified in the IEEE 802.3an (10GBASE-T) standard. This decoder achieves an area of 1.44 mm^2, average throughput of 172 Gbps, and an energy efficiency of 2.8 pJ/bit, which represent major improvements over previous decoders of this code with similar error correction performance.

Finally, we present novel decoding algorithms based on gear-shift decoding, along with accompanying hardware designs. In gear-shift decoding, multiple decoding rules may be applied at different stages of decoding a frame. This can be applied to energy-efficient decoding by first attempting low-complexity decoding algorithms, then switching to high-complexity algorithms if decoding fails. In this way, energy efficiency can be increased without sacrificing error correction performance. We show that PWM-MS and IDB are naturally amenable to this strategy, and use them as the basis of two gear-shift algorithms: gear-shift pulse-width (GSP), and IDB with GSP (IGSP). We again implement these algorithms for the (2048, 1723) LDPC code in 65 nm CMOS. Our unpipelined GSP decoder achieves an area of 5.29 mm^2, average throughput of 65.8 Gbps, and an energy efficiency of 37.5 pJ/bit, while a pipelined GSP decoder achieves the same area, 88.1 Gbps throughput, and energy efficiency of 39.3 pJ/bit. These represent slight improvements in energy efficiency and major improvements in throughput over previous decoders with the same level of error correction performance. Our IGSP decoder circuit layout design achieves an area of 6.00 mm^2, average throughput of 100.3 Gbps, and an energy efficiency of 14.6 pJ/bit, which are even larger improvements, albeit at the expense of higher silicon area.

As this dissertation is based on previously published works, some of the work presented herein was performed by the co-authors.

Camille Leroux performed the simulations used to measure the switching activity data plotted in Fig. 4–2. He also wrote the FPGA testbench and hardware

AWGN channel emulator used for deep BER/FER measurements in Chapters 4 and 5, and performed simulations of the decoders described therein. The results of these simulations appear in Figs. 4–9b and 5–4.

Saied Hemati developed the pulse-width modulated message exchange concept that forms the basis of our PWM-MS LDPC decoder, and was a co-developer of the DD-BMP and MDD-BMP algorithms, for which we present hardware designs in Chapter 5. He also contributed the degree-2 variable node design for PWM-MS decoders described in Section 4.3.1, and the PWM-MS check node design shown in Fig. 4–3.

All of the listed co-authors participated in the development of ideas and editorial aspects of the papers in which they are credited, but did not perform any further design or experimentation work.

CHAPTER 1
Introduction

1.1 Motivation

Error control coding is a discipline of information and coding theory used to ensure reliable transmission of information over an unreliable channel. In the most basic sense, this is accomplished by adding structured redundancy to the information to be transmitted, thus allowing the receiver to detect (and possibly correct) errors induced by corrupting noise. It has many uses in computing and communications, and has become an indispensable part of modern applications in these fields [1].

Low density parity check (LDPC) codes are one such type of error control code. Originally described by Robert Gallager in 1962 [2] [3], their proposed iterative decoding algorithms were too computationally complex to be practical at the time. As such, they were largely forgotten until the 1990s, when the discovery of Turbo codes in 1993 [4] led to renewed interest in iterative decoding algorithms.

The major motivating factor behind this interest was that Turbo codes, with a decoding algorithm consisting of iterative soft-decision message-passing, could achieve error correction performance approaching the theoretical limit. This limit, called the "Shannon limit" after its discoverer Claude Shannon, is a hard upper bound on the information-carrying capacity of a channel [5]. In other words, this limit states the maximum achievable efficiency of an error control code. The practical implications of getting closer to the Shannon limit are that a higher information

throughput becomes possible, spectral bandwidth is used more efficiently, and less power can be used in signal transmission compared to other error correction codes. LDPC codes were soon re-discovered, and like Turbo codes, were found to achieve near-capacity performance under iterative message-passing decoding [6] [7] [8].

Since then, LDPC codes have been a topic of great interest in multiple fields, especially information theory, signal processing, communications, and very large-scale integration (VLSI). Proof of the popularity of LDPC codes can be seen in their selection for use in several current and upcoming standards for communications systems:

- For terrestrial wireless communications, WiMax (IEEE 802.16e) [9], Wi-Fi (IEEE 802.11n), [10], and wireless personal area networks (WPAN) (IEEE 802.15.3c) [11].
- For satellite communications, DVB-S2 [12].
- For wireline communications, 10 gigabit per second Ethernet (IEEE 802.3an-2006) [13], and broadband over power lines (IEEE 1901) [14].
- For long-haul fibre-optic communications, ITU G.975.1 [15].

In addition to communications systems, LDPC codes have also been proposed for error correction in data storage, particularly Flash memory [16].

However, despite their popularity, LDPC codes face a number of implementation difficulties owing to the high computational complexity of their decoders. Practical LDPC codes generally have block lengths ranging from a few hundred to a few thousand bits, all of which require recomputing at every iteration of the decoding process. Another major implementation challenge for a decoder is the communication

structure necessary to support the message passing. Straightforward fully parallel architectures require thousands of interconnections, leading to congested routing networks, and significant energy loss due to wiring parasitics [17].

Because many applications of LDPC codes require energy consumption to be as low as possible, there is a high demand for energy efficient decoders. Thermal management is a limiting factor in many systems, particularly large systems-on-chip (SoCs) and microprocessors in which performance is temperature- or hotspot-limited [18]. This is especially true in the emerging field of three-dimensional chip stacks [19]. Energy efficiency is key in embedded systems, which typically have highly constrained power budgets and limited thermal management options [20]. Perhaps the greatest source of demand is from battery-powered gadgets, such as laptop computers, smartphones, and tablets - these all have large and highly competitive markets, and require energy efficiency to be a top design priority in order to extend battery life. Another application is datacenter servers, as the cost of operation is primarily dependant on power consumption [21] [22], and the increasing popularity of decentralized computing is driving higher demand for datacenter resources. Emerging or niche fields requiring energy efficient communications exist as well, such as sensor networks, building monitoring, smart warehouses, and implantable medical devices [23]. Finally, increased efficiency leads to lower cost of operation, and in aggregates of large numbers of devices, also reduces the need for energy generation infrastructure and associated environmental costs.

To address this problem, this dissertation proposes four novel energy-efficient LDPC decoder designs. The first is a bidirectional interleaver based on transmission gates, which attempts to address energy consumption by reducing interconnect complexity. The second is a *pulse width modulated min-sum* (PWM-MS) decoder. The third is a series of decoders based on differential binary (DB) algorithms: the previously proposed *differential decoding with binary message passing* (DD-BMP) algorithm and its modified variant (MDD-BMP) [24]. We also propose an *improved differential binary* (IDB) decoding algorithm, and present an accompanying hardware design. Finally, the fourth design is a series of decoders based on energy-efficient gear-shift decoding algorithms, which are algorithms that use multiple update rules. Our previously mentioned PWM-MS and IDB decoders are highly adaptable to this concept. We propose two gear-shift algorithms based thereupon called *gear-shift pulse-width* (GSP) and *IDB with GSP* (IGSP) and their hardware designs.

1.2 Objectives

The primary goal of this research is to produce LDPC decoder designs with improved energy efficiency compared to previously published designs. Secondary design goals are hardware efficiency, high throughput, and good error correction performance. Recently published LDPC decoder designs, particularly those that are designed for energy efficiency, are used as benchmarks. However, as trade-offs between conflicting goals are a necessary part of design, justifiable trade-offs that increase energy efficiency are given the highest consideration.

1.3 List of Publications

The work presented in this dissertation has been published in the following peer-reviewed scholarly publications:

- K. Cushon, W. J. Gross, and S. Mannor. Bidirectional interleavers for LDPC decoders using transmission gates. In *Proceedings of the IEEE Workshop on Signal Processing Systems*, pages 232 237, Tampere, Finland, 2009.
- K. Cushon, C. Leroux, S. Hemati, S. Mannor, and W. J. Gross, "A Min-Sum Iterative Decoder Based on Pulsewidth Message Encoding," *IEEE Transactions on Circuits and Systems-II: Express Briefs*, vol. 57, no. 11, pp. 893 897, Nov. 2010.
- K. Cushon, C. Leroux, S. Hemati, S. Mannor, and W. J. Gross, "High-Throughput Energy-Efficient LDPC Decoders Using Differential Binary Message Passing," *IEEE Transactions on Signal Processing* vol. 62, no. 3, pp. 619 631, Mar. 2014.
- K. Cushon, S. Hemati, S. Mannor, and W. J. Gross, "Energy-Efficient Gear-Shift LDPC Decoders," in *Proceedings of the 2014 IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, pp. 219 223, Zürich, Switzerland, Jun. 2014.

1.4 Dissertation Outline

The remainder of this dissertation is structured as follows.

Chapter 2 provides background for the material presented in the later chapters. It contains a review of LDPC codes, including commonly-used decoding algorithms, as well as the problems and strategies of hardware decoder implementation.

It also reviews energy consumption in micro-electronic circuits, and strategies used in energy-efficient digital circuit design. Finally, this chapter also provides a literature review of energy-efficient LDPC decoders, including state-of-the-art designs.

Chapter 3 proposes a bi-directional interleaver and circular shift network for multi-mode quasi-cyclic (QC) LDPC codes. This interleaver halves the number of required wires and occupies less area than the equivalent unidirectional design. This chapter is based in part on material from our conference paper [25].

Chapter 4 proposes an LDPC decoder design based on pulse-width encoded message exchange, which we refer to as a *pulse-width modulated min-sum* (PWM-MS) decoder. This message encoding increases energy efficiency by reducing switching activity and computational complexity. We also present an ASIC layout design of a PWM-MS decoder. This chapter is based in part on material from our journal paper [26].

Chapter 5 describes a series of LDPC decoders based on differential binary (DB) algorithms. We present the first hardware designs of the DD-BMP and MDD-BMP algorithms [24], as well as the first fully parallel implementations of FG-LDPC codes. We also present an improved differential binary (IDB) decoding algorithm, and design it in hardware for the (2048,1723) LDPC code from the IEEE 802.3an (10GBASE-T) standard [13]. Post-layout results for these decoders are compared with other state-of-the-art energy-efficient LDPC decoder designs. This chapter is based in part on material from our journal paper [27].

Chapter 6 explores decoders based on gear-shift algorithms that are capable of decoding using multiple algorithms. We show that the the PWM-MS and IDB algorithms from the previous chapters can be extended or combined to create gear-shift algorithms with little hardware overhead. We introduce and GSP and IGSP algorithms, and show that they achieve energy efficiency comparable to low-complexity decoding algorithms, and the error correction performance of offset min-sum. We also propose hardware designs for these decoders that take advantage of architectural commonalities to attain good area efficiency. As before, we present circuit layout results and perform comparisons with previously proposed decoder designs. Material from this chapter also appears in our conference paper [28].

Finally, Chapter 7 discusses possible avenues for future research, and concludes the dissertation.

1.5 Summary

Many applications exist for energy efficient LDPC decoders - there are current applications in communications, and emerging applications in fields such as implanted biomedical devices. However, the high complexity of the required computations and message passing clashes with the low energy requirement. Many different approaches have been taken to make energy efficient LDPC decoders. Dynamic energy dominates total energy consumption in published decoder designs down to 65 nm, though static energy is an important consideration with smaller processes. The goal of this research is to improve energy efficiency over previously published LDPC

decoder designs by addressing the factors contributing to dynamic energy consumption - computational complexity, switching activity, interconnect parastitics, and supply voltage.

CHAPTER 2
Background and Literature Review

2.1 Introduction

This chapter will provide background information and a review of relevant prior literature.

First, a brief overview and tutorial on LDPC codes will be given, followed by descriptions of the most well-known decoding algorithms: the Gallager A and B algorithms, the sum-product algorithm (SPA), and the min-sum algorithm (MSA) and its variants. The next section will review the evolution of LDPC decoder design and complexity management strategies, with examples from the literature. This review will emphasize designs intended to be energy-efficient. Finally, this section will be concluded with an overview of general strategies for energy reduction in digital CMOS circuits, and how they were applied to the designs presented in this dissertation.

2.2 LDPC Codes

An LDPC code is defined by a parity check matrix $\mathbf{H}$. An (n, k) LDPC code has a parity check matrix with dimensions k rows by n columns. The elements of $\mathbf{H}$ describe the relationships between n received symbols, which are represented by the columns, and k parity checks, which are represented by the rows. As this

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

Figure 2–1: The parity check matrix of a $(16, 8)$ regular LDPC code with column weight 2 and row weight 4.

dissertation is heavily focused on practical circuit implementation, we will restrict our consideration to binary LDPC block codes.*

A bit i participates in parity check j if $\mathrm{H}_{i,j} = 1$, otherwise $\mathrm{H}_{i,j} = 0$. The name "low-density parity check" comes from the sparsity of non-zero entries in $\mathbf{H}$. The row and column weights of H are defined as the number of non-zero entries in a given row or column, respectively. If all rows have equal weight and all columns have equal weight, then the code is said to be regular; otherwise, it is irregular. Figure 2–1 shows an example parity check matrix for a $(16, 8)$ regular LDPC code.

A frame $\mathbf{d} = (d_0, d_1, ..., d_{n-1})$ constitutes a valid codeword if and only if:

* Non-binary definitions of LDPC codes date back to the original Gallager thesis [2], but have failed to gain popularity among implementors due to obnoxiously high decoding complexity and low throughput compared to binary codes [29] [30] [31]. Likewise, LDPC convolutional codes have also been investigated in the literature [32], but have been largely ignored in favor of block codes for practical implementations.

$$\mathbf{dH}^{\mathrm{T}} = \mathbf{0}. \tag{2.1}$$

This is defined as the syndrome of $\mathbf{d}$. If the syndrome computed over GF(2) is a zero vector, then all parity checks are satisfied. A non-zero syndrome implies one or more unsatisfied checks.

Encoding, which is typically performed by the transmitter, is done using a generator matrix $\mathbf{G}$ in which:

$$\mathbf{GH}^{\mathrm{T}} = \mathbf{0}. \tag{2.2}$$

The matrix $\mathbf{G}$ has dimensions of $k \times n$. Given a vector $\mathbf{b}$ of k arbitrary information bits $\mathbf{b} = (b_0, b_1, ..., b_{k-1})$, a frame, or valid codeword $\mathbf{x} = (x_0, x_1, ..., x_{n-1})$, can be generated as follows:

$$\mathbf{x} = \mathbf{bG}. \tag{2.3}$$

Alternatively, $\mathbf{x}$ is equivalent to the concatenation of $\mathbf{b}$ with a vector of $n - k$ parity bits $\mathbf{p}$, i.e. $\mathbf{x} = (x_0, x_1, ..., x_{n-1}) = (b_0, b_1, ..., b_{k-1}, p_0, p_1, ..., p_{n-k-1})$. Thus, $\mathbf{G}$ consists of a $k \times k$ identity matrix and a $k \times (n - k)$ non-systematic portion. The non-systematic portion of $\mathbf{G}$, $\mathbf{G_p}$, can be used without the systematic portion to generate $\mathbf{p}$:

$$\mathbf{p} = \mathbf{bG_p}. \tag{2.4}$$

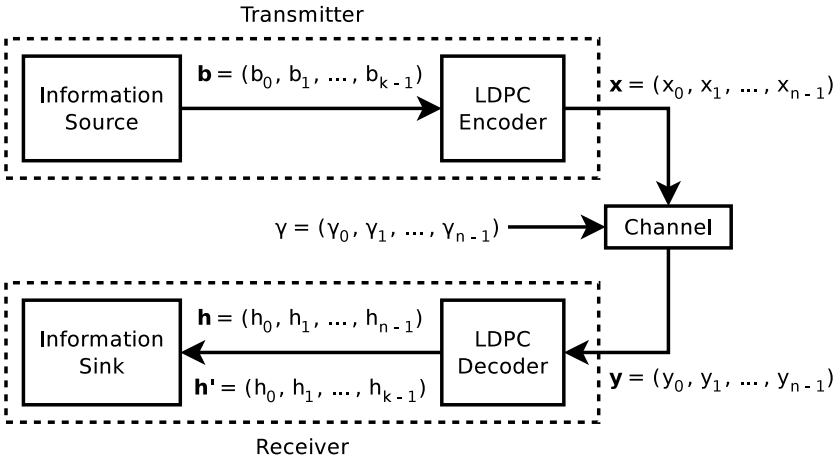


Figure 2–2: A communication system showing LDPC encoding and decoding.

After the frame $\mathbf{x}$ is constructed, it is then transmitted through a channel, where it may be corrupted by noise $\boldsymbol{\gamma}$. The receiver acquires the noisy frame $\mathbf{y} = (y_0, y_1, ..., y_{n-1})$, which serves as input to the decoder, which in turn produces as output the decoded frame $\mathbf{h}$. At this point, the parity portion of $\mathbf{h}$ can be discarded, since it serves no use to the receiver, leaving $\mathbf{h}' = (h_0, h_1, ..., h_{k-1})$. If decoding is successful, then $\mathbf{h}' = \mathbf{b}$. Figure 2–2 illustrates the process of encoding, transmission, and decoding.

2.2.1 Tanner Graphs

An LDPC code can also be described using a graphical model called a Tanner graph [33]. These graphs provide a generic realization and decoding framework for many codes. In particular, LDPC and Turbo codes can both be described using Tanner graphs, and decoded using iterative message-passing algorithms operating on these graphs [34] [35].

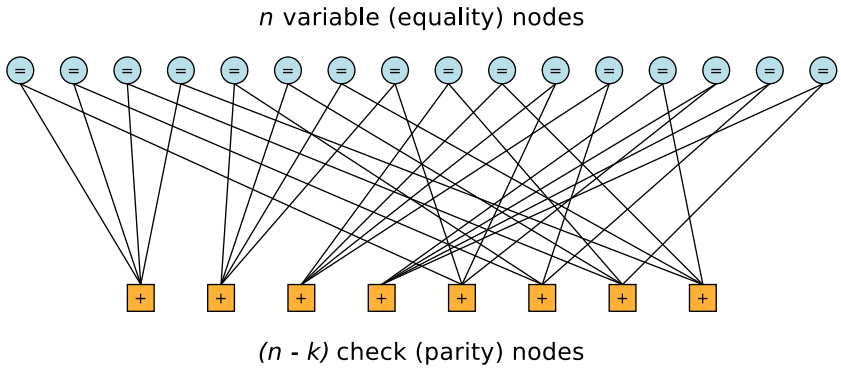


Figure 2–3: Tanner graph representation of the $(16, 8)$ LDPC code whose parity check matrix is shown in Figure 2–1.

Figure 2–3 shows a Tanner graph representation of the $(16, 8)$ LDPC code from Figure 2–1. A Tanner graph is a bipartite graph that describes the constraints or parity check equations of a linear block code, and thus is an alternate representation of the parity check matrix $\mathbf{H}$. One set of vertices, called the *variable nodes* (VNs),[†] represent the variables, or estimations of the decoded message. They are equivalent to the columns of $\mathbf{H}$. The other set of vertices, called the *check nodes* (CNs),[‡] represent the parity check constraints, or the rows of $\mathbf{H}$. An edge between VN i and CN j exists if and only if symbol i participates in parity check j. Edges are thus equivalent to non-zero elements in $\mathbf{H}$. The number of edges incident to a node is called the *degree* of the node. The variable node degrees (d_v) and check node degrees (d_c) are equivalent to the column and row weights of $\mathbf{H}$, respectively.

[†] Also called "bit", "equality", and in general (non-binary) cases, "digit" or "symbol" nodes.

[‡] Also called "parity" or "subcode" nodes.

2.3 Decoding Algorithms

While many reasonable message-passing algorithms for decoding LDPC codes have been devised, this section will describe the most prominent and important ones: the Gallager A and B algorithms, the sum-product algorithm (SPA), the min-sum algorithm (MSA), and MSA variants offset min-sum (OMS) and normalized min-sum (NMS).

2.3.1 Gallager A and B Algorithms

Gallager originally proposed two different algorithms for decoding LDPC codes [2]. The first is a low-complexity bit-flipping algorithm which is better known today as the "Gallager B" algorithm.

I. Initialization

This algorithm operates entirely in the binary domain and takes as input a block of n bits from the channel:

$$y_v \in \{0, 1\} \, \forall \, 0 \leq v < n \tag{2.5}$$

The initial variable-to-check messages are set to the received bits:

$$m_{v \to c} = y_v \tag{2.6}$$

II. Check node computations

Next, the check-to-variable messages are computed as the modulo-2 sum of all inputs to the check node, excluding the input from the edge for which the output is being calculated:

$$m_{c\to v} = \bigoplus_{v'\in N_c\setminus v} m_{v'\to c}, \tag{2.7}$$

where N_c is the set of VNs neighbouring check node c, and the $\bigoplus$ symbol signifies modulo-2 addition.

III. Variable node computations

Now the variable-to-check messages are re-computed using the messages received from the check nodes:

$$m_{v\to c} = \begin{cases} \overline{m_{v\to c}}, \text{ if } |\{c : m_{c\to v} = \overline{m_{v\to c}}\}| \geq b_k\,, \\ \\ m_{v\to c}, \text{ if otherwise.} \end{cases} \tag{2.8}$$

Put differently, the variable node flips the value of an outgoing message if at least b_k of the incoming messages suggests that it should. The threshold b_k can vary with the iteration number k, and can take any integer value in the range $\frac{d_v-1}{2} \leq b_k \leq d_v-1$.

The Gallager A algorithm is, in fact, a special case of Gallager B in which $b_k = d_v - 1$. In other words, an outgoing message flips only when all incoming messages disagree with its current value.

IV. Output

The estimation of the decoded codeword $\mathbf{h}$ can be computed by majority vote of each variable node's outputs:

$$h_v = \begin{cases} 0, \text{ if } \sum_{c\in N_v} m_{c\to v} \leq \dfrac{d_v}{2}\,, \\ \\ 1, \text{ if otherwise.} \end{cases} \tag{2.9}$$

Some alternative ways of computing $\mathbf{h}$ are to assign h_v a random bit if the sum is exactly zero, or to leave h_v unassigned until all outputs of variable node v are in agreement.

Equations 2.7 - 2.9 are then iteratively repeated until some criterion for stopping is met. Usually, this is when a valid codeword is detected (i.e., $\mathbf{hH}^{\mathrm{T}} = \mathbf{0}$), or when a maximum iteration limit k_{max} is reached.

In both the variable and check node computations, the calculation for a given outgoing edge excludes the input from that edge. This is because this information is intrinsic to the calculation, while the contributions from the other edges are extrinsic information. Including the intrinsic information would create feedback loops between neighbouring nodes that bias the message update calculations in favour of their current values. This reduces the error correction performance of the algorithm. However, as will be shown in later chapters of this dissertation, this can be used as a heuristic to reduce computational complexity, particularly in the check node.

2.3.2 The Sum-Product Algorithm

In addition to the simple bit-flipping algorithm described above, Gallager also proposed a decoding algorithm which operated on probabilities, which he called "probabilistic decoding" [2]. While much more computationally complex, this algorithm gives much better error correction performance, as it can take into account the level of confidence in which a bit is correct. This algorithm turned out to be a form of the sum-product algorithm [34], which is itself a special case of Pearl's belief propagation algorithm [36].

Like the Gallager A & B algorithms, the basic principle of this algorithm is the exchange of messages, or "beliefs", between the variable and check nodes. When a processing node is activated, it reads the messages from neighbouring nodes over the graph's edges, and updates its belief based on these messages. Unlike the Gallager A & B algorithms, which are "hard decision", the sum-product algorithm is "soft decision" - the exchanged messages and internal node computations operate on multi-level likelihood estimations rather than a hard estimation of the decoded bit.

The modern formulation of the sum-product LDPC decoding algorithm is as follows.

I. Initialization

The variable nodes are initialized with soft information from the channel. The n-element vector $\mathbf{L}$ contains the real-valued log-likelihood ratios (LLRs) of each of the n received symbols: [§]

$$L_v = \ln\left[\frac{P(x_v = 0|y_v)}{P(x_v = 1|y_v)}\right] \in \mathbb{R} \; \forall \; 0 \leq v < n\,. \tag{2.10}$$

In most studies of LDPC decoder performance, a binary phase-shift keying (BPSK) channel with additive white Gaussian noise (AWGN) is used. In this special case, it can be shown that the LLRs are a function of the received values $\mathbf{y}$ and channel noise variance σ^2:

[§] The SPA works in the logarithmic domain as a matter of computational simplification, as products of raw probabilities become sums of LLRs.

$$L_v = \frac{2}{\sigma^2} \cdot y_v \in \mathbb{R} \; \forall \; 0 \leq v < n \, . \tag{2.11}$$

The initial VN-to-CN messages are set to $\mathbf{L}$, while the iteration number (indicated by the superscript on m) is initialized to 0:

$$m^0_{v \to c} = L_v \tag{2.12}$$

II. Check node computations

Next, the CN-to-VN messages are computed. The SPA check node operation is:

$$m^k_{c \to v} = 2 \cdot \operatorname{atanh} \left(\prod_{v' \in N_c \backslash v} \tanh \left(\frac{m^k_{v' \to c}}{2} \right) \right) . \tag{2.13}$$

III. Variable node computations

Now, the iteration number k is incremented, and the new VN-to-CN messages are computed based on the previous iteration's CN-to-VN messages:

$$m^k_{v \to c} = L_v + \sum_{c' \in N_v \backslash c} m^{k-1}_{c' \to v}, \tag{2.14}$$

IV. Output

To generate the estimation of the decoded block $\mathbf{h}$, a hard decision for each bit is computed at the VNs based on a sum of all incoming messages from neighbouring VNs:

$$h_v^k = \begin{cases} 0, \text{ if } L_v + \sum_{c \in N_v} m_{c \to v}^{k-1} \geq 0\,, \\ \\ 1, \text{ if otherwise.} \end{cases} \tag{2.15}$$

As before, Equations 2.13 - 2.15 are repeated iteratively until a stopping criterion is met, such as detection of a valid codeword or reaching the maximum iteration limit k_{max}.

In practical implementations, SPA-based decoders face major implementation problems owing mainly to two factors. The first is the difficulty of computing the transcendental functions *tanh* and *atanh* [17]. The second related problem is the imprecise representation of real numbers in systems with a finite number of quantization bits, which can lead to major degradations in error correction performance - particularly due to numerical saturation in the output of *tanh* and *atanh* [37].

However, SPA is still widely accepted as the "gold standard" of LDPC decoding algorithms, as it has been proven to be optimal on cycle-free graphs [7] [38] [39]. Practical LDPC codes are not cycle-free, although SPA decoding with high-precision floating-point computations often results in the best attainable error correction performance - cases where it does not are usually noted in the literature when discovered.

2.3.3 The Min-Sum Algorithm

The MSA can be considered a heuristic variant of the SPA in which the transcendental functions in the check nodes are replaced with simpler computations. Like the SPA, the MSA is a belief propagation algorithm operating over a graphical model of the code [40]. They are, in fact, highly similar - the only difference is the check node

operation, in which the magnitude of the outgoing messages is set to the minimum of the incoming messages.

Thus, Equation 2.13 above is replaced with:

$$m_{c\to v}^{k} = \prod_{v'\in N_c\setminus v} \operatorname{sgn}\left(m_{v'\to c}^{k}\right) \cdot \min_{v'\in N_c\setminus v}\left(\left|m_{v'\to c}^{k}\right|\right). \tag{2.16}$$

The initialization, VN update, and hard decision equations are the same as in the SPA.

2.3.4 Normalized and Offset Min-Sum

As the MSA check node operation is an approximation of the optimal SPA operation, there is a loss in error correction performance compared to the SPA. However, as the MSA CN messages consistently have higher magnitude than in SPA [41], correction factors can be applied to minimize this loss [42].

In the normalized min-sum algorithm (NMSA), the output of the check node is scaled by a constant factor α, with $0 < \alpha < 1$:

$$m_{c\to v}^{k} = \alpha \cdot \prod_{v'\in N_c\setminus v} \operatorname{sgn}\left(m_{v'\to c}^{k}\right) \cdot \min_{v'\in N_c\setminus v}\left(\left|m_{v'\to c}^{k}\right|\right). \tag{2.17}$$

In the offset min-sum algorithm (OMSA), a subtractive offset β, with $\beta > 0$, is applied to the magnitude:

$$m_{c\to v}^{k} = \prod_{v'\in N_c\setminus v} \operatorname{sgn}\left(m_{v'\to c}^{k}\right) \cdot \min_{v'\in N_c\setminus v}\left(\max\left(\left|m_{v'\to c}^{k} - \beta\right|, 0\right)\right). \tag{2.18}$$

The correction factors α and β can also be applied simultaneously, in which case the algorithm is known as normalized offset min-sum. Typical values for α and

β are 0.75 and 1, respectively, although the optimal values depend on the LDPC code, channel, and internal number representation of the decoder. A carefully tuned NMSA/OMSA decoder can generally achieve error correction performance within 0.1 dB of an SPA decoder [43]. With certain codes, NMSA/OMSA can actually outperform SPA due to the effects of cycles in the code's graph [44].

2.3.5 A Note on Quantization

Until now, we have only considered decoding algorithms operating over the range of real numbers. However, any practical decoder implementation must necessarily have a finite number of quantization levels, which will have a negative impact on the decoder's performance due to numerical saturation and quantization error. LDPC decoder implementations in software generally use high-precision floating point numbers; however, 4 to 7 bits of quantization is generally sufficient to achieve negligible (or at least justifiable) performance loss compared to floating-point numbers [45]. VLSI decoder implementations generally use as few quantization bits as possible to reduce memory and computational unit complexity.

Quantization schemes generally involve clipping LLR values above a certain threshold T_{clip}, and dividing the range $[-T_{clip}, T_{clip}]$ into 2^q uniform intervals, where q is the number of quantization bits [46]. Some more advanced quantization techniques have been proposed to increase error correction performance, such as adaptively varying T_{clip} based on channel noise estimations or the number of unsatisfied parity checks [47].

2.4 Literature Review of LDPC Decoder Architectures

This section will review the progression and current state of LDPC decoder hardware architectures, with an emphasis on energy-efficient design.

2.4.1 Early Attempts and Decoder Architectures

The first VLSI implementation of an LDPC decoder was presented by Blanksby and Howland in 2002 [17], and was a fully-parallel SPA-based design, with the check node computations performed via look-up tables. This design soon became emblematic of the difficulties of implementing LDPC decoders, particularly with the interconnections required to support fully parallel message passing. Routing congestion caused this design to be quite area-inefficient, with 50% utilization. High wiring delay and parasitics also caused it to have a longer critical path and higher power consumption than expected.

While the check node computation problem could be eased by MSA-based decoders, the routing problem was (and remains) a serious issue affecting the viability of LDPC codes. This led to the co-development of architecture-aware LDPC codes - i.e., LDPC codes in which the parity-check matrix $\mathbf{H}$ has a regular structure that can be exploited in hardware implementations [48] - and partially-parallel decoder architectures for these codes [49].

The most important and prominent type of architecture-aware LDPC code is the quasi-cyclic (QC)-LDPC code [50]. In these codes, the $\mathbf{H}$ matrix is composed of sub-blocks of circulant permutation matrices - usually in the form of circularly shifted identity matrices. Figure 2–4 shows an example of one such code. This $(648, 486)$

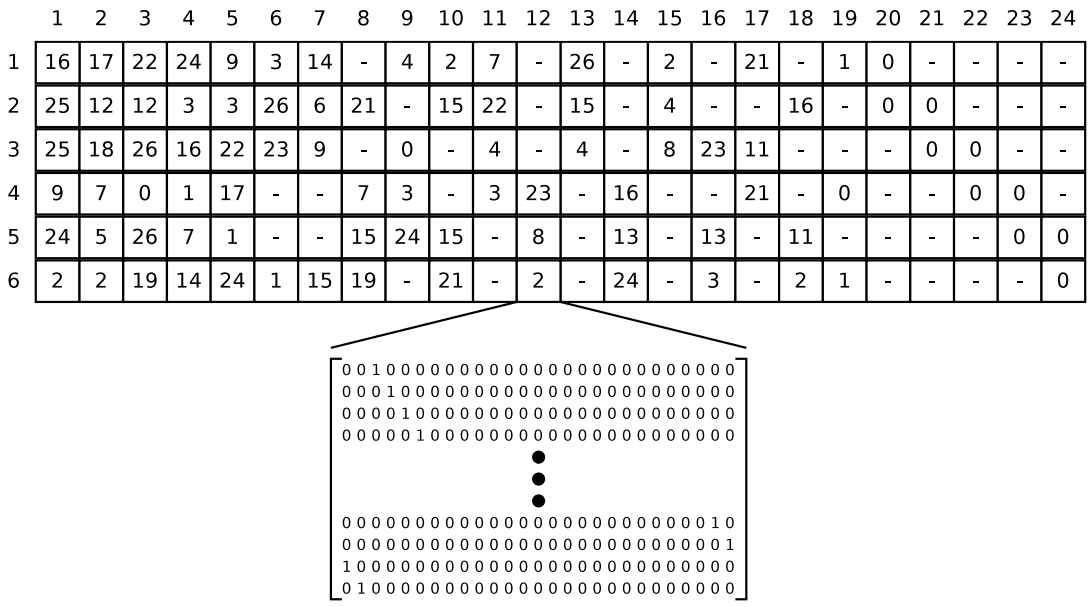


Figure 2–4: Example of an **H** matrix for a quasi-cyclic (QC)-LDPC code, showing its construction from blocks of circularly shifted identity matrices.

QC-LDPC code from the IEEE 802.11n standard [10] is made up of 27×27 sub-blocks that are either an identity matrix circularly shifted to the right ρ places, or a zero matrix (indicated by a dash). The expanded block illustrates how the model matrix of sub-blocks translates to a conventional binary **H** matrix.

QC-LDPC codes became popular standard codes because they lend themselves well to efficient partially parallel implementations, allowing designers to avoid the problems faced by early fully parallel decoders. Partially parallel architectures implement a subset of the overall graph, and multiplex the message passing and node computations over time. At the extreme end of this concept are serial decoders [51], which implement a single processing element and compute each entry in **H** serially like a microprocessor. However, serial decoding results in throughput too low for

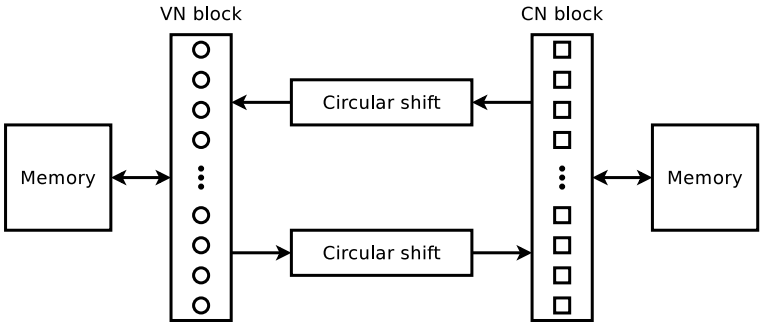


Figure 2–5: Basic architecture for a partially parallel decoder of QC-LDPC codes.

most modern communications applications. Partially parallel architectures aim for a good trade-off between hardware complexity and throughput.

Figure 2–5 shows a straightforward (though inefficient) partially parallel decoder architecture for QC-LDPC codes. The **H** matrix is processed one sub-block at a time, with the partial results stored in memory. The VN and CN blocks are connected through circular shift networks, which implement all the necessary sub-block permutations in **H**.

2.4.2 Efficiency Improvements

While these early efforts established the basic algorithms and decoder architectures, a great deal of research effort has since gone into improving upon them. Improvements in hardware and algorithmic efficiency generally also improve energy efficiency, either by reducing the amount of computational "work" needed to decode a frame, or lessening auxiliary circuit implementation issues (such as high parasitic losses in wiring due to routing congestion).

Layered decoding algorithms are one such improvement. Conventional decoding uses a strict two-phase-per-iteration message-passing schedule, in which all VN messages are computed and transmitted, followed by all CN messages (this is sometimes referred to as a "flooding" schedule). Layered decoding algorithms encompass a range of algorithms in which subsets of the graph (referred to as "layers") update their beliefs in sequence, which are then immediately made available to successive layers. For instance, in row-layered decoding [52], subsets of CNs (i.e., rows of $\mathbf{H}$) update in sequence, with the neighbouring VNs immediately updating their beliefs before the next layer of CNs is processed. As a result, the VNs update their beliefs several times per iteration. Column-layered decoding [53] is the same concept with the layers divided into columns. Layered decoding is very popular in modern partially-parallel VLSI implementations, since it significantly reduces the average number of iterations required for convergence to a valid codeword - generally by about a factor of 2. Other message passing schedules similarly yield faster convergence. Turbo-decoding message passing is a special case of row-layered decoding in which each sub-matrix "layer" has a column weight of 1 - this allows VN and CN circuitry to be significantly simplified [54]. An interesting area of ongoing work is residual belief propagation, which uses a type of informed dynamic message passing [55].

In spite of all the interest in partially-parallel decoders, a number of highly-parallel and fully-parallel decoder designs have been proposed. These generally make use of more efficient circuit architectures or heuristics to avoid the hardware complexity and routing issues made famous in [17].

One example, called *broadcasting*, reduces computational and routing complexity by computing only one outgoing message per node, and broadcasting it to all neighbouring nodes. A half-broadcasting min-sum decoder is proposed in [56], in which broadcasting is applied to the CNs, while the VNs compute unique messages for each outgoing edge as in Equation 2.14. While this is a heuristic, the error correction performance loss can be mostly recovered with local computations in the VN, i.e., if a VN detects that it produced the minimum message, it increases the magnitude of the corresponding message slightly. In this way, the performance loss can be reduced to approximately 0.1 dB. A non-heuristic variant of broadcasting is presented in [57], in which the VNs broadcast a single message, and CNs broadcast 4 messages: first and second minima, the index of the input that produced the first minimum, and the one-bit product of the signs of all incoming messages. The "true" min-sum CN-to-VN messages can then be recovered without any loss via local computations in the VN.

Bit-serial architectures are proposed specifically as energy-efficient LDPC decoder designs in [58] and [59]. In these designs, the messages are exchanged bit-serially over single wires. This not only reduces routing congestion, but also computational complexity, as the primary operations of the MSA (i.e., addition and minimum-finding) are very amenable to bit-serial computation. Despite being fully

parallel,[¶] these decoders avoid routing congestion and impractical size due to these characteristics.

The authors of [58] further argue that higher parallelism can be used to improve energy efficiency through voltage and frequency scaling (VFS). Reducing the supply voltage greatly reduces the energy use of a circuit, as dynamic energy consumption scales with V_{DD}^2. A fully parallel decoder can achieve a given throughput with a lower clock frequency than a less parallel decoder, and can therefore use a lower supply voltage.

Some partially parallel decoders (albeit with a high degree of parallelism) have also been proposed as energy-efficient architectures. One example is [60], which proposes decoders of the $(2048, 1723)$ 10 Gbps Ethernet LDPC code. This code is made up of 64×64 permutation sub-matrices, which is exploited using techniques the authors refer to as node grouping and wire bundling. In the implemented decoder, all VNs are implemented in parallel. Each sequentially transmits a single VN-to-CN message, then receives the returning messages from the CNs (as $d_v = 6$ in this code, a decoding iteration thus takes 12 clock cycles). The computations are non-heuristic offset min-sum, with a post-processing phase for improving error correction

[¶] There is an argument that bit-serial architectures are not fully parallel, as they multiplex computation and message passing over time. However, the cited designs are generally considered fully parallel as they implement the entire graph in parallel - every edge and processing node maps to a unique instance in the circuit. In this dissertation, we use the term "fully node-parallel" when a distinction between the two is required.

performance in the high SNR region by lowering error floors.[‖] As this decoder achieves high throughput, VFS is applied to dramatically increase energy efficiency.

A similar architecture called "shift-structured decoding" is presented in [61]. In this design, each VN has fixed connections to a subset of its neighbouring CNs. The CNs are connected together in a circular shift register, which is used to pass messages directly between the CNs for intra-iteration computations. A second inter-iteration communication network between the CNs is used to complete the CN computations. Originally proposed for QC-LDPC codes, this architecture has also been applied to RS-based codes, including the 10 Gbps Ethernet code [62].

Split-row min-sum is another proposed energy-efficient decoding architecture [63]. Split-row decoding uses a heuristic algorithm that divides the **H** matrix into several columns, or sub-codes. Check node computations for each sub-code are performed separately, using limited information from other columns. In other words, the "global" CNs are replaced with several "local" ones with considerably reduced complexity. While the original split-row algorithm had a significant loss in error correction performance, a revised version presented in [64] gained back most of this loss with improved inter-column communication and offsetting computations. A fully-parallel implementation of improved split-row for the 10 Gbps Ethernet LDPC code achieved very high throughput and area efficiency. Prior to the designs presented in this dissertation, split-row also had the best reported energy efficiency for this code.

[‖] The topic of error floors is covered in more detail in Chapter 5.

Another approach to energy efficient LDPC decoding is to use a simpler decoding algorithm, thus trading off error correction performance for reductions in complexity and energy cost. There are many decoding algorithms across the performance/complexity range of LDPC decoding. At the low-complexity end of this range are hard decision algorithms, such as the aforementioned Gallager A and B algorithms, bit flipping [65], and majority-based algorithms [66]. At the high-complexity end are SPA and MSA. In the middle are a variety of different decoding algorithms, each offering a different balance between complexity, speed, and error correction performance. Some notable examples are the heuristic variants of MSA mentioned above, hybrid hard-soft algorithms like bootstrap decoding [67], and weighted bit flipping [68]. However, it is generally not desirable to give up significant (or even any) error correction performance, as near-capacity performance is the main reason for using LDPC codes in the first place. Furthermore, communication standards generally include stringent requirements for frame (FER) or bit error rate (BER) that the less complex algorithms cannot achieve.

Two notable examples of reduced-complexity algorithms that achieve error correction performance similar to MSA are stochastic algorithms [69] and differential decoding with binary message passing (DD-BMP) [24]. In stochastic decoding, internode communication is performed with stochastic Bernoulli streams wherein the statistical properties constitute the information being exchanged. These streams are transmitted over single wires, as in bit-serial min-sum, so these decoders do not face routing congestion problems. VN and CN computations are also greatly simplified in the stochastic domain, although additional memory elements are needed to

"decouple" the stochastic streams. Fully parallel stochastic LDPC decoder designs can be found in [70] and [71]. Relaxed half-stochastic (RHS) decoding is a variant in which VN computations are performed in the definite number domain, while message passing and CN computations are performed in the stochastic domain [72].

DD-BMP is a decoding algorithm in which the VNs and CNs exchange binary messages, and the VNs update their beliefs by applying the differential sum of their inputs to their previous states. This algorithm is discussed in more detail in Chapter 5, in which we present VLSI designs of DD-BMP decoders, along with an improved version of this algorithm (called "improved differential binary", or IDB).

Finally, various analog iterative decoders have also been proposed and built [73], [74]. As they operate in the analog domain, these decoders compute continuously and settle into a steady-state equilibrium as they converge, thus avoiding the unnecessary message exchanges that occur in digital decoders. In theory, these characteristics give them very good energy efficiency, potentially much better than digital decoders. However, they are considerably limited by practical factors, especially device mismatch and on-chip process variation. As such, block sizes longer than a few dozen bits are not currently practical. These analog decoders also have much lower throughput compared to digital decoders, in part due to their small block lengths.

2.5 Energy Consumption in Digital CMOS Circuits

In digital CMOS circuits, causes of energy consumption can be classified as either static or dynamic [75]. Static energy refers to constant leakage currents that are present even when the circuit is inactive, such as subthreshold conduction in switched-off transistors, electron tunnelling through transistor gates, and leakage

through reverse-biased diodes. Dynamic energy refers to energy dissipated when circuit nodes switch from one voltage level to another. This causes energy to be consumed charging parasitic capacitances, as well as through short-circuit currents caused when an input to a logic gate transitions.

Although static energy consumption has been observed to increase as a share of total energy at smaller manufacturing process sizes, dynamic energy is the dominant factor in LDPC decoder ICs. For instance, in [64], which presents a fully parallel split-row min-sum LDPC decoder in 65 nm CMOS, static power accounts for only 0.5-1% of the total power. Our own research projects, also fully parallel LDPC decoders implemented in 65 nm technology, affirm this number, with leakage accounting for 0.5-3%. Furthermore, recent developments in manufacturing processes have shown that gate leakage currents can be greatly reduced - [76] demonstrates 45 nm transistors with gate leakage currents reduced 2-3 orders of magnitude compared to 65 nm transistors. At even smaller process sizes, emerging devices like fin-FETs have proven successful in keeping leakage currents below practical limits [77] [78].

Leakage can also be controlled using standard circuit design techniques. Subthreshold currents, for instance, can be reduced by using transistors with high threshold voltages. Power gating is another effective technique for eliminating leakage currents in parts of the circuit that are not in use.

Since dynamic energy is the dominant factor, it is a much better target for reduction efforts than static energy. The dynamic energy of a circuit can be approximated by:

$$E_{\text{dyn}} = \sum_{\text{nodes}} \alpha_{\text{node}} C_{\text{node}} V_{\text{DD}}^2 \ , \tag{2.19}$$

Where α_{node}, the switching activity for a given circuit node, is defined as the number of 0-to-1 transitions (1-to-0 transitions are not considered to consume power because they do not draw any charge from the supply voltage), C_{node} is the total parasitic capacitance of the node, and V_{DD} is the supply voltage.

2.6 Energy Reduction Strategies for LDPC Decoders

When deciding how to approach the problem of making LDPC decoders more energy efficient, it is important to note that the most relevant metrics of performance for iterative decoders are *energy consumed per unit of information decoded* and *throughput per unit area* [79]. The reason for the former is that power consumption alone is not a meaningful metric, as it gives no indication of how much work is completed through that power use. Likewise for area and throughput, as nearly any figure for either can be achieved by using differing degrees of parallelism, or with multiple decoders in parallel. Therefore, these so-called "efficiency metrics" give a more meaningful assessment of performance, because they normalize the capacity to do computational work with the quantity of resources consumed in doing so.

Of course, these are not the only important LDPC decoder metrics. Practical applications also require certain standards for error correction performance (e.g., a maximum bit error rate for a given set of channel conditions) and decoding latency. A Gallager B decoder will be much more energy-efficient than an offset min-sum decoder, but won't find much use in a modern communication system due to its

poor error correction capability. As in any engineering problem, there are several contradicting design goals that must be balanced and kept within practical limits.

Based on our review of prior works, reducing computational complexity is one of the most effective ways to improve energy efficiency. The two concepts are actually closely linked, since reducing the amount of computational "work" done in decoding will naturally lead to a reduction in energy spent per decoded bit. In (2.19) above, this corresponds to a reduction in the number of circuit nodes over which summation is performed. This can be done without applying heuristics, as in the partially-parallel 10 Gbps Ethernet decoder of [60], or with heuristics, as in split-row min-sum [64]. However, in heuristic methods, the trade-off of error correction performance must be taken into account.

Another way to reduce energy consumption is to reduce switching activity and routing complexity, particularly in the inter-node communication network (commonly referred to as the interleaver). One reason for this is that the interleaver tends to have many long wires, crossing from one side of the die to another to connect distant nodes, and thus they have high parasitic capacitance and resistance causing energy loss. Routing congestion contributes to this problem, since it creates longer wires by two different mechanisms: larger die sizes, and fewer wires with direct connections. Some prior examples of wiring reduction are broadcasting [56], bit-serial [58], and binary message algorithms [24].

One final strategy is supply voltage scaling. Since dynamic energy consumption is proportional to V_{DD}^2, and in subthreshold conduction, the dominant component of static energy loss, current varies exponentially with V_{DD}, reducing the supply

voltage generally leads to increased energy efficiency - but only to a certain point. At low enough voltages (which are generally in the subthreshold region, when V_{DD} is lower than the device threshold voltage v_{t}), static energy consumption becomes dominant, as it scales more slowly with reduced V_{DD} than dynamic energy. Thus, past a certain point, lowering the supply voltage will actually increase the amount of energy consumed per operation.

Furthermore, supply voltage scaling results in a throughput trade-off, since CMOS circuits operate more slowly at reduced voltage. As noted in [58], highly parallel architectures can achieve a given throughput with a lower supply voltage than less parallel designs, making them good candidates for energy-efficient design. The partially-parallel 10 Gbps Ethernet decoder of [60] and split-row min-sum [64] are other examples of highly-parallel decoders that make use of voltage and frequency scaling.

2.7 Summary

This chapter has provided a summary of background information and prior work relevant to this dissertation. The background information included a description of LDPC codes, their matrix and graph definitions, and their most well-known decoding algorithms: the Gallager A and B algorithms, the SPA, and the MSA and variants. This was followed by a literature review of LDPC decoder architectures, beginning with the first VLSI designs, and ending with state-of-the-art energy-efficient designs. Finally, a brief primer on energy consumption and low-energy design techniques in digital CMOS circuits was given, along with strategies for energy-efficient LDPC decoder design that were used in designing the works presented herein.

CHAPTER 3
A Multi-Mode LDPC Decoder Interleaver Based On Transmission Gates

3.1 Introduction

One of the energy reduction strategies identified in the previous chapter is to reduce routing congestion. In highly parallel LDPC decoders, the interleaver (i.e., the message-passing network between the VNs and CNs) causes congested routing conditions, leading to many long wires and increased buffering requirements. These increase the amount of energy consumed, due to higher parasitic capacitances in longer wires, and of course the extra energy consumed by the buffers themselves [17]. Therefore, it is a good place to focus effort for an energy-efficient design.

As pointed out in [80], there is an immense volume of data exchanged over the interleaver. Let us consider the $(2048, 1723)$ 10 Gbps Ethernet LDPC code [81]. The graph of this code has 12288 edges, over which 2 messages are exchanged every iteration. With 4-bit messages, as in [60], 98304 bits are exchanged over the interleaver every iteration. If we assume an average of 5 iterations per decoded frame, that makes 491520 bits exchanged to decode a frame of 2048 bits. At 10 Gbps coded throughput, this comes to 2.4×10^{12} bits per second exchanged over the interleaver - quite a large amount of data. Reducing parasitic losses over these wires, even by a small amount, will therefore lead to considerable energy savings.

More complexity is added to the interleaver if the decoder must support multiple codes. Standards for wireless communication, notably WiMax [9] and Wi-Fi [10], are

designed to achieve good performance over a wide range of channel conditions. To that end, they define several different codes of varying rate and block length. Thus in a low-noise channel, for instance, a higher-rate code would be used to achieve higher information throughput, while in a noisy, congested channel, a lower-rate code would increase the probability of decoding success. Supporting multiple codes in a single decoder requires a flexible interleaver with configurable switching logic, in addition to the basic interconnections.

The aforementioned standards take interleaver complexity into consideration by using QC-LDPC codes.* As a result, partially-parallel architectures are well-suited to these codes, and achieve throughput per area superior to fully-parallel architectures [82] [83]. In addition, multiple code support can be easily achieved by adding the necessary permutations to the circular shift networks. For the reasons outlined in Section 2.6, a more highly parallel implementation of these codes might achieve superior energy efficiency. A parallel multi-mode decoder would face major implementation problems in routing congestion and shift network design, however.

The architecture proposed in this chapter aims to solve those problems by using tristate buffers and a bidirectional circular shift network architecture based on transmission gates. Inter-node messages are transferred over the same wires, and through the same shift networks, on alternating clock cycles. Because this design halves the required number of shifters and wires in the interconnection framework, the full advantages are realized in highly parallel decoder architectures, ideally with single-wire

* See Section 2.4.1 for a brief overview of QC-LDPC codes.

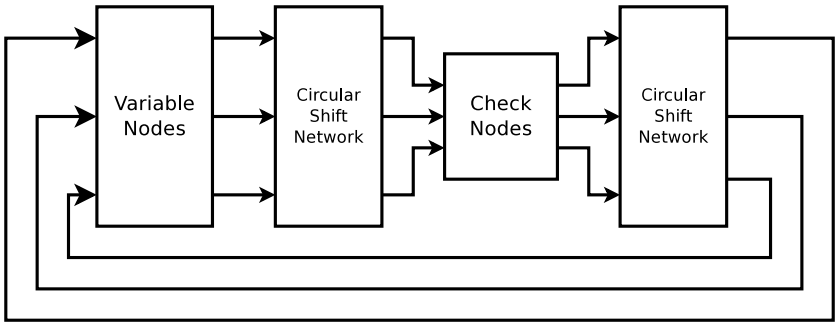


Figure 3–1: Canonical LDPC decoder architecture.

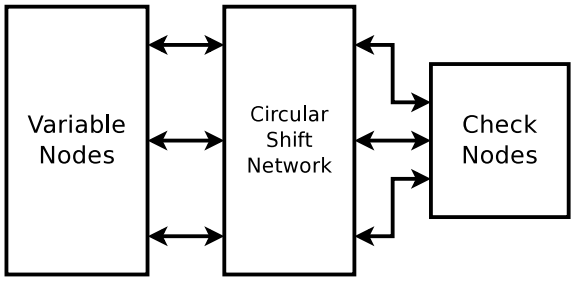


Figure 3–2: LDPC decoder architecture with a bidirectional interleaver.

message passing to minimize routing congestion. Thus, some decoding algorithms well-suited to this interleaver design are stochastic [71], differential binary [24], and bit-serial min-sum [58]. However, the bidirectional switch design is equally applicable to generic interleavers, and could be beneficial in any decoder design suffering from interleaver routing congestion.

3.2 LDPC Interleaving

A very simplified architecture for an LDPC decoder is shown in Fig. 3–1. Each variable node (VN) sends out messages to the check nodes (CN), with the circular shift network implementing the required interconnections. The CNs then generate the parity check result and send it back through another shift network, which reverses the shift performed by the first, thus sending the CN messages back to the originating

VNs. This basic architecture has undergone several modifications and improvements while maintaining the same basic functionality, such as with split-row decoding [63] and broadcasting [56].

In a QC decoder, the VNs and CNs are divided into sub-blocks, and all node interconnections consist solely of circular shifts local to the sub-block. Support for multiple codes (or, in a partially parallel design, multiple rows of the same code) is accomplished through the use of shift networks. These shift networks connect their inputs and outputs together in multiple combinations depending on a selection input. In the context of QC codes, these combinations consist of a range of circular shifts, which additionally may be around multiple block lengths. Hence, they are sometimes referred to as switch networks or circular shift networks.

Our bidirectional architecture, shown in Fig. 3–2, does not require the second set of shifters on the CN-to-VN path or the accompanying wiring. Instead, the internode-messages are driven over the same wires using tristate buffers. The shifters, which would ordinarily be implemented using standard CMOS muxes in the unidirectional design, are implemented using one-hot transmission gate muxes. This design requires 2 clock cycles to complete a decoding cycle, since the VN-to-CN and CN-to-VN message transfers can't take place at the same time. On the first clock of each decoding cycle, the VN-to-CN transfer occurs, and the result of the parity check is stored in flip-flops at the CNs. On the next clock cycle, the CN-to-VN transfer is driven back through the shifters.

The same mux selection vector is used for both cycles. Fig. 3–3 illustrates a 3-bit circular shift network made of transmission gates, which can shift 0 or 1 bits to

the left. When driven in "reverse" (CN-to-VN), the reverse of the shift performed in the "forward" (VN-to-CN) direction is applied. An alternative explanation is that since a transmission gate is logically equivalent to a switch, the shift network connects the appropriate VNs and CNs together on a single bus wire.

Although this interleaver architecture can be used with any LDPC decoder, we chose to implement it for a binary message passing decoder of the IEEE 802.11n codes for several reasons. The main reason is that this design is most advantageous for highly parallel decoders implementing multi-mode codes, since the most circular shift networks and wires will be saved. Binary message decoders, besides being practical to implement in fully parallel by virtue of their single-bit-width messages, also have very simple check nodes; the complete check node function for a given bit is the XOR of every other bit. This is equivalent to the sign update function seen in the check nodes of other LDPC decoding algorithms, including the sum-product and min-sum algorithms. A schematic for this function is shown in Fig. 3–4.

IEEE 802.11n was chosen for the code because it is a high profile multi-mode code, with a size and complexity suitable for our architecture. It has 3 codeword block lengths ($n = 648$, 1296, or 1944) and 4 rates for each length ($R = \frac{1}{2}$, $\frac{2}{3}$, $\frac{3}{4}$, or $\frac{5}{6}$), for a total of 12 modes.

Previous research in LDPC interleaving has been focused mainly on partially parallel or block serial architectures. Since these designs require circular shift networks supporting a large number of shifts and sub-block sizes, they are typically based on the well-known Benes network, as in [84], or barrel shifters, as in [85] and [86]. These are unsuitable for highly parallel decoders, which require a larger

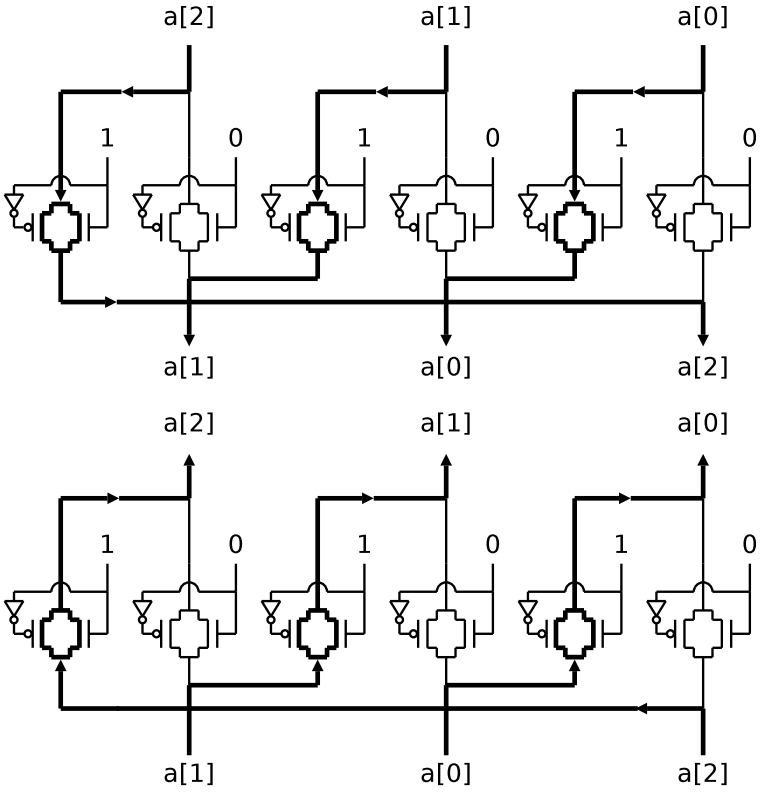


Figure 3–3: A circular shift network composed of transmission gates performs both the forward and reverse circular shift, depending on which direction the signal is driven.

number of less generalized shifters. Furthermore, these designs do not address the routing congestion problem that occurs in highly parallel decoders.

3.3 System Architecture

Our system design is divided into 12 separate row processors, so called because they each implement one row of $\mathbf{H}_{bm}$, where $\mathbf{H}_{bm}$ is defined as one row of $\mathbf{H}$ in permuation matrix form (as shown in Fig. 2–4 on p. 23). This is the minimum necessary to fully implement the IEEE 802.11n codes, as the rate $R = \frac{1}{2}$ codes require 12 check node blocks. Each of these row processors is multi-mode and implements

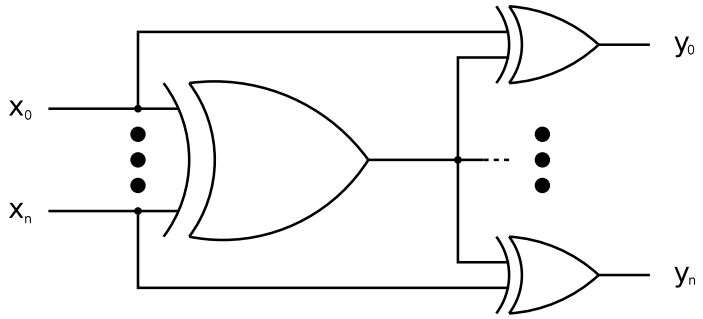


Figure 3–4: Schematic of a check node with n bits.

rows belonging to between 3 and 12 codes. Row 12 in our design implements the bottom rows from all 12 codes, while row 1 implements only the top rows of the 3 $R = \frac{1}{2}$ codes. The reason for this bottom-up superposition is to take advantage of commonalities in the right-hand side parity check portion of the $\mathbf{H}_{bm}$ matrices (referring back to Fig. 2–4, notice the dual-diagonal of zero entries on the right hand side - this structure is common to all 12 codes, so a bottom-up superposition completely eliminates the need for shift networks in that region).

Since each row processor is in parallel with and independent from the others, this section will focus on describing in detail the architecture of a single generalized row processor.

A block-level row processor schematic is shown in Figs. 3–5 and 3–6. Fig. 3–5 shows the VN-to-CN (forward) operation, while Fig. 3–6 illustrates the CN-to-VN (backward) operation.

Before decoding begins, the decoder controller must select one of the 12 supported modes using the *sel* input. This selection signal sets the code used in the

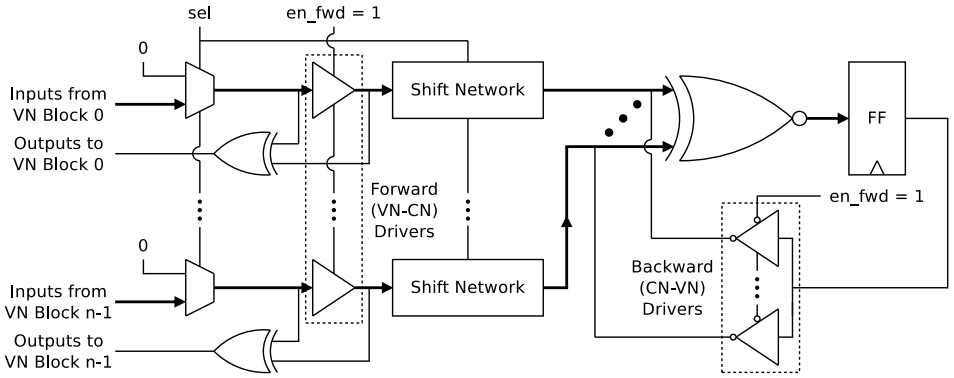


Figure 3–5: Top level block diagram of our bidirectional system, in VN-to-CN (forward) operation with the data flow highlighted.

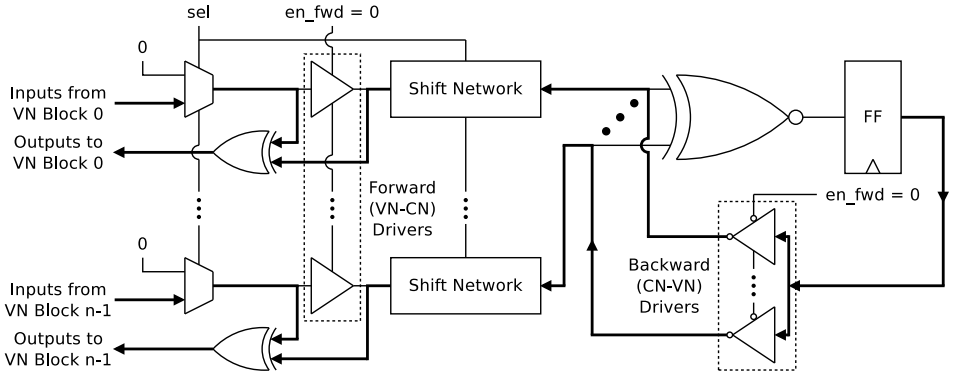


Figure 3–6: Top level block diagram of our bidirectional system, in CN-to-VN (reverse) operation with the data flow highlighted.

interleaver. It controls both the input muxes and the transmission gate shift networks.

The interleaver cycle begins with the *en_fwd* signal being set to logic 1, and the VN blocks transmitting their messages into the input at the left side of Fig 3–5. Because not all VN blocks connected to the row processor necessarily contribute to the parity check in every mode, there must be a way to disable the VNs in question or

prevent them from affecting the result. In a binary decoder, this can be accomplished by setting these bits to 0. This task is performed by the input muxes.

The width of the datapath originating at each VN block is equal to the sub-block size z of the current code, which can be 27, 54, or 81. Since each block is re-used in multiple modes, it must be able to support the sub-block size of the largest code it is used in. When a shorter code is active, the upper bits of each sub-block are disabled, along with the corresponding drivers and check nodes. The number of VN blocks contributing to each row processor is equal to the maximum row weight of the codes it implements. In this system, this number ranges between 10 and 22.

Next, the data flows through the VN-to-CN (forward) tristate drivers. Since *en_fwd* is set, the buffers are on and the data is driven onto the internal bus into the circular shift networks. The shift networks are of the form shown in Fig. 3–3. The shifted data is then input to the XNOR tree. The XNOR output constitutes the (inverted) parity result, which must now be de-shifted and output back to the VNs. The result is stored in a register of width equal to the largest sub-block size supported by the row. In this design, this width is $z_{max} = 81$ for all row processors.

On the next clock cycle, *en_fwd* is cleared and the CN-to-VN (reverse) path is activated as shown in Fig. 3–6. The parity result passes from the register into the CN-to-VN tristate driver bank, and from there into the "back" end of the shift networks. In order to ensure a high clock frequency, each wire on the bus is driven by a dedicated inverter. The shift networks perform the inverse of the shifts that they performed on the previous cycle, after which the data carries on to the 2-input XOR intrinsic bit filters. As shown in Fig. 3–4, the final step in the check

node computation is to take the XOR of the overall parity result with the bit that originated from each VN. The original bit is provided by the VNs themselves, which have been holding their state over the last clock cycle. These results serve as the interleaver output and are sent back to the VNs.

The internal structure of the transmission gate circular shift networks is essentially as shown in Fig. 3–3. Transmission gates with one-hot decoding connect every combination of front- and back-side wires needed to implement the required circular shifts. Redundant gates are merged, while unnecessary ones (those connecting wires that do not connect to any other transmission gates) are optimized out. The decoders themselves take *sel* as input and produce the one-hot control signals. The complementary signals are not required because our transmission gate standard cell includes an inverter.

3.4 Circuit Design

We synthesized the bidirectional interleaver system as described using the STMicroelectronics 90 nm CMOS design kit and standard cell library, augmented with a fully custom transmission gate cell. The schematic and layout views of the transmission gate cell are shown in Figures 3–7 and 3–8, respectively. The cell is sized to conform with the library cell grid and has dimensions 1.12µm × 3.92µm, for a total area of 4.39µm^2 - equal to 1 minimal-size NAND gate, or 1.33 minimal-size inverters. All transistors have the minimum length of 0.1µm. The transmission gate NMOS has a width of 0.32µm, while the PMOS has a width of 0.48µm. These sizes were chosen to be as wide as possible and thus minimize resistance, while having balanced propagation of rising and falling inputs.

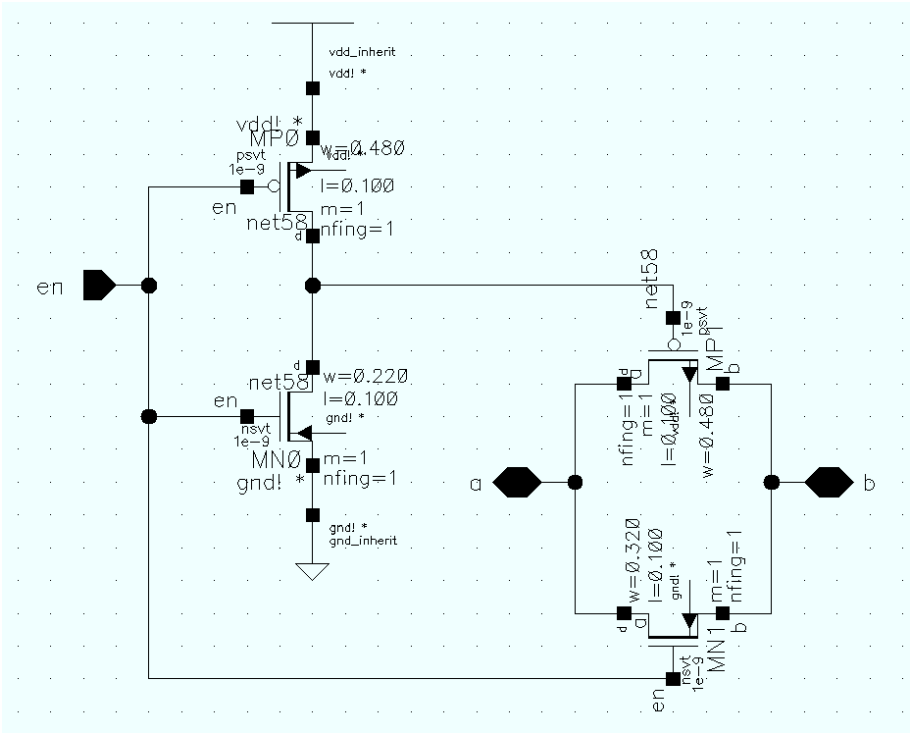


Figure 3–7: Schematic view of the transmission gate cell, showing transistor sizes.

In order to quantify improvements over the equivalent unidirectional interleaver, we also synthesized a reference system with seperate VN-to-CN and CN-to-VN shift networks as per the architecture shown in Fig. 3–1. The reference system uses standard CMOS muxes to implement the shift networks. It also does not have sequential elements and thus completes one decoding cycle per clock cycle. Otherwise it is equivalent to the bidirectional system - both use the same *sel* input and interface to the VNs. Both systems constitute a complete implementation of the interleaver and check nodes for a complete LDPC decoder using stochastic or differential binary algorithms.

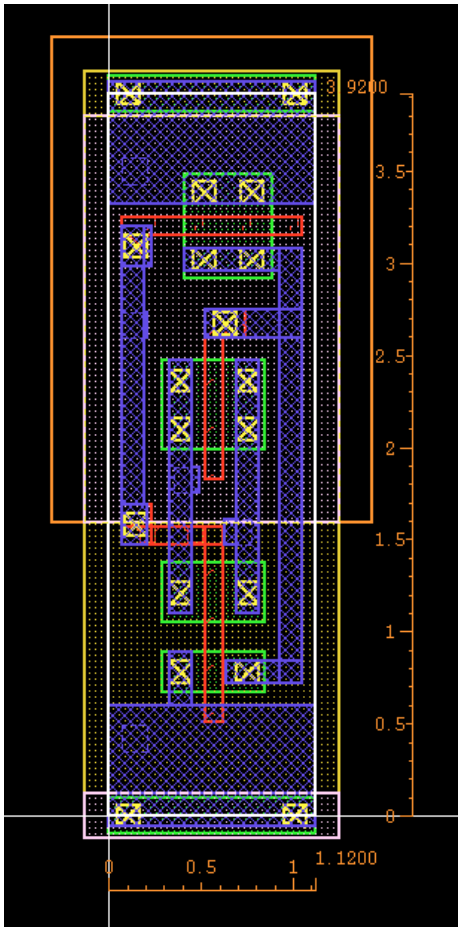


Figure 3–8: Layout of the transmission gate cell, with rulers showing the cell's overall dimensions (1.12µm × 3.92µm).

Synthesis results for both systems are summarized in Table 3–1. The bidirectional system is 28% smaller overall than the reference system for an equal throughput in terms of decoding cycles (DCs) per second. The area savings arises mainly from the elimination of the second shift network. The transmission gate mux architecture does not actually result in any savings compared to standard CMOS muxes, due to the need for additional decoding logic.

Table 3–1: Synthesis results

	Bidir. system	Reference system
Cell area (μm^2)	7.63×10^5	1.06×10^6
Gate equiv.	144.2k	199.7k
f_{clk}	729 MHz	364 MHz
Clocks per DC	2	1
DCs per second	3.64×10^8	3.64×10^8

The clock frequency of the bidirectional system was targetted to exactly double that of the reference system in order to compare area for equal throughputs. The frequency of the reference system is purely nominal; naturally additional area and speed tradeoffs can be made in both systems.

The cell area utilized by each category of logic for the bidirectional interleaver is shown in Table 3–2. Sequential elements account for only 3.3% of the total area. The shift network, and the tristates that drive it, account for more than half the total area. Note that the area figure for the transmission gates includes the inverters used to generate the complementary control signals. The remainder is made up of CMOS combinational logic.

The interleaver portion of the bidirectional system - consisting of the transmission gate shift networks, the decoders for the transmission gates, and the tristate drivers - has a total area of $3.98 \times 10^5 \mu m^2$, or 75.2k equivalent gates. In the reference system, the interleaver logic occupies an area of $7.20 \times 10^5 \mu m^2$, or 136.2k equivalent gates, giving an area reduction of 45%. The remainder of the size difference is accounted for by the flip-flops in the bidirectional system.

Table 3–2: Logic utilization

Cell type	Area (μm^2)
Transmission gates	1.86×10^5
Tristates	2.08×10^5
Flip-flops	2.47×10^4
All other logic	3.44×10^5

In order to ensure the integrity of the bidirectional datapath through synthesis, all bidirectional elements were instantiated directly as a cell-level netlist in verilog. This included the transmission gates as well as the tristate buffers. The standard, unidirectional CMOS portions of the circuit - the VN interfaces, decoders, XORs, and flip-flops - were synthesized normally. Timing in the bidirectional system was determined using a combination of static timing analysis and HSPICE simulations. For these simulations, the critical path and wire loading were extracted from the synthesis result and used to create a model circuit as shown in Fig. 3–9. The peripheral CMOS logic gates and slewing on the input signal were all imported from synthesis into this SPICE model. The effects of switching transients on the transmission gates are ignored because the selection inputs are constant during decoding. Net resistances are ignored because they are insignificant compared to the equivalent ohmic resistance of the active transmission gate [87].

The propagation delay through this path - from the input of one of the drivers, through the transmission gate, to the output of the opposite layer of static CMOS - was then measured, and the worst-case delay substituted back into the static timing result. This model circuit was also used to determine the strengths of the tristate drivers, with stronger drivers being assigned to the most heavily loaded paths. As

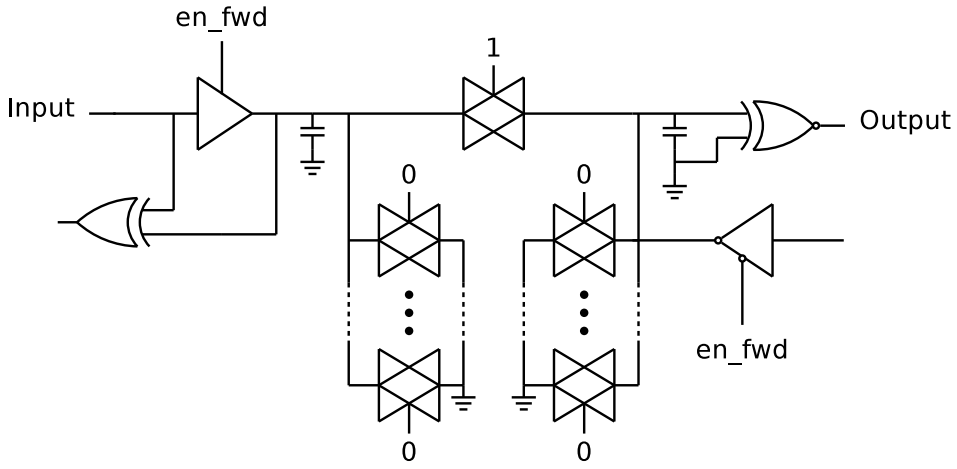


Figure 3–9: SPICE circuit used to determine timing.

expected, the critical path is on the "forward" path due to the extra levels of logic in the XNOR tree.

It should be noted that the bidirectional and reference systems are both capable of throughput several times greater than required for IEEE 802.11n. At the maximum net bit rate of 600 megabits per second and codeword size of 1944 bits, both systems can support over 1000 iterations per codeword, which is far greater than needed for any reasonable decoding algorithm. The respective maximum speeds of each system prove that the bidirectional interleaver does not incur any performance penalty relative to the reference system.

One disadvantage of the bidirectional interleaver is that it cannot be pipelined to increase throughput. The unidirectional architecture could therefore achieve higher throughput via pipelining. However, this would require an additional flip-flop for each variable and check node and thus incur a significant increase in area. Thus, the main advantage of this design is the reduction in area and routing, as this reduction

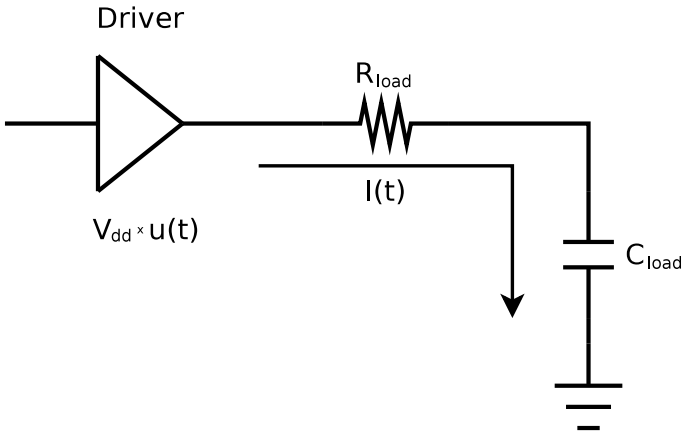


Figure 3–10: Interconnect model used to estimate energy consumption.

cannot be matched by partially parallel architectures without sacrificing a disproportionate amount of throughput. The primary reason for this is the shift networks in a partially parallel system must support more shift combinations, and no significant area savings occur until the architecture has been largely serialized. Furthermore, partially parallel systems incur extra overhead due to the additional memory and logic required to handle intermediate results.

3.5 Energy Analysis

While the bidirectional interleaver design has less logic and wiring overhead than the baseline unidirectional design, it requires tristate drivers, which consume more energy than ordinary buffers or inverters for an equal drive strength. The transmission gates also result in higher equivalent resistance on the interleaver wires. Therefore, it is not immediately clear whether the bidirectional system is more energy efficient than the unidirectional one.

In order to model the energy consumption compared to a unidirectional interleaver, we can use the interconnect model shown in Figure 3–10. In this model,

the wire resistance (and in the case of the bidirectional circuit, transmission gate resistance) are lumped into a single resistance R_{load}, while wire capacitance and gate capacitances of downstream logic are lumped into C_{load}. The resulting model is a simple series RC circuit.

We can now compare the energy required to charge C_{load} from zero voltage to V_{dd} in the two circuits. The energy stored in the capacitor is equal to $\left[\frac{1}{2} \cdot C_{load} V_{dd}^2\right]$. It can be shown that the energy dissipated by R_{load} is not dependent on its value, and is also equal to $\left[\frac{1}{2} \cdot C_{load} V_{dd}^2\right]$. The total energy expended is therefore $C_{load} V_{dd}^2$ - with V_{dd} constant, energy is entirely determined by C_{load}, which we can expect to be lower in the bidirectional circuit.

However, the lower C_{load} in the bidirectional circuit must be balanced against the higher internal energy dissipated in tristate buffers compared to normal buffers. In addition, the higher R_{load} caused by the transmission gates creates a need for tristate buffers with higher drive strength to achieve the same timing - higher drive strength translates directly to higher energy. This is also indirectly an energy efficiency issue, as we want fast transitions to minimize short-circuit energy losses in downstream logic.

The break-even point between driver type and average wire length can be found with the following:

$$L_{wire} = \frac{E_{tristate} - E_{buffer}}{C_L \cdot V_{dd}^2}, \tag{3.1}$$

where L_{wire} is the difference in average interleaver wire length, $E_{tristate}$ is the internal energy expended by a tristate buffer switching from 0 to 1 (i.e. the dynamic energy consumption caused by switching at circuit nodes internal to the cell), E_{buffer} is the internal energy of a conventional buffer, and C_L is the capacitance per unit length of wiring. Using the appropriate values from the STMicroelectronics datasheets [88] [89], we obtain $L_{wire} = 466$ µm, meaning the average interleaver wire must be 466 µm shorter in the bidirectional interleaver to break even on energy.

This amount of wiring savings is unlikely to occur, except in a very large chip. Furthermore, that figure gives only break-even energy - to obtain significant energy savings using this design, at least a few times that would be needed. Therefore, it can be said that the effectiveness of a transmission gate interleaver as an energy saving architecture is marginal. It is, however, more area-efficient than the equivalent interleaver using unidirectional logic, as we have already shown.

This assessment is supported by the results presented in [90], which describes a decoder of the 10GBASE-T LDPC code using a bidirectional interleaver. While this design is smaller than most contemporary decoders for this code, it is not the most energy efficient - detailed comparisons between this decoder and others can be found in Tables 5–1 and 5–4 in Chapter 5.

3.6 Summary

In this chapter, we presented a bidirectional interleaver architecture for LDPC decoders. We synthesized a fully parallel interleaver, including check nodes, for a stochastic LDPC decoder, implementing the 12 codes specified in the IEEE 802.11n draft standard. Due to the reduced amount of shuffling and interconnect framework

required in our design, our system is significantly smaller than the equivalent interleaver implemented in unidirectional CMOS logic. The complete system is 28% smaller than the reference system and achieves equivalent throughput. However, it is not as effective as an energy saving architecture, due to the increased energy demands of tristate drivers compared to their non-tristate equivalents.

CHAPTER 4
Pulse Width Modulated Min-Sum Decoding

4.1 Introduction

This chapter describes a new approach to low power iterative decoding called *pulse width modulated min-sum* (PWM-MS). In PWM-MS, messages are exchanged over single wires, with their magnitudes encoded using a single pulse of variable width. This technique has a number of advantages in the power domain: single wire messages reduce overall wire lengths and parasitic capacitances, it has significantly lower average switching activity in the interleaver compared to bit-serial min-sum, and it has very simple computational units.

While the variable node processing consists of simple addition, the check node processors represent a large part of the complexity in a min-sum iterative decoder. The reason for this is that each check node must compute the minima of all incoming extrinsic messages for each of its outputs. It is also possible to apply heuristics to the check node computation, reducing complexity at the cost of some error correction performance. Split-row min-sum [64], and the single minimum with correction technique for bit-serial min-sum in [58] are examples of this. Pulse width encoded messages, on the other hand, are naturally suited to min-sum decoding, as finding the minima of these messages is very simple, eliminating this source of complexity.

A VLSI circuit design of a PWM-MS decoder with a (660, 484) LDPC code in 0.13µm CMOS achieved throughput of 5.71 Gbps, an area of 5.76 mm^2, and energy

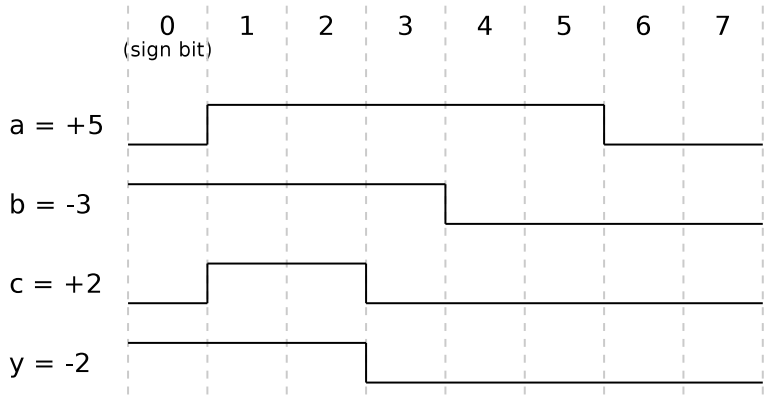


Figure 4–1: Some examples of pulse width encoded messages. The message y is the check node result produced from the inputs a, b, and c.

consumption of 65.8 pJ per information bit decoded at an SNR of 5.5 dB. This is roughly a 19% improvement in energy efficiency over the similar bit-serial min-sum (BS-MS) architecture [58].

This project was the subject of a journal article published in the November 2010 issue of IEEE Transactions on Circuits and Systems-II [26].

4.2 Pulse Width Message Encoding

In an iterative decoder, the switching activity generated by message passing has a particularly high priority for reduction. One reason for this is message passing represents a great volume of exchanged information, as mentioned in Chapter 3. Representing these messages in a power-efficient format can have a large impact on the circuit's overall power consumption [80].

In our proposed pulse width message encoding, the messages between the variable and check nodes are exchanged in sign-magnitude format, with the magnitude determined by the width of a digital pulse, and transferred over single wires. Figure

4–1 shows some examples of one possible pulse width encoding scheme, in which the first bit of each message is a sign bit. The subsequent bits indicate magnitude, with the signal held at logic 1 for a number of clock cycles equal to the magnitude. The length of a decoding iteration in clock cycles is thus equal the maximum message magnitude, plus 1 for the sign bit. Since this number can be set arbitrarily, it is possible to have a non-power-of-2 number of quantization levels.

One of the key motivations for using such a message encoding scheme is the very low switching activity. Using the encoding scheme shown in Figure 4–1, for instance, each message has a maximum of one 0-to-1 transition per decoding cycle. This low activity translates directly to low dynamic power consumption. Switching activity could be reduced even further using a transition-based encoding, in which the signal holds the same state as the sign bit, then transitions to indicate magnitude. However, this would result in a more complex CMOS circuit implementation as it requires transition-sensitive circuitry. For these works, we consider only the level-based scheme as indicated in Figure 4–1.

Since the activity factor varies as the decoding process proceeds, it is more relevant to characterize switching activity using the total transition count observed during decoding [80]. Figure 4–2 plots the simulated average number of transitions per edge in the Tanner graph per decoded codeword for pulse width min-sum (PWM-MS), pulse width offset min-sum (PWM-OMS), and bit-serial min-sum (BS-MS) to serve as comparison. This result shows that PWM has lower switching activity than BS-MS for all given values of the number of quantization bits q and SNR. The difference is relatively minor at $q = 4$ (about 5%), though at $q = 6$ this advantage

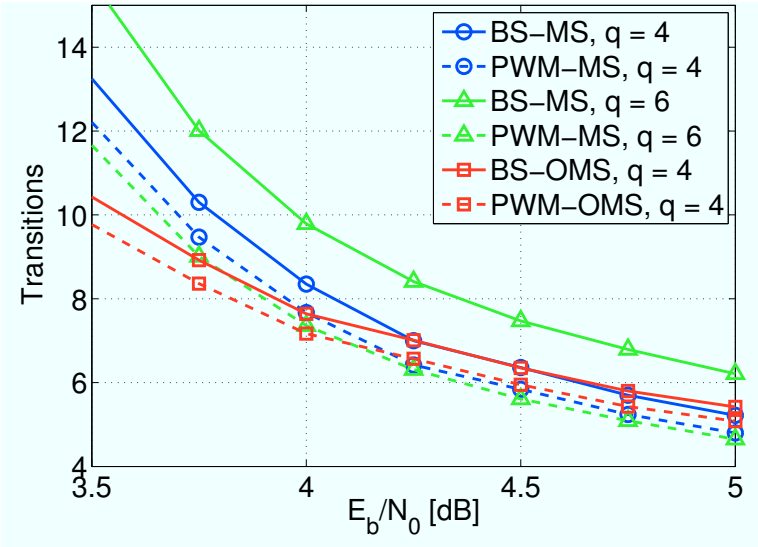


Figure 4–2: The average number of transitions per edge per decoded codeword, for the (660, 484) LDPC code used in this work.

increases to over 25%. This is because switching activity for PWM-MS remains roughly constant with q. In BS-MS, the average transition count increases with the number of bits. Results for offset min-sum (PWM-OMS and BS-OMS) are also plotted, with PWM-OMS showing a similar advantage.

4.3 Low Complexity Architecture of PWM-MS Decoders

Besides low switching activity, another major advantage of PWM-MS is that it results in a very low complexity check node. As shown in the check node schematic of Figure 4–3, the minimum of a group of messages can be computed with a single AND gate. The check node is fully combinatorial, so the CMOS implementation is very compact and power is further saved by the lack of sequential elements. The XOR gate network is used to determine the signs of the outgoing messages on each edge, while the AND gates determine the magnitudes in accordance with (2.16).

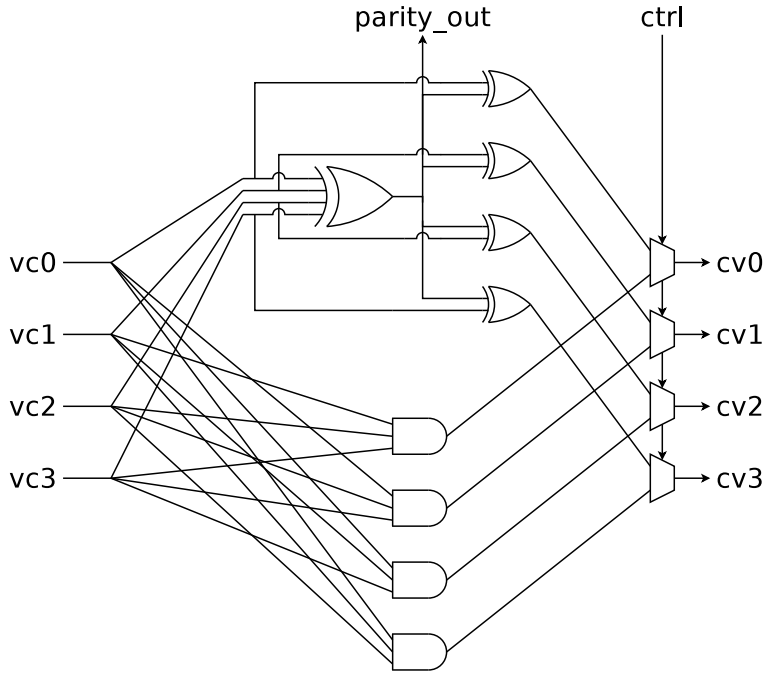


Figure 4–3: PWM-MS check node of degree $d_c = 4$.

A control signal *ctrl* selects the appropriate output during the sign and magnitude computation phases. A parity output is used for convergence detection and early termination logic.

Not only is this check node architecture very compact, it offers an exact implementation of the min-sum check node function. There is no performance loss resulting from the application of heuristics.

The PWM-MS variable node is shown in Figure 4–4. It consists of a modified up/down counter on each incoming edge, which converts the incoming PWM message to 2's complement binary. When *ctrl* is high, the up/down counters load their sign bits from their corresponding inputs, and reset their magnitudes to zero. Thereafter, they continue to count up or down for each clock cycle that their inputs remian

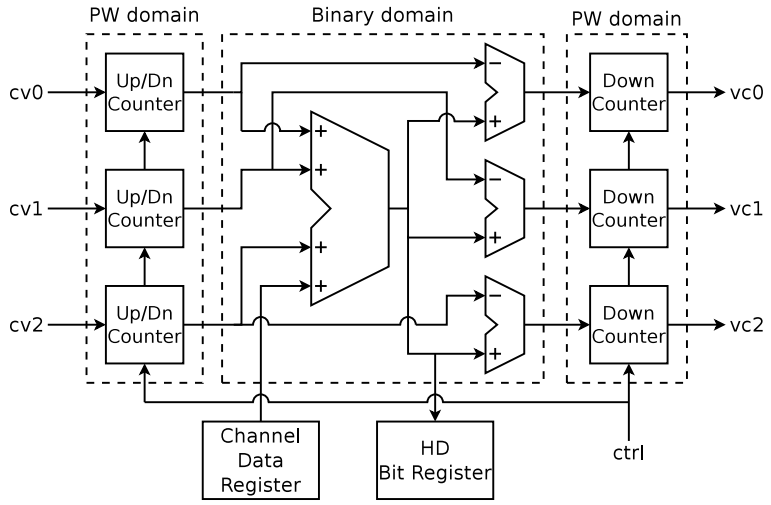


Figure 4–4: PWM-MS variable node of degree $d_v = 3$.

high. An adder network computes the hard decision bit estimation, as well as the outgoing messages on each edge. Down counters on each outgoing edge take the binary messages as input, and encode them in PWM format. The down counters load new values from the adder outputs when *ctrl* is high.

One decoding iteration has a latency of $2^q - 1$ clock cycles, where q is the message quantization width in bits. On the first cycle of every iteration, nodes compute and exchange the sign bits of their respective messages. The sequence of computations on this cycle begins in the VNs. The global control signal *ctrl* goes active, which loads the magnitudes of the $m_{v\to c}^{(k)}$ messages and passes the sign bits through to the VN outputs. In the CNs, the signs of the $m_{c\to v}^{(k)}$ messages are computed in the XOR gate network. Back in the VNs, these sign bits are stored in the up/down counters, which are also reset to zero by *ctrl*. On the $2^q - 2$ subsequent clock cycles, the message magnitudes are exchanged. The VN down counters count down internally, with their outputs transitioning from 1 to 0 when their count reaches zero. In the

CNs, the AND gates find the minima of their incoming messages, as the first input that transitions to 0 causes the output to transition to 0 as well. Back in the VNs, the up/down counters count up or down based on their stored sign bit as long as the incoming signal remains at 1. When it transitions to 0, the counter holds its current value. At this point, the value stored in the up/down counter is the final $m_{c\to v}^{(k)}$ message of the corresponding edge. To conserve power by preventing unnecesary switching in the VN adder networks, the up/down counter output updates only on the first clock cycle of every iteration, when *ctrl* is active. It is set to 0 otherwise. After $2^q - 1$ cycles, *ctrl* goes active again and the next decoding iteration begins.

One notable property of this variable node design is that it can be used to implement offset min-sum for a negligible increase in complexity, as opposed to other decoder architectures including bit-serial. This is because the sign-magnitude format of PWM allows the offset to be efficiently applied in the variable node. The offset is implemented by setting the output counters to transition when their internal count reaches the OMS offset value, β, rather than zero. Furthermore, applying an offset reduces the maximum message magnitude. With pulse-width message encoding, this reduces the number of clock cycles per iteration, and increases throughput accordingly. For instance, with 4-bit quantization and an offset of 1, the maximum message magnitude decreases from 7 to 6, and the number of clock cycles per iteration from 8 to 7, increasing throughput by 12.5%. It is therefore very advantageous to use PWM-OMS over PWM-MS.

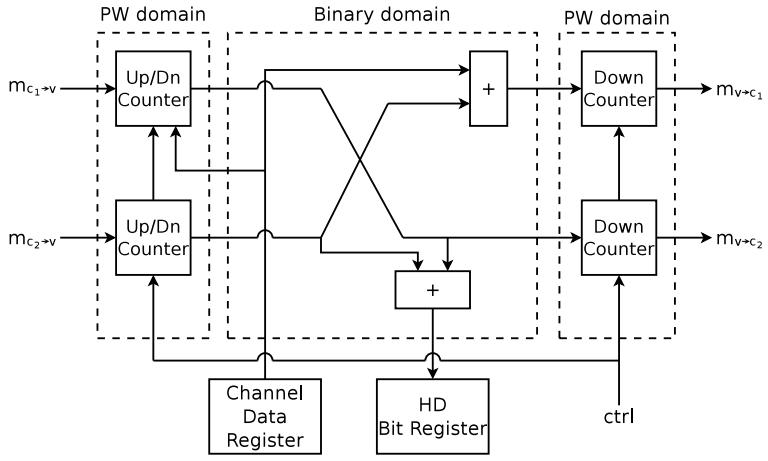


Figure 4–5: Special architecture for PWM-MS variable node with degree $d_v = 2$.

4.3.1 Degree-2 Variable Nodes

For LDPC codes with degree-2 VNs, such as the Wi-Fi and WiMax standard codes, the special VN architecture shown in Figure 4–5 can be used. In this architecture, one of the up/down counters is initialized to L_v, the initial channel data, rather than zero. Fewer adders are then needed to compute the outgoing messages and hard decision bit. However, the up/down counter that initializes to L_v requires an additional bit to accommodate the larger values it can now reach. This architecture attains 5-10% lower area in ASIC synthesis than a degree-2 VN using the "conventional" architecture shown in Figure 4–4.

4.3.2 Augmented Early Termination Check

Decoders with many low-degree VNs also include additional circuitry to perform an augmented early termination check. This is necessary because low-degree nodes have a high likelihood of the hard decision bit h_v differing from the signs of the outgoing messages, leading to cases where all parity checks are satisfied, but the

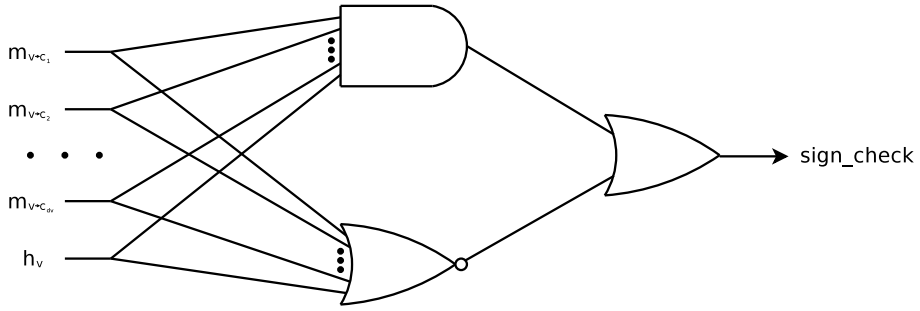


Figure 4–6: Sign-checking circuit for PWM-MS variable nodes.

decoded bits **h** contain errors. These cases arise because the initial channel data, L_v, participates in the calculation of h_v, but does not participate in the calculation of the outgoing messages.

These cases can be prevented by adding the circuit shown in Figure 4–6 to each VN. This circuit detects whether or not h_v is in agreement with the outgoing messages during the sign computation phase of each iteration (this is also when h_v is updated). If it is not, the parity checks used for early termination are unreliable, and so early termination is augmented using these sign checks in the VNs. All sign checks as well as parity checks must be satisfied for early termination to occur.

In theory, it is possible for these "false terminations" to occur with any degree of VN, depending on the exact values of d_v and q. However, in our simulations, they were only observed occurring in degree-2 VNs. For codes where $\min(d_v) \geq 4$, these events are vanishingly rare and so the sign check circuitry is not necessary.

4.4 Design Results

We designed PWM-OMS decoders in three different configurations using the IBM CMRF8SF standard-V_t 0.13 µm CMOS design kit to obtain estimates for silicon area, throughput, and power consumption. We also designed PWM-OMS and

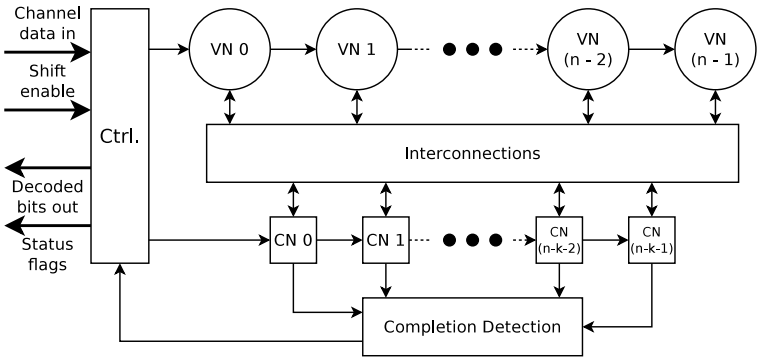


Figure 4–7: Block diagram of the designed decoder.

PWM-MS decoder prototypes on a Xilinx Virtex-5 LX330 FPGA for verification and BER/FER measurements. In all cases, the LDPC code used is a (660, 484) regular (4, 15) progressive edge growth based LDPC code. This code has also been used in [58] for verifying the bit-serial approach. This section contains discussion of these results, as well as comparisons with other decoders.

Figure 4–7 shows a top-level block diagram of the designed decoder. It is a fully node-parallel architecture, and interconnections between the nodes are each a single wire. Early termination logic uses parity checks from the check nodes to detect convergence. A state machine *Ctrl* generates control signals and acts as the off-chip interface.

We designed PWM-OMS decoders in both unpipelined and pipelined configurations. Figure 4–8 shows a more detailed block diagram of a single VN and CN pair, which illustrates the difference between the unpipelined and pipelined versions. The critical path of the unpipelined decoders is traced in red, while the critical path for the pipelined version is traced in blue. In the unpipelined decoders, the DCs are bypassed during the sign computation phase. This allows the new signs to be

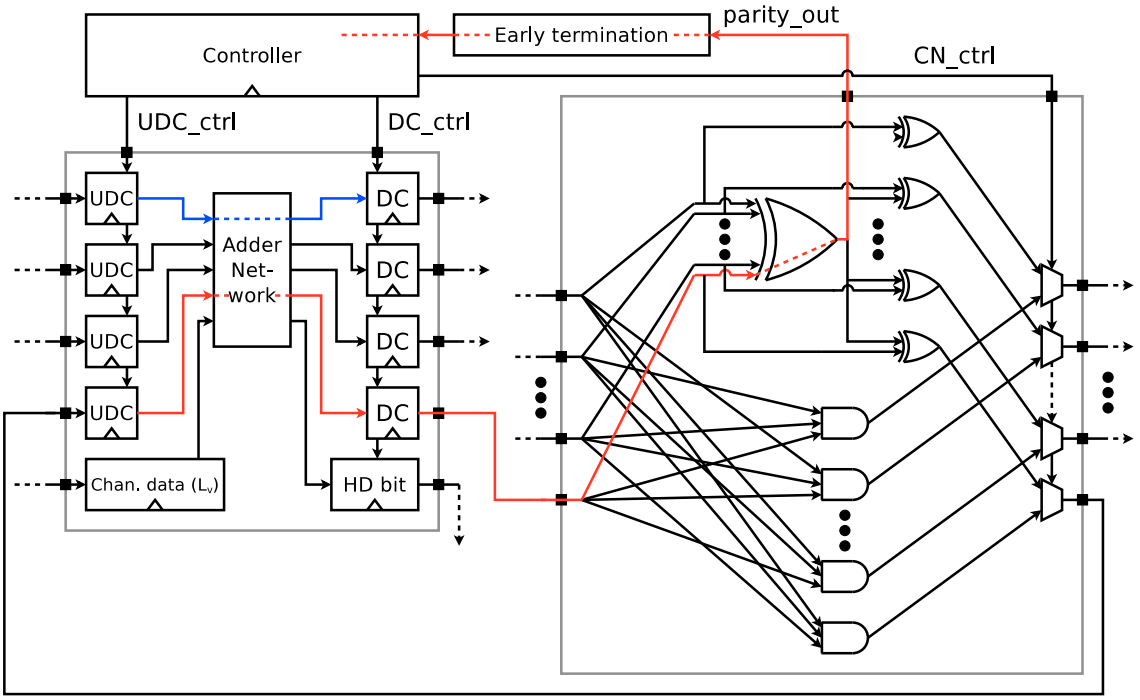
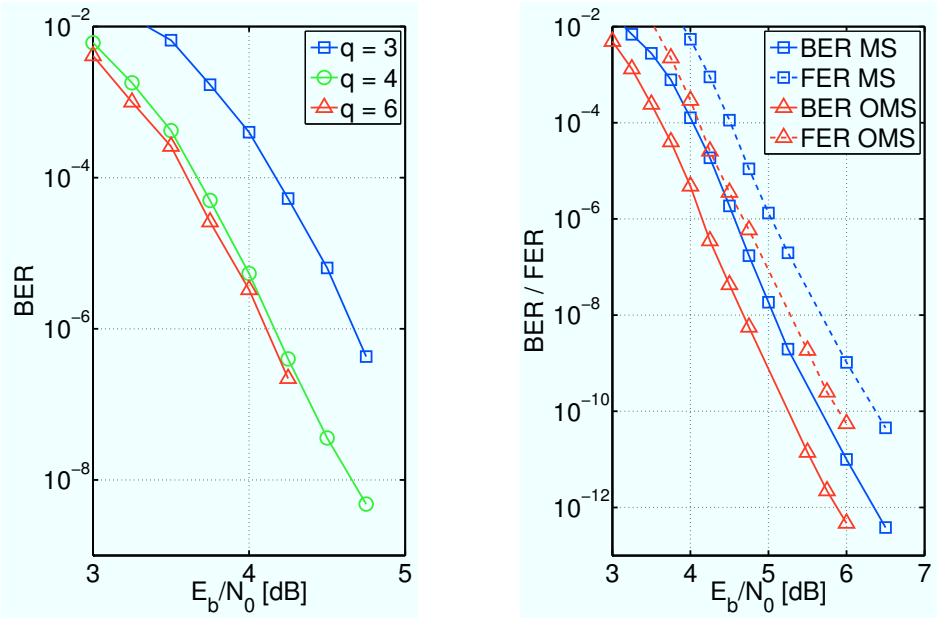


Figure 4–8: Detailed block diagram of the PWM-MS decoders showing the critical paths of the unpipelined decoders (in red) and the pipelined decoder (in blue).

calculated in the CNs and registered in the UDCs on the same clock cycle. However, the critical path is not from one UDC to a UDC in another VN, but rather from a UDC through the early termination block to the controller. In the pipelined decoder design, the DCs are not bypassed. Thus, during the sign computation phase, the new sign does not reach the UDCs until the next clock cycle - this causes the pipelined decoders to have one extra clock cycle of latency per iteration. This also necessitates the UDC and DC control signals to be separate. In the unpipelined decoders, the UDC_ctrl, DC_ctrl, and CN_ctrl signals are all equivalent. However, in the pipelined decoders, UDC_ctrl and CN_ctrl are equivalent, while DC_ctrl is



(a) (b)

Figure 4–9: Decoding performance for different quantization levels of offset min-sum with an offset of 1 (a), and for 4-bit PWM-MS and PWM-OMS decoders implemented on FPGA (b).

equal to UDC_ctrl time-delayed by 1 clock cycle. In the pipelined case, the critical path is completely internal to the VN, and runs from the UDC to the DC.

The impact of the number of quantization bits on the BER performance of PWM-OMS is shown in Figure 4–9a. In terms of the number of quantization bits, $q = 4$ results in a very small performance loss compared to $q = 6$, while $q = 3$ incurs another loss of 0.4-0.5 dB. Figure 4–9b shows the BER and FER performance of the FPGA decoder prototypes. For this code, OMS gives a performance gain of 0.4-0.5 dB over conventional min-sum. All BER tests use a maximum of 15 decoding

iterations. These FPGA prototype results exactly match results obtained with a software model. These performance tests used an emulated Gaussian channel [91].

Table 4–1 shows ASIC post-layout design results for our decoder in 0.13µm CMOS, alongside the bit-serial approximate min-sum decoder in [58]. Comparisons between our decoder and this one are particularly relevant, as they both use the same process size, supply voltage, and LDPC code. They also have architectural similarities - both are fully node-parallel with single-wire node interconnections, and both were designed primarily for low-power applications. Throughput and power for our designs were calculated using post-layout simulations, with back-annotated delays and wiring parasitics, and includes the clock trees.

We designed 3 configurations of PWM-OMS decoders. Configuration A is optimized for decoding performance. Configuration B is optimized for energy efficiency, while configuration C is optimized for throughput.

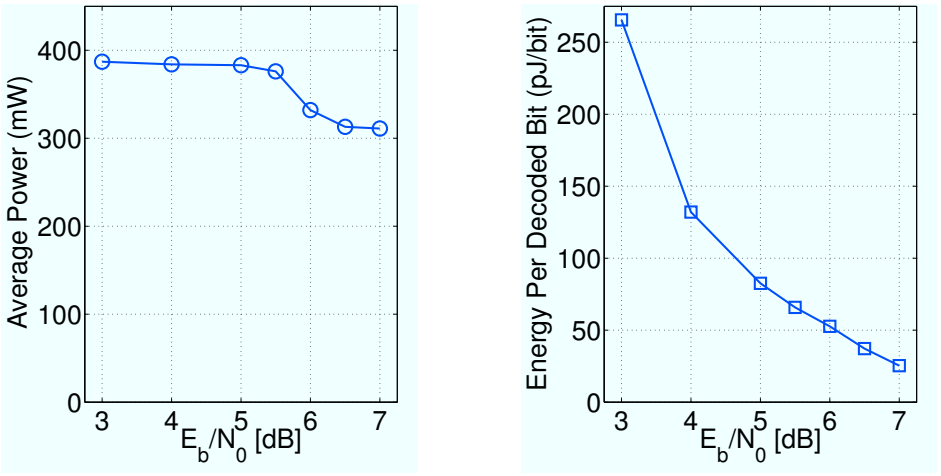
Since our decoder architecture uses OMS and no heuristics in the check node, configuration A achieves a coding gain of approximately 0.5 dB over conventional 4-bit min-sum with a maximum of 15 iterations, as shown in Figure 4–9b. On the other hand, [58] incurs a loss of 0.1 dB due to a check node approximation. We note that with $q = 3$, PWM-OMS can achieve approximately the same BER performance as the 4-bit approximate min-sum used in [58]. Hence, in configurations B and C, we trade off this 0.6 dB coding gain for reduced area, higher throughput, and better energy efficiency.

Configuration A has 7 clock cycles per decoding iteration. In standard min-sum, 8 cycles would be required - 1 for the message sign, plus 7 for the 3 bits of

Table 4–1: Post-Layout Results and Comparison With Bit-Serial Min-Sum

	This work			[58]
Architecture	PWM-OMS			Bit-serial approx. MS
LDPC code	(660, 484)			(660, 484)
Process and supply voltage	0.13µm / 1.2 V			0.13µm / 1.2 V
Configuration	A	B	C	n/a
Quantization width [bits]	4	3	3	4
Pipelining	no	no	yes	yes
Clock cycles / iteration	7	3	4	4
Clock frequency [MHz]	150	150	250	300
Core area [mm^2]	5.76	4.50	4.54	7.3
Cell area [mm^2]	4.24	3.31	3.30	5.26
Gate equivalent [Kgates]	556	434	433	690
Max. iterations	15			15
Coding gain [dB][a]	0.5	-0.1	-0.1	-0.1
Avg. information throughput (5.5 dB) [Gbps]	5.71	9.18	12.2	17.1
Avg. power (5.5 dB) [mW]	376	465	819	1383
Avg. energy per decoded bit (5.5 dB) [pJ/bit]	65.9	50.7	67.1	80.9
T/P per unit area [Gbps / mm^2]	0.99	2.04	2.69	2.34
T/P per unit area per unit power [Gbps / ($mm^2 \cdot$ W)]	2.64	4.39	3.28	1.69

[a] Relative to conventional min-sum with $q = 4$ and 15 max. iterations.



(a) (b)

Figure 4–10: Average power consumption (a) and energy efficiency (b) for a PWM-OMS decoder with $q = 4$.

magnitude. However, since we implement offset min-sum, we apply an offset of 1, reducing the maximum magnitude to 6. Likewise, configuration B takes only 3 cycles per decoding iteration (1 for the sign, and 2 for the magnitude). Pipelining adds a cycle of latency, so configuration C requires 4 cycles.

Area is reduced relative to bit-serial min-sum due to the simpler check node. We define average throughput as the throughput achieved by immediately beginning decoding of a new codeword once the current one has converged. Average throughput is therefore determined by the average number of iterations required for convergence, and varies for different values of SNR. Raw throughput is lower, due to both the lower clock frequencies and (in the case of configuration A) higher number of clock cycles per iteration. In terms of throughput per unit area, however, configuration C is 15% higher than [58]. We also define another metric that takes power consumption into

account: throughput per unit area per unit power. In these terms, all 3 PWM-OMS designs are superior to [58]. Energy efficiency, defined in terms of energy consumed per decoded bit, is also better in PWM-OMS. Plots of average power consumption and energy efficiency for the A decoder over a range of SNR values are shown in Figure 4–10. Energy efficiency was determined by dividing average power by the average throughput at each given value of SNR.

The complexity and energy efficiency of PWM-OMS also compare favourably with other recent LDPC decoders, although differences in the process size and LDPC code make direct comparisons difficult. One example is the partially parallel OMS decoder in [60], which implements a (2048, 1723) Reed-Solomon based code in 65 nm CMOS. It achieves an average information throughput of 47.7 Gbps at 5.5 dB SNR, with power of 2800 mW and energy of 58.7 pJ/bit.

Another recent architecture is the 90 nm fully node- and bit-parallel min-sum decoder in [92], which achieves an information throughput of 13.21 Gbps, with an average power of 1323 mW and energy of 98 pJ/bit at 5 dB SNR. It should be noted that these figures are without early termination, which was used to greatly increase throughput and decrease energy in [58], [60], and this work. But since [92] partitions long inter-node wires with registers, early termination would be less effective due to added complexity and increased latency. In addition, this architecture requires parallel check nodes, which become very complex at high degrees, whereas in this work the check node complexity is extremely low, even at high degrees.

If we assume energy consumption per information bit scales quadratically with feature size, the scaled energy of [60] is 279 pJ/bit and [92] is 204 pJ/bit, giving this work respective improvements of 76% and 68%.

4.4.1 Energy Impact of Loading and Interfacing

One additional consideration to make with these results is that the figures for power and energy do not include any I/O buffering, and assume that all channel data L_v and hard decision bits h_v are loaded and unloaded in parallel. This is an important distinction between fabricated decoders, such as [58], and those that report results based on post-layout simulations, such as this work. Fabricated decoders must necessarily include I/O buffers and wire bond pads to provide an off-chip interface, which will consume additional energy.

In addition, if test data generation and result evaluation are not included on the fabricated die, L_v and h_v must be loaded and unloaded from off-chip test equipment. Doing so in a fully parallel manner would require a prohibitively large number of I/O ports. For highly parallel decoder designs, the most common solution to this is to connect the L_v and h_v registers together as shift registers, as this technique requires little hardware overhead. The data is then shifted in and out of the chip in a semi-parallel or serial manner. This consumes more energy than parallel loading, because of large amounts of switching activity in these shift registers, and also because this loading and unloading takes more time, and so leakage losses are greater. Therefore, in addition to the inherent uncertainty of post-layout energy consumption as compared to fabricated silicon, the impact of I/O buffers and data loading must be considered. This section will examine those factors.

If test data and hard decision bits are loaded from off-chip, each decoded bit requires $q = 4$ bits loaded into the chip and 1 bit loaded out. Using dynamic energy consumption tables from the standard cell library datasheets [93], this adds 0.786 pJ/bit from the core power supply.* This amounts to an increase of 1.2%.

For calculating the energy cost of non-fully-parallel loading and unloading, we assume all L_v registers begin cleared, h_v registers are loaded with zeroes at the tail of the chain, and both L_v and h_v are random data with an average activity factor of 0.5. We also assume w output pins for offloading h_v, and $q \cdot w$ input pins for loading L_v. Thus, loading and unloading require $\lceil \frac{n}{w} \rceil$ clock cycles (recalling that the code block length n is 660 in this case).

With these parameters, serial loading (i.e., $w = 1$) has a total energy cost of 74.2 pJ/bit - more than doubling the energy consumed by decoding. As mentioned before, this high figure can be attributed to the long period of high switching activity as L_v and h_v are shifted through their respective registers. If we use the same number of input pins as [58] (44 pins, meaning $w = 11$), then loading adds a much more manageable 6.73 pJ/bit. Together, loading and I/O add 7.52 pJ/bit, making the total energy consumption 73.4 pJ/bit - still an approximately 9% improvement over [58],

* The pad sides of I/O buffers are driven from a separate, generally higher-voltage power supply with very strong drivers relative to anything found in the chip core, since these are designed to drive signals across circuit board traces. Consequently, pad-side energy consumption is much higher than the core side. However, it is reasonable to ignore pad-side energy for our purposes, since it originates from a separate supply, and is dependent on the chip package and test equipment. Other designs in the literature do likewise.

but significantly diminished from our earlier figure of 19%. From this result, we can conclude that loading presents a significant energy cost, and that an energy-efficient design should use as high of a degree of parallelism as the number of I/O pins will allow.

Due to the degree of uncertainty of post-layout energy estimation, it is possible that our PWM-OMS designs actually consume more energy than BS-MS, particularly if we take loading and I/O into account. However, the standard in the literature for non-fabricated LDPC decoders is to report "core-only" power and energy based off post-layout back-annotated simulations and ignore loading and I/O, such as in [64] and [94]. One argument in favor of this measure is that an LDPC decoder would not be placed on a separate die in practical applications, but rather as part of an SoC, so I/O is irrelevant, and any necessary deserialization is not a core part of the decoder. In addition, some fabricated LDPC decoders perform test pattern generation and BER/FER monitoring on-die, and use external I/O pins only to set parameters and read status signals, such as [60]. Because of these reasons, and to maintain consistent reporting standards with previous works, we therefore ignore deserialization, non-fully-parallel loading, and I/O in our reported energy and power results.

4.5 Summary

In this section, we presented a new iterative decoding technique called pulse width modulated min-sum (PWM-MS), and implemented it for a (660, 484) regular (4, 15) LDPC code. The advantages of this architecture include very low complexity check nodes, and low message exchange switching activity. Our post-layout VLSI

design of a decoder with 4 bits of message quantization achieved a core area of 5.76 mm^2, and an average information throughput of 5.71 Gbps at 5.5 dB SNR. It also achieved an energy efficiency of 65.9 pJ/bit at this SNR, and a coding gain of 0.5 dB over conventional min-sum owing to the use of the offset min-sum algorithm. This coding gain can be traded off for additional improvements in area, throughput, and energy by reducing the number of quantization levels. These results for area and energy represent respective improvements of 21% and 19% compared to the similar bit-serial min-sum decoder architecture. These results also compare favourably with other recent LDPC decoder architectures.

CHAPTER 5
Decoders Based On Differential Binary Message Passing Algorithms

5.1 Introduction

This chapter presents LDPC decoder designs using differential binary (DB) algorithms, such as the *differential decoding with binary message passing* (DD-BMP) algorithm.

While DD-BMP has been shown to achieve error correction performance equal or superior to MSA when used with LDPC codes based on finite geometries (FG-LDPC codes) [95], it has never before been analyzed in hardware. At the system level, this algorithm has several qualities that are useful in energy-efficient designs. The variable and check nodes are both very simple, which translates to lower area and power consumption in VLSI chips. Furthermore, binary message exchange that is highly amenable to broadcasting means wiring overhead is very low, and since the signs of the memories rarely flip, switching activity is low as well. Finally, DD-BMP has a low average number of iterations required for convergence to a valid codeword, giving it a very high average throughput. Voltage and frequency scaling (VFS) can be applied, trading off throughput for reduced dynamic energy consumption.

Furthermore, FG-LDPC codes, while very powerful, are also highly complex, which has been an obstacle to their widespread use [96]. In this project, we revise the DD-BMP algorithm to make it more amenable to VLSI implementation, and design fully parallel decoders using DD-BMP and modified DD-BMP (MDD-BMP) in 65

nm CMOS for $(273, 191)$, $(1023, 781)$, and $(4095, 3367)$ FG-LDPC codes. We show that these algorithms overcome the implementation complexity of FG-LDPC codes, and demonstrate low area, high throughput, and high energy efficiency competitive with other state-of-the-art decoder designs.

We also present a new decoding algorithm called IDB, for "improved differential binary" message passing, which is suitable for decoding LDPC codes other than FG codes. It was noted in [24] that DD-BMP exhibits poor error correction performance with non-FG codes. In this work, we find that this poor performance is due to trapping sets [97] and propose two algorithmic modifications to overcome them: degeneration, and relaunching. We then present a fully parallel IDB decoder design for the $(2048, 1723)$ Reed-Solomon-based LDPC code specified in the IEEE 802.3an (10GBASE-T) standard [13], and show that it achieves significant improvements in area, throughput, and energy efficiency over other decoders with similar error correction performance.

5.2 FG-LDPC Codes and the DD-BMP Algorithm

LDPC codes are classified into several types based on the method used to construct them. One of these types is based on Euclidean (EG) and projective (PG) geometries over finite fields, collectively known as finite geometric (FG) codes [95]. These codes are known to have excellent error correction performance and high minimum distances. In addition, their encoding can be performed in linear time with simple feedback shift registers. However, FG-LDPC codes have a much higher number of edges in their Tanner graph representations and higher degree nodes than many other types of LDPC codes. The $(273, 191)$, $(1023, 781)$, and $(4095, 3367)$

FG-LDPC codes used in this work have variable and check node degrees (d_v, d_c) of $(17, 17)$, $(32, 32)$ and $(64, 64)$ respectively. This high complexity has been an obstacle to their widespread use, as noted in the hybrid soft bit-flipping decoder presented in [96]. This implementation is highly routing-limited, even with a partially parallel architecture. With DD-BMP and its variants, however, we have produced efficient VLSI designs of FG-LDPC codes, demonstrating that these powerful codes are not too complex for widespread application. It should also be noted that DD-BMP is not limited to use with FG-LDPC codes, as it has also been found to perform well with any code with high row and column weights [24].

The DD-BMP algorithm implemented in this work is based on [24]. A number of changes have been made to make it more amenable to VLSI implementation. In particular, a random process used during initialization has been eliminated, the hard decision bits are not functions of their previous values, and convergence detection is performed with parity checks as in [58] and [26], rather than a syndrome check.

First, the log-likelihood ratios (LLRs) L_v of the received bits are read from the channel:

$$L_v = \ln \left[\frac{P(x_v = 0|y_v)}{P(x_v = 1|y_v)} \right] , \tag{5.1}$$

where x_v is the vth transmitted bit, and y_v is the vth value received from the channel.

Next, the variable nodes are initialized with quantized LLRs:

$$M_{v,c}^{(0)} = \begin{cases} -2^{q-1}, \text{ if } L_v \leq -T_{clip} \\ 2^{q-1} - 1, \text{ if } L_v \geq T_{clip} \\ \left\lfloor \dfrac{L_v}{\left(\dfrac{T_{clip}}{2^{q-1}}\right)} \right\rfloor, \text{ if otherwise.} \end{cases} \tag{5.2}$$

where $M_{v,c}$ is the memory corresponding to the edge between variable node v and check node c, T_{clip} is the LLR clipping threshold, and q is the number of bits of quantization. For the decoder designs presented in this paper, we use $T_{clip} = 10.5$ and $q = 6$, and thus our quantization step is 0.328125, giving the quantized values the equivalent of 4.39 integer bits (including sign) and 1.61 fractional bits. However, for the sake of simplicity, quantized values are treated internally as q-bit signed integers as per Equation 5.2, and thus differ from the "true" LLR values by a scaling factor equal to the quantization step. This value of T_{clip} is optimized for the decoder designs presented in this chapter, and offers a slight improvement in error rate performance and convergence speed compared to a conventional fixed-point number representation with an integer number of bits representing the magnitude and sign.

The variable-to-check messages are determined by the signs of the values in the corresponding memories:

$$b_{v\to c}^{(k)} = \text{sgn}_r\left(M_{v,c}^{(k)}\right), \tag{5.3}$$

where $b_{v\to c}^{(k)}$ is the single-bit message from variable node v to check node c at iteration k, with k being initially set to 0, and sgn_r is a sign function modified so that $\text{sgn}_r(0) = 1$.

The check node operation consists of:

$$b_{c\to v}^{(k)} = \prod_{v'\in V_j\setminus v} \operatorname{sgn}_r\left(b_{v'\to c}^{(k)}\right), \tag{5.4}$$

where $b_{c\to v}^{(k)}$ is the single-bit message from check node c to variable node v at iteration k, and V_j is the set of variable nodes incident to check node c.

At this point, the iteration count k is incremented and the variable node operation updates the memories $M_{v,c}$ as follows:

$$M_{v,c}^{(k)} = M_{v,c}^{(k-1)} + \left\lfloor s \cdot \sum_{c'\in C_i\setminus c} b_{c'\to v}^{(k-1)} \right\rfloor, \tag{5.5}$$

where the arbitrary scaling factor s has been added to the original algorithm to improve performance.

The hard decision bits of each variable node are computed by a majority vote of the sign bits of each memory $M_{v,c}$ and the initial channel data L_v:

$$h_v^{(k)} = \begin{cases} 0, \text{ if } \operatorname{sgn}_r\left(L_v\right) + \sum_{c\in C_i} \operatorname{sgn}_r\left(M_{v,c}^{(k)}\right) \geq 0 \\ \\ 1, \text{ if otherwise.} \end{cases} \tag{5.6}$$

The check and variable node operations are repeated iteratively until either a valid codeword is detected, or the iteration count k reaches a limit k_{max}.

5.3 Architectural Description

The check node function of DD-BMP is implemented in CMOS with a network of XOR gates, as shown in Figure 5–1. Figure 5–1a shows a direct implementation, with separate connections to and from each adjacent variable node. The check nodes

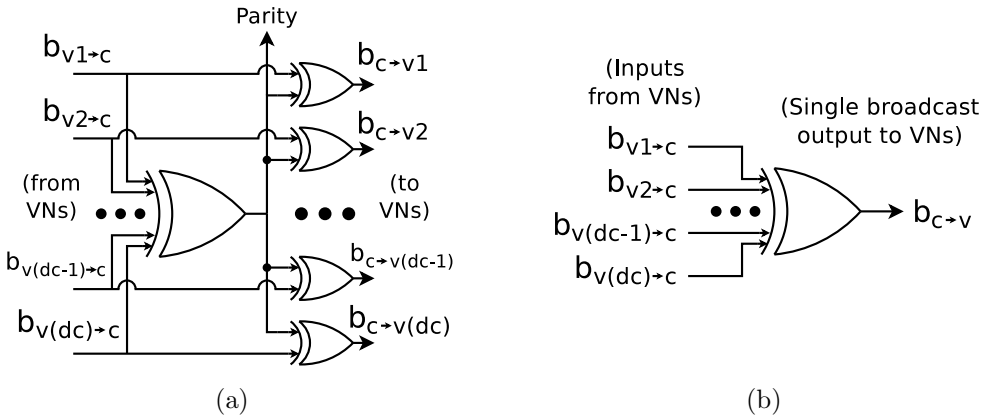


Figure 5–1: DD-BMP check node schematics for non-broadcast (a) and broadcast (b) versions.

can also be adapted to a "broadcasting" type as shown in Figure 5–1b. In this architecture, the 2-input XOR gates are relocated to the variable nodes, where their intrinsic contribution to the global check result is removed locally, and each CN has a single "broadcast" output. Because sign computation makes up the entirety of the CN operation in DD-BMP, this results in a decoder that is logically equivalent to the non-broadcasting version. However, the broadcasting version yields a simpler inter-node connection structure which tends to produce better results with automated place-and-route tools. In [56], broadcasting resulted in a 26% reduction in the average node-to-node wire length of a generic fully parallel decoder of a $(2048, 1723)$ Reed-Solomon (RS) based LDPC code [81].

The variable node schematic is shown in Figure 5–2. It consists of 2-input XOR gates at the inputs to implement broadcasting, an adder network to compute the sums of the input messages, and accumulators to perform the memory operations. The binary logic 0 and 1 respectively correspond to the $+1$ and -1 messages used

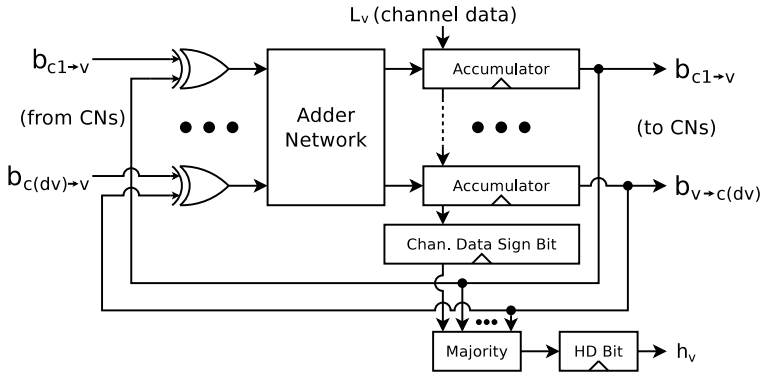


Figure 5–2: DD-BMP variable node schematic.

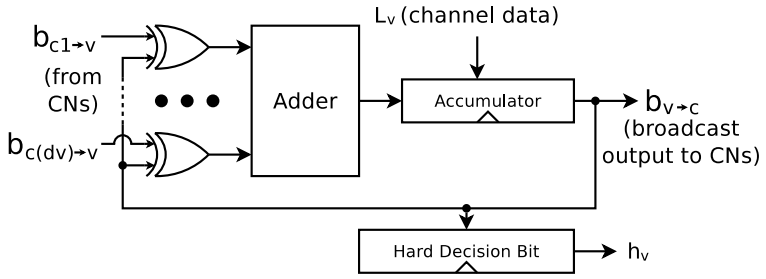


Figure 5–3: MDD-BMP variable node schematic.

in DD-BMP. The accumulators take the sums of the incoming CN-to-VN messages as input, and change their stored value by the corresponding amount. The stored values saturate at the maximum positive and negative numbers to prevent overflow.

The variable-to-check message outputs are taken from the sign bits of these accumulators. The initial values of the accumulators are set with soft decision data received from the channel. Finally, the hard decision bit is determined by a majority vote of all accumulator signs, plus the sign of the initial channel data.

Modified DD-BMP (MDD-BMP) is a variant of the original algorithm in which only a single memory is assigned to each variable node, replacing the unique variable-to-check messages with a single global message. Hence the memory update function of Equation 5.5 is replaced with:

$$M_v^{(k)} = M_v^{(k-1)} + \left\lfloor s \cdot \sum_{c' \in C_i} b_{c' \to v}^{(k-1)} \right\rfloor . \tag{5.7}$$

The variable node schematic for MDD-BMP is shown in Figure 5–3. This simplification results in a loss in BER performance, though it also greatly reduces the complexity of the variable node. We demonstrate that in the case of MDD-BMP, the performance degradation is small, while the complexity reduction is significant, so using MDD-BMP in VLSI implementations is well justified. Furthermore, it allows the broadcasting concept to be applied to the variable-to-check messages. In this work, we present results for decoders based on both DD-BMP and MDD-BMP.

At the system level, these architectures posses several merits that are useful in energy-efficient designs. The variable and check nodes are both simple, which translates to lower area and power consumption in VLSI chips. Furthermore, binary message exchange and broadcasting reduces wiring overhead. Since several successive messages opposing the value stored in a VN must be received before the sign of the VN changes, switching activity is also low [24]. Furthermore, DD-BMP is amenable to efficient highly parallel architectures. Unlike some other fully node-parallel decoder architectures such as bit-serial min-sum [58] and our previous pulse width min-sum decoder from Chapter 4, iterations in DD-BMP complete in a single clock cycle. As a result, DD-BMP tends to converge to a valid codeword rapidly,

giving it a very high average throughput. Voltage and frequency scaling (VFS) can be applied, trading off throughput for reduced dynamic energy consumption.

5.4 Design Results

We designed DD-BMP and MDD-BMP decoders for a $(273, 191)$ FG-LDPC code, as well as an MDD-BMP decoders for $(1023, 781)$ and $(4095, 3367)$ codes using TSMC 65 nm 7LM single-V_t CMOS technology. Due to high memory requirements, designs using the larger codes with DD-BMP turned out to be impractical. All decoders are fully parallel, and use an early termination scheme based on parity check satisfaction. In addition, none of the decoders employ pipelining, and therefore iterations complete in a single clock cycle with no additional latency.

All results presented herein are based on post-layout simulations of ASIC standard cell designs, and include the clock tree, with back-annotated delays and wiring parasitics. Each decoder was produced using the same design flow: we used Cadence RTL Compiler Ultra for logic synthesis, Cadence EDI System for place and route, Synopsys VCS for post-layout simulation, and Synopsys Primetime for power estimation. All presented results use "typical" operating conditions. For nominal supply voltage, these are TT process, $V_{DD} = 1.0$ V, and $T = 25°$C. For reduced supply voltage results, these are TT process, $V_{DD} = 0.8$ V, and $T = 25°$C. Results for both voltages are obtained using design kit libraries.

Figure 5–4 shows the BER performance of our designs, along with baselines of min-sum (MSA) and offset min-sum (OMS). All use 6 quantization bits and 31 maximum iterations. With these parameters and codes, OMS delivers performance within 0.1 dB of floating point SPA (FP-SPA). These measurements all use random

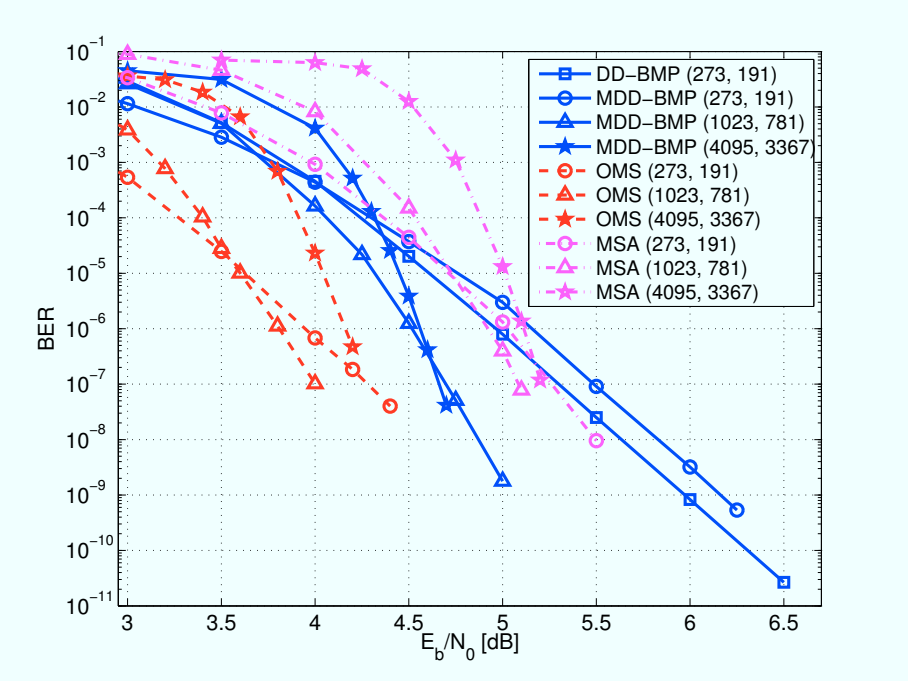


Figure 5–4: BER performance of the codes in this work decoded with DD-BMP, MDD-BMP, offset MSA (OMS), and MSA. All simulations use 6 quantization bits and 31 maximum iterations.

codewords, with binary phase-shift keying (BPSK) modulation, and noise added using an emulated additive white Gaussian noise (AWGN) channel. These results were obtained with software simulations, and in the case of the $(273, 191)$ code, with a prototype implemented on a Xilinx Virtex-5 LX330 FPGA.

The number of quantization bits q and scaling factor s are selected to give the best trade-off between performance and area. For the $(273, 191)$ decoders, $q = 6$ and $s = 1$ are used, while for the $(1023, 781)$ and $(4095, 3367)$ decoders, $q = 6$ and $s = 0.5$ are used. As shown, decoding of the $(273, 191)$ code is slightly worse than MSA with both DD-BMP and MDD-BMP. The longer codes exhibit performance

between OMS and MSA when decoded with MDD-BMP, with the $(1023, 781)$ code performing 0.7 dB worse than OMS and 0.4 dB better than MSA at a BER of 10^{-6}. The $(4095, 3367)$ code delivers the best relative performance, being about 0.4 dB worse than OMS and 0.5 dB better than MSA. Standard MSA performs quite poorly with these codes, because numerical saturation occurs in the high-degree VNs. Thus, DD-BMP and MDD-BMP can perform better, despite having less inter-node information exchange. In addition, the updating rules of DD-BMP and MDD-BMP are based on relaxed successive substitution, which underestimates the reliability of check node outputs. This has been shown to improve the error correction performance of iterative decoders in general [98]. MSA, by contrast, overestimates check-to-variable messages compared to SPA. These problems can be solved with OMS, although it is very costly to implement, due to the high degree check nodes in these codes. These results demonstrate that DD-BMP and MDD-BMP achieve good error correction performance that is suitable for many applications, despite having far lower complexity than MSA. Notably, a highly parallel decoder based on MSA or its variants would be too complex for a practical VLSI implementation of the longer FG-LDPC codes.

Simulation results for throughput, average iteration count, power consumption, and energy efficiency are shown in Figures 5–5, 5–6, 5–7, and 5–8 respectively. The data sets for both nominal supply voltage ($V_{DD} = 1.0$ V) and reduced supply voltage ($V_{DD} = 0.8$ V) are obtained by post-layout simulations using standard cell libraries for each of the respective supply voltages. In these simulations the decoders are

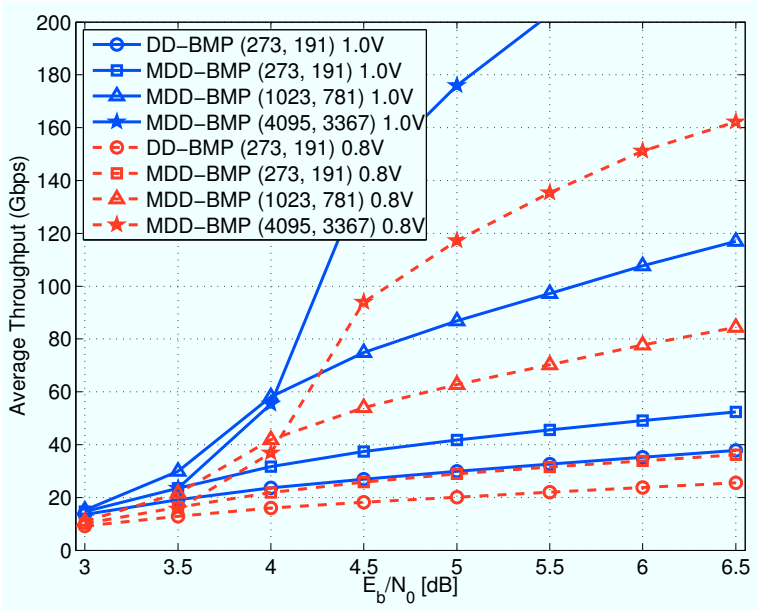


Figure 5–5: Average throughput for the designed decoders.

operated continuously. Once the decoder signals completion, either through convergence detection or by reaching the iteration limit, another codeword is immediately loaded and decoding begins again. The values for energy efficiency were calculated by dividing power by throughput.

The full post-layout ASIC design results are summarized in Table 5–1, alongside results for comparable previously published decoders. For the shorter codes, DD-BMP and MDD-BMP have highly area-efficient implementations, despite their high node degrees and edge counts, demonstrating that FG-LDPC codes can be efficiently implemented in VLSI. In the case of the $(4095, 3367)$ code, the huge number of edges (262,080) and high node degrees lead to severe routing congestion, which results in much lower utilization, clock frequency, and energy efficiency. Despite this, our design using the $(4095, 3367)$ code is still practicable. Furthermore, its decoding

Table 5–1: Post-Layout Design Results and Comparisons With Other Works

	This work								[64]		[60]		[96]
Decoding algorithm	DD-BMP		MDD-BMP		MDD-BMP		MDD-BMP		Split-row MS		Offset MS		Hybrid SBF
LDPC code	(273, 191)		(273, 191)		(1023, 781)		(4095, 3367)		(2048, 1723)		(2048, 1723)		(1057, 813)
Node degrees (d_v, d_c)	(17, 17)		(17, 17)		(32, 32)		(64, 64)		(6, 32)		(6, 32)		(33, 33)
Edge count	4641		4641		32736		262080		12288		12288		34881
Technology	65 nm		65 nm		65 nm		65 nm		65 nm		65 nm		180 nm
Quantization bits	6		6		6		6		5		4		4
Area	1.44 mm^2		0.276 mm^2		1.38 mm^2		15.37 mm^2		4.84 mm^2		5.35 mm^2		7.4 mm^2
Utilization	89%		90%		93%		43%		97%		84.5%		50%
Decoding iterations	31		31		31		31		11		8 + 6 post proc.		40
Supply voltage (V)	1.0	0.8	1.0	0.8	1.0	0.8	1.0	0.8	1.3	0.7	1.2	0.7	1.8
Clock frequency (MHz)	400	270	550	380	360	260	180	120	195	35	700	100	345
Min. throughput (Gbps)	3.41	2.30	4.69	3.24	11.5	8.31	23.0	15.3	36.3	6.52	15.8	2.26	0.25
Av. throughput (Gbps)*	26.9	18.2	37.4	25.8	74.8	54.0	140.9	93.9	92.8	16.6	47.7	6.67	1.05
Av. power (mW)*	894	375.2	183	79.3	989	441	5354	2210	1359	62	2800	144	1450
Av. energy (pJ/bit)*	33.3	20.6	4.88	3.07	13.2	8.16	37.9	23.5	14.6	3.7	58.7	21.5	1381
Scaled energy (pJ/bit)†	33.3		4.88		13.2		37.9		8.64		40.8		153.9
Scaled tpt. (Gbps/mm^2)‡	18.7		135.5		54.2		9.2		19.2		8.92		1.1

* Averages are measured at $E_b/N_0 = 4.5$ dB in this work, at $E_b/N_0 = 4.55$ dB in [64], at $E_b/N_0 = 5.5$ dB in [60], and at BER $= 10^{-5}$ in [96].

† Energy scaled to 1.0 V and 65 nm process.

‡ Average throughput per unit area scaled to 65 nm at constant frequency.

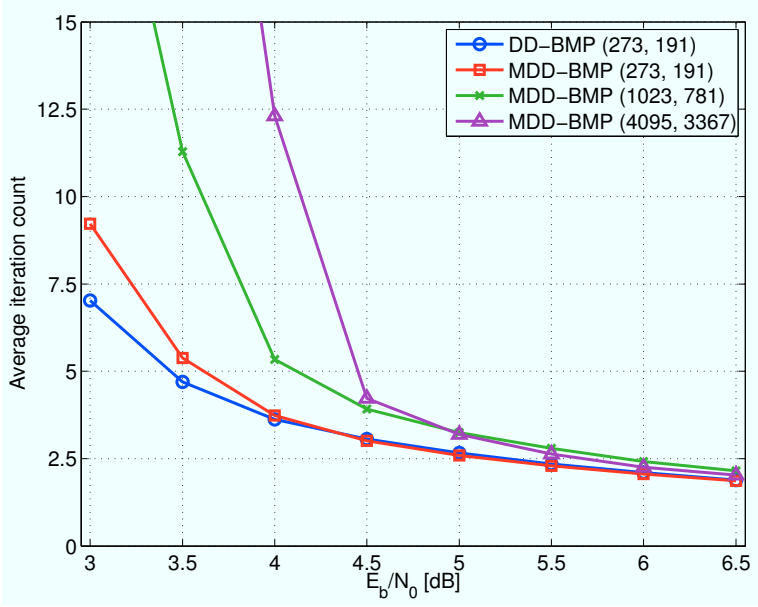


Figure 5–6: Average iteration counts for the designed decoders.

throughput exceeds 200 Gbps for values of E_b/N_0 above 5.5 dB. It also has the largest BER performance gain over MSA, and the smallest gap to OMS performance.

Due to differing codes and other parameters, it is difficult to make fair comparisons between this work and other energy-efficient designs. However, scaled throughput per unit area and energy per decoded bit are considered to be the most meaningful efficiency metrics for comparing decoders under similar conditions [79].

Two different designs for an energy-efficient min-sum decoder are presented in [64] and [60], the former using a split-row approach and the latter using a grouped-parallel approach. Compared to [60], MDD-BMP with the $(1023, 781)$ code achieves 6.1 times greater throughput per unit area and approximately 3.1 times the energy efficiency after scaling to a 1.0 V supply voltage. Compared to the split-row decoder in [64], this MDD-BMP decoder achieves 2.8 times greater throughput at nominal

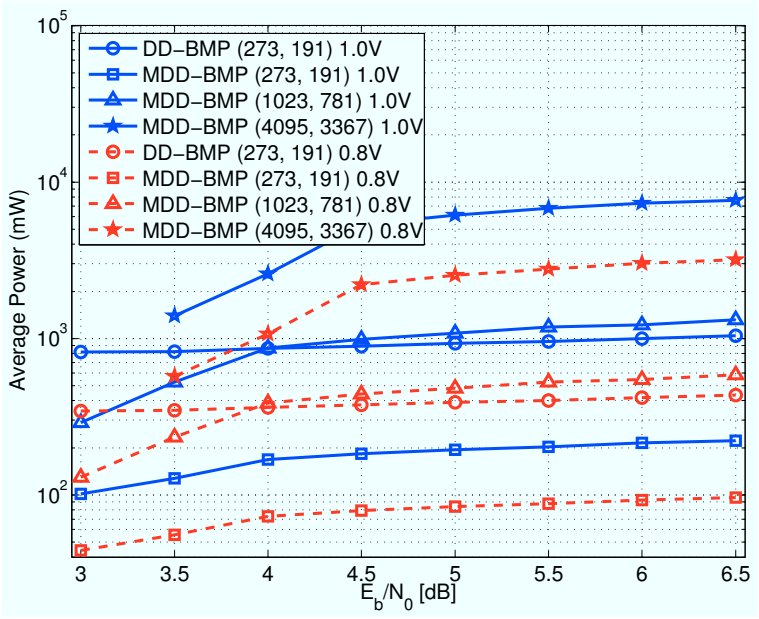


Figure 5–7: Average power for the designed decoders.

supply voltage, but has 35% higher energy consumption per decoded bit after supply voltage scaling to 1.0 V. However, since [64] uses a 1.3 V supply, scaling would considerably reduce its throughput, and MDD-BMP has 10.6% better energy efficiency at the nominal voltages.

In addition, both [64] and [60] use a $(2048, 1723)$ LDPC code that is known to have an early error floor - the offset MSA of [60] exhibits an error floor at BER $= 10^{-10}$ that is overcome with post processing. Split-row [64], which uses normalized MSA with check node heuristics, only provides results down to BER $= 10^{-7}$ and does not investigate the presence of an error floor. FG-LDPC codes, on the other hand, are well known for the absence of early error floors. This is confirmed by the BER measurements in Figure 5–4, which show that DD-BMP and MDD-BMP do not exhibit any error floor above BER $= 10^{-11}$, even with the short $(273, 191)$ code.

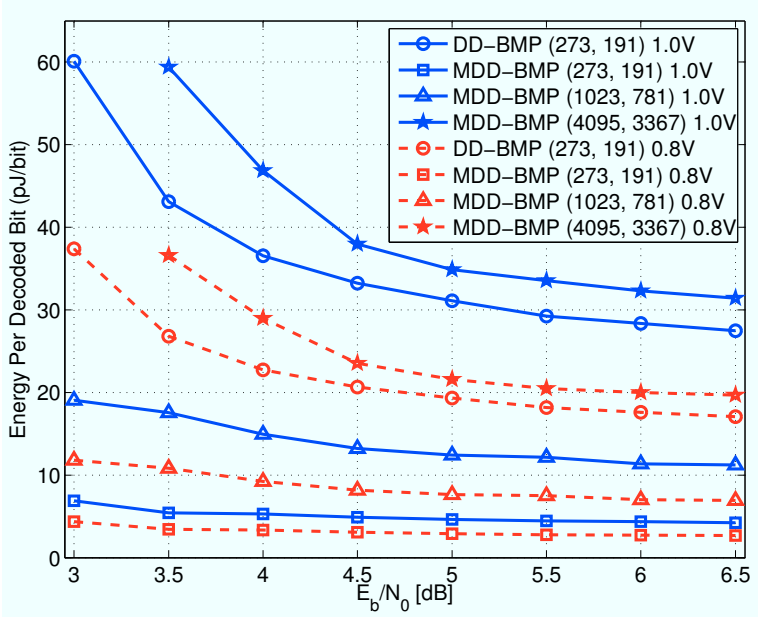


Figure 5–8: Average energy per decoded bit for the designed decoders.

The hybrid soft bit-flipping (SBF) decoder in [96] uses a $(1057, 813)$ FG-LDPC code very similar to the $(1023, 781)$ code used in this work, and achieves performance approximately 0.2 dB better than MDD-BMP. However, even with a partially parallel architecture, this algorithm is more computationally intensive and highly routing-limited, and throughput is much lower than fully parallel designs. DD-BMP and MDD-BMP also achieve better energy efficiency after scaling for process technology and supply voltage.

5.5 Use With Other LDPC Codes

Although we have shown that differential binary (DB) algorithms are an effective, energy-efficient means of decoding FG-LDPC codes, it is not always practical or possible to use an FG code. In particular, communications standards mandate the use of standard codes, and there are currently no standards that use FG codes.

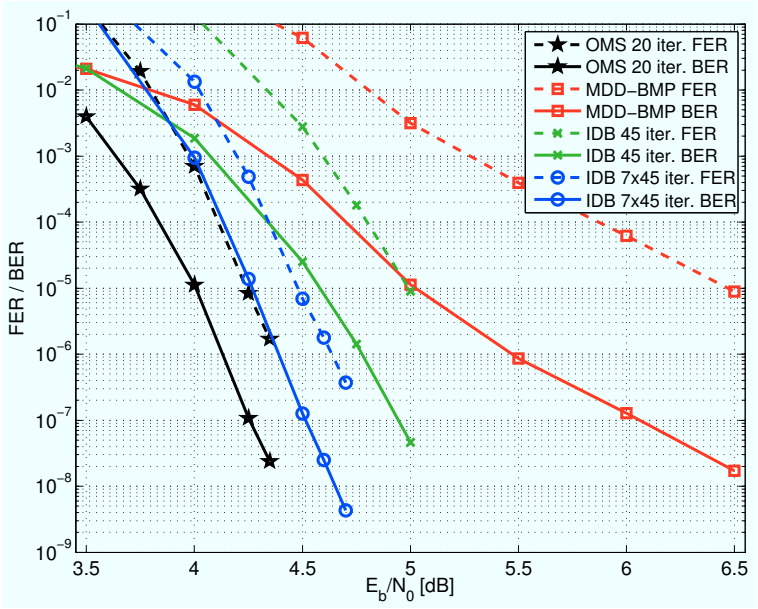


Figure 5–9: BER/FER performance of IDB with the $(2048, 1723)$ RS-LDPC code, compared to the original MDD-BMP algorithm with 100 iterations, and 4-bit OMS with 20 iterations.

Simulation results in [24] show that DD-BMP can effectively decode randomly generated LDPC codes, as long as the VN degree is high enough, but suffers from poor error correction performance otherwise. We have also verified that MDD-BMP has poor decoding performance for a $(2048, 1723)$ RS-based LDPC code with (d_v, d_c) of $(6, 32)$ [81]. This code is used in the IEEE 802.3an standard for 10 Gbps Ethernet (10GBASE-T) [13], and is also a common benchmark code for highly parallel LDPC decoder implementations.

Figure 5–9 plots the BER/FER performance of this code decoded using MDD-BMP with $q = 6$, $s = 0.5$ and $k_{max} = 100$, and offset min-sum (OMS) with $q = 4$ and $k_{max} = 20$. MDD-BMP performs considerably worse than OMS with this code,

and additionally exhibits a high "error floor" - a region in which the slope of the curve falls significantly above a certain signal-to-noise ratio.

The topic of error floors and their causes in LDPC codes has been the subject of several previous works. It is well known that a major cause of error floor behaviour is the presence of inherent structural flaws in the code's graph. Due to these flaws, certain error patterns occurring in small sub-graphs are maintained or strengthened under iterative message passing. Consequently, the decoding algorithm is "trapped" in this error pattern, and fails as it is unable to correct it.

In [99], this phenomenon was observed in a $(2640, 1320)$ Margulis LDPC code. The cause was identified as "near-codewords" on account of the stability of these structures under iterative decoding, despite them not being valid codewords. The term *trapping sets* was introduced in [97] to describe in general the patterns which cause these failures in LDPC codes over an AWGN channel. An (a, b) trapping set is defined as a subgraph consisting of a erroneous VNs (the "weight" of the set), with an extrinsic message degree (EMD) of b (i.e., the number of unsatisfied CNs incident to the erroneous VNs).

The related concept of *stopping sets*, which determine error floor performance of LDPC codes over the binary erasure channel, was studied in [100]. Error floor behaviour has also proven to be a major issue in the $(2048, 1723)$ RS-LDPC code, as the 10 Gbps Ethernet specification mandates very low BER performance. The dominant trapping sets of this code, termed *absorbing sets*, are defined as a special subclass of trapping set which is guaranteed to be stable under Gallager bit-flipping

decoding [101]. Additional analysis of the formation and dynamics of absorbing sets is performed in [102].

There have also been a number of implementation-oriented methods proposed for lowering the error floors of LDPC codes by overcoming or avoiding trapping sets. A prominent example is [60], which presents a hardware implementation of an OMS decoder for the $(2048, 1723)$ RS-LDPC code employing a post-processing decoding stage. This technique has proven highly effective at overcoming the dominant $(8, 8)$ absorbing set of this code, and thereby lowering the error floor. Another hardware implementation in [103] proposes an iterative decoding algorithm with *backtracking*, which attempts to collapse trapping sets by identifying participating bits and flipping them. The stochastic decoders in [71] and [72] can employ *redecoding*, which restarts decoding with a different random number generator seed - since these algorithms are probabilistic, decoding may thus take a different trajectory and avoid entering a trapping set that caused a previous attempt to fail. Similarly, in *dithered belief propagation*, random processes are used in attempts to avoid or break out of trapping sets [104].

Likewise, early error floors and poor error correction performance are issues that must be solved for DB algorithms to be of practical use with general (non-FG) LDPC codes. In this work, we will focus on the $(2048, 1723)$ RS-LDPC code, due to its importance to the IEEE 802.3an standard, the large body of prior work investigating its trapping sets, and its high popularity as an implementation target for highly-parallel decoder architectures.

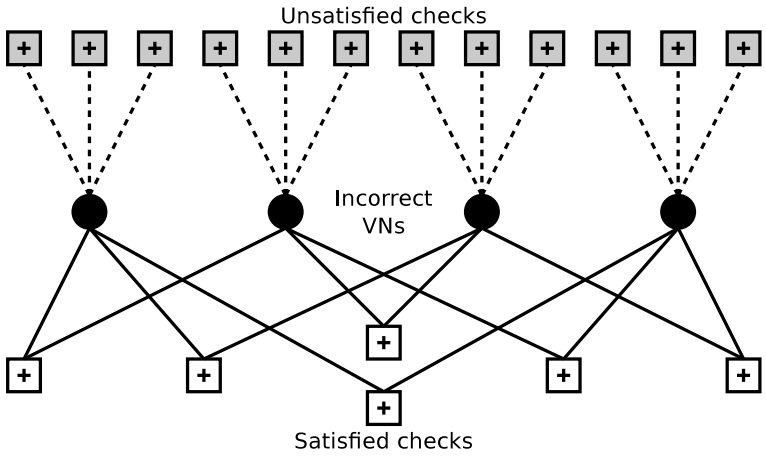


Figure 5–10: The dominant $(4, 12)$ trapping set of the $(2048, 1723)$ RS-LDPC code under MDD-BMP decoding.

5.6 Trapping Sets

A detailed examination of the failure cases of MDD-BMP with the $(2048, 1723)$ code revealed that decoding failures in the error floor region occurred due to trapping sets with very low weights and high EMDs. The dominant set among these is a $(4, 12)$ trapping set, the subgraph of which is shown in Figure 5–10. It is easy to see how this configuration fails under MDD-BMP decoding: each of the erroneous VNs receives 3 messages opposing the incorrect bit, and 3 reinforcing it. These messages sum to 0, and so the state of the VN does not change. The decoder is therefore deadlocked and unable to correct these errors.

This set can, in fact, be considered an absorbing set in the context of MDD-BMP, because it is stable under bit-flipping operations. This arises because MDD-BMP employs VN message broadcasting (i.e., each VN computes a single global message, which is sent to all neighbouring check nodes). We therefore define a *weak absorbing set* as a set of VNs, such that at least $d_v/2$ of each VN's neighbouring CNs are

connected to the set an even number of times. This differs from the absorbing set definition given in [101] and [102], in which strictly greater than $d_v/2$ of each VN's neighbours must have an even number of connections to the set. We henceforth refer to this as a *strong* absorbing set to distinguish it from a weak one. The $(2048, 1723)$ RS-LDPC code contains a great number of weak absorbing sets. Many of these, such as the $(4, 12)$ set detailed above, are very small in size and have very large multiplicities, and thus result in the severe error floor seen in Figure 5–9 [97].

Furthermore, the use of binary messages exacerbates the vulnerability of MDD-BMP to trapping sets, as a strong extrinsic message cannot overcome a group of weak intrinsic messages. One consequence of this is that once a (weak) absorbing set is entered, it cannot be escaped, regardless of the LLR values in the participating VNs.

5.7 An Improved Differential Binary Algorithm

While a variety of algorithmic methods have been proposed for lessening the impact of trapping sets, none are particularly applicable to MDD-BMP, or to the weak absorbing sets that dominate its error floor with the 10 Gbps Ethernet LDPC code.

For instance, in the post-processing decoding phase of [60], the magnitudes of messages from unsatisfied check nodes are scaled up, while messages from satisfied check nodes are scaled down, but this technique is ineffective with binary messages. The backtracking algorithm of [103] attempts to escape trapping sets by flipping VNs connected to unsatisfied CNs, but this is only effective for trapping sets with few candidate VNs. For the case of our $(4, 12)$ weak absorbing set, there are only 4

erroneous VNs out of 384 candidates, so a large number of trials would be needed to be effective. The random sign flip (RSF) technique of [104] faces a similar issue. Other techniques, such as bi-mode decoding [105], significantly increase the complexity of VLSI implementations.

We propose instead two low-complexity modifications to the MDD-BMP algorithm, called *degeneration* and *relaunching*. We refer to MDD-BMP with either or both of these modifications as IDB, for "improved differential binary" decoding algorithm.

The degeneration technique modifies the VN operation such that an offset d is subtracted from the differential value at each iteration. The MDD-BMP variable node update function of (5.7) is thus modified to:

$$M_v^{(k)} = M_v^{(k-1)} + \left\lfloor s \cdot \sum_{c' \in C_i} b_{c' \to v}^{(k-1)} \right\rfloor - d \cdot \mathrm{sgn}_r \left(M_v^{(k-1)} \right). \tag{5.8}$$

This will cause the stored value M_v to "degenerate" towards zero over time, unless the message sum has a magnitude equal to or greater than d. Also note that if $M_v^{(k-1)}$ and the message sum are sufficiently small, M_v will oscillate between positive and negative values on successive iterations. The primary purpose of this is to provide a mechanism to break up weak absorbing sets. The VNs in such a set with $d_v/2$ connections to falsely satisfied CNs will have an incoming message sum of zero, and will therefore eventually change sign and possibly cause the set to collapse. Returning to our $(4, 12)$ weak absorbing set example in Figure 5–10, it can

be readily seen that flipping the sign of one or more VNs will quickly correct the remaining VNs.

However, while the degeneration technique is effective at overcoming the weak absorbing sets that beleaguer MDD-BMP, it is ineffective at correcting strong absorbing sets. This is because flipping a VN in a strong absorbing set would require a large value of d, which would then cause correct bits outside the set to be erroneously flipped and propagate errors through the entire graph. In fact, strong absorbing sets, including the infamous $(8, 8)$ set that has been the subject of much scrutiny in prior works, make up the dominant failure modes of IDB with degeneration. We also observe that these sets are encountered with much greater frequency than in OMS. As noted above, the use of binary messages contributes to this, as an absorbing set cannot be disrupted by a small number of strong extrinsic messages.

In order to further improve the performance of IDB, we propose another algorithmic modification called "relaunching". Prior works have noted that since the decoding outcome is dependent on the initial state of the decoder, an unsuccessful decoding attempt can possibly be corrected by applying small perturbations to the initial state and retrying. For example, the concept of "redecoding" for stochastic LDPC decoders was introduced in [107], and is also used in [71]. As stochastic decoding is a probabilistic process, redecoding re-attempts decoding with the same initial channel LLR values, but a different random number generator (RNG) seed. Similarly, the random initial state (RIS) technique proposed in [104] re-attempts decoding with random changes applied to the initial channel data.

With the relaunching technique, we divide the decoding process into multiple phases denoted by an index p. At the start of each phase, the variable nodes are reset to the initial channel LLR values with an offset applied as follows:

$$M_v^{(0,p)} = \text{sgn}_r\left(L_v\right) \cdot \max\left(\frac{1 - \text{sgn}_r\left(L_v\right)}{2}, |L_v| - F\left(p, v\right)\right), \tag{5.9}$$

where $F\left(p, v\right)$ is a non-negative function of the decoding phase p and VN index v. Decoding then proceeds normally, until a valid codeword is detected or k_{max}^p iterations have passed. If the decoder did not converge to a valid codeword, p is incremented and the next phase of decoding begins, or if p is the final phase, failure is declared and decoding stops.

The relaunching technique is similar to RIS in that it attempts to circumvent decoding failures by re-attempting decoding with different initial channel values. However, in relaunching, the changes applied to the initial channel values are deterministic, their magnitudes can never be increased or their signs flipped as a result of these changes, and different phases p may use different values for the parameters s and d in the VN update function.

5.7.1 Parameter Selection

The IDB algorithm has a large design space with several parameters: s, d, number of phases p, k_{max}^p, and even the relaunch function $F\left(p, v\right)$. This section describes the trade-offs associated with each parameter, and presents a balanced decoding schedule that is used in the subsequent IDB decoder design.

The parameters s and d control the VN update function. As mentioned earlier, setting $s < 1$ improves error correction performance through successive relaxation [98]. However, lower values of s also increase the average number of iterations required for convergence, reducing throughput. Very low values of s should also be avoided, since they may cause the sum of incoming messages to be incorrectly truncated to zero, or unable to overcome the degeneration factor d. In terms of hardware impact, $s = 2^x$ (where x is an integer) can be implemented with no additional hardware, which makes $s = 1$ and $s = 0.5$ attractive options. Other values require one or more additional adders in the VN, which increase the amount of logic, critical path, and energy consumption of the decoder. While $s = 0.75$ is a reasonable value in terms of algorithmic performance, the additional circuit and implementation costs are much higher than the benefits of this value. Thus, we consider only $s = 1$ and $s = 0.5$.

For d, the value should be as small as possible. As mentioned before, a large value of d can cause correct bits or majority inputs to be overruled. Thus, $d = 1$ is optimal, as representing any smaller number would require additional hardware.

The relaunching function $F(p, v)$ would ideally be random, as in the RIS technique upon which it is based [104]. However, large amounts of random data necessitating a large number of RNGs would be required for this, which conflicts with the design goals of energy efficiency and low area. Instead, we chose a modulo function, using only adders and comparator. This design was found to have very little impact on error correction performance compared to random data, while also being much simpler.

Table 5–2: Decoding Schedule For IDB With Relaunching

Phase (p)	$F(p, v)$	k^p_{max}	s	d
0	1	45	1	1
1	1	45	0.5	1
2 - 6	$[(p + v - 1) \bmod 5 + 1]$	45	0.5	1

Finally, the number of phases p and iterations per phase k^p_{max} is primarily dependent on the desired level of error correction performance. In this case, we would like to achieve BER/FER performance comparable to or better than the split-row min-sum [64], stochastic [71], and RHS algorithms [72]. A lower priority in this design is maximum latency, or worst-case performance, which is determined by the maximum number of iterations.

The IDB decoder designed in this work uses the parameters and decoding schedule shown in Table 5–2. The initial phase applies a uniform offset of 1 to the initial LLR data. The relaxation factor s of 1 reduces error correction performance for this phase, but also reduces the average number of iterations, thus giving higher average throughput. The next phase uses the same uniform offsets with $s = 0.5$. In successive phases, offset values from 1 to 5 are assigned sequentially to each L_v. All phases have a degeneration factor of 1, and 45 maximum iterations, giving the overall decoding process 315 maximum iterations.

The error correction performance of IDB is shown in Figure 5–9, along with OMS ($q = 4$, $k_{max} = 20$) and MDD-BMP ($q = 6$, $k_{max} = 100$) as baselines. The parameters used for IDB with only the degeneration technique are $q = 6$, $s = 0.5$, $k_{max} = 45$, and $d = 1$, while IDB with both the relaunching and degeneration techniques uses $q = 6$ and the schedule from Table 5–2 (labelled in the legend as 7x45 iter.) As these

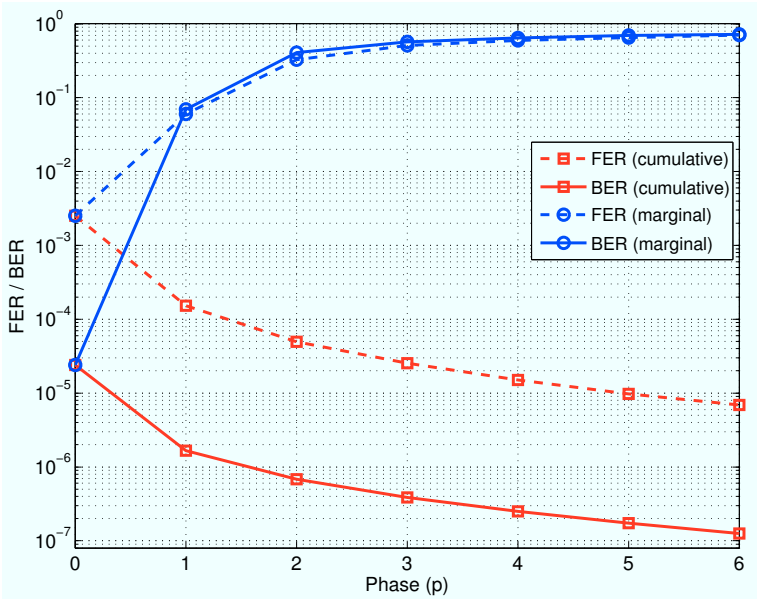


Figure 5–11: Cumulative and marginal FER/BER performance of IDB with relaunching, at $E_b/N_0 = 4.5$dB, using the schedule shown in Table 5–2.

results show, degeneration is hugely successful in improving the error correction performance over MDD-BMP. The error floor is reduced by at least 2 orders of magnitude, and BER performance within 0.7 dB of OMS is achieved in the waterfall region. The improvement from relaunching is less dramatic but still significant, about 0.4 dB around BER $= 10^{-7}$. Notably, this gives IDB error correction performance comparable to split-row min-sum [64], stochastic [71], and relaxed half-stochastic algorithms [72], but with much lower computational complexity.

The cumulative and marginal error correction performance for each relaunching phase at $E_b/N_0 = 4.5$dB is plotted in Figure 5–11. The marginal error rate of a phase p is defined as the proportion of frames (or bits) that are not successfully decoded during phase p, out of all frames (or bits) in which phase p is reached.

These results show the effectiveness of relaunching at correcting frames that could not be decoded successfully with a single phase of IDB. In addition, despite the high maximum number of iterations for the overall decoding process (315), we observe that the vast majority of frames decode successfully in the first phases, meaning that the average number of iterations is much lower (see Figure 5–16). We also observe steeply diminishing returns in the marginal performance of each phase, from which we can conclude increasing the number of phases will not significantly improve error correction performance. It is also notable that the marginal BER is higher than the marginal FER for each relaunch, meaning that relaunching improves the FER more than it improves the BER, and thus also increases the average number of bit errors per frame error. This shows that relaunching corrects a greater proportion of frame errors with few bit errors, such as the $(8, 8)$ and other small absorbing sets.

A histogram showing the distribution of the number of iterations required for convergence to a valid codeword $E_b/N_0 = 4.5$dB is shown in Figure 5–12. A total of 10^8 randomly generated frames are plotted. This histogram also shows the mean and points of interest for the cumulative distribution function D. This again shows that the convergence performance of IDB is very fast on average, but also has a very long-tailed distribution. The overall mean is 6.68 iterations, and 99% of frames converge in 17 or fewer iterations, but a non-negligible number of frames also converge after 200 or more iterations. This histogram also justifies the choice of $k^p_{max} = 45$ for each schedule entry - after 45 iterations, decoding is more likely to be successful after a relaunch, which can be seen in the peaks of the distributions for each individual phase rising above the rightmost edge of the previous phase.

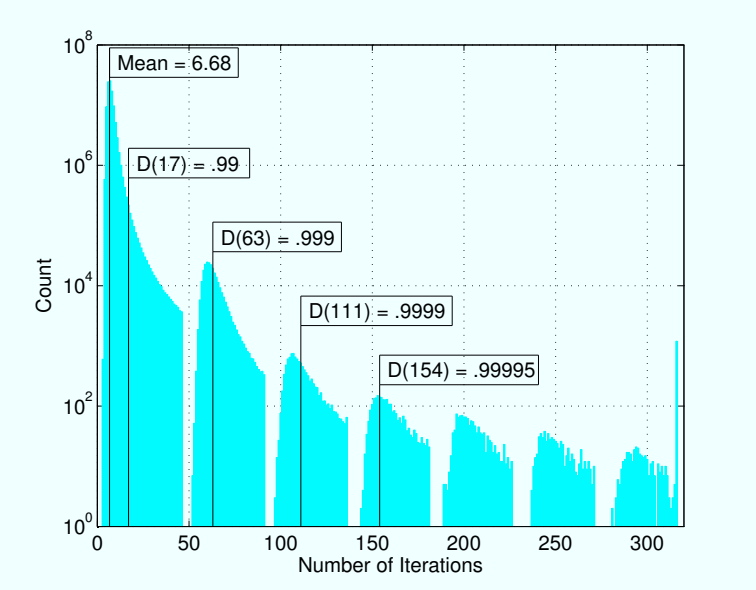


Figure 5–12: Histogram showing the distribution of decoding iterations required for convergence for IDB, using the schedule shown in Table 5–2. A total of 10^8 frames at $E_b/N_0 = 4.5$dB were sampled.

Figure 5–13 shows the error correction performance of IDB decoding with other LDPC codes with other construction methods: $(648, 324)$ and $(648, 540)$ irregular quasi-cyclic (QC)-LDPC codes from the Wi-Fi (IEEE 802.11n) specification [10], and a $(660, 484)$ regular progressive edge growth (PEG) based code, with $(d_v, d_c) = (4, 15)$. As expected, MDD-BMP and IDB perform quite poorly with the Wi-Fi codes, due to the presence of large numbers of degree-2 VNs in these codes. IDB fares slightly better with the $(648, 540)$ code, as it has fewer of these low degree nodes, but is still 0.6 - 1.0 dB worse than OMS. The performance of IDB with the $(660, 484)$ PEG-based code is considerably better, greatly improving over MDD-BMP and approaching within 0.4 dB of OMS at its closest point. However, IDB also exhibits a severe error floor with this code. These results again show that for best

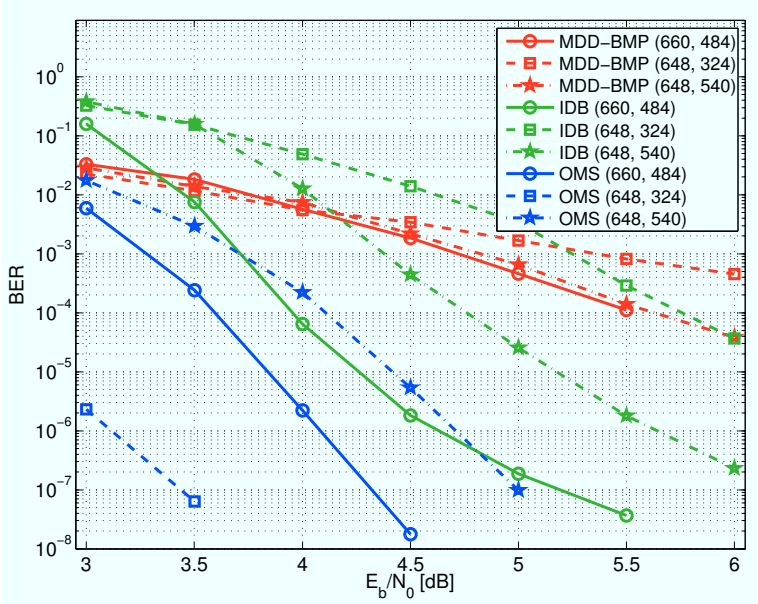


Figure 5–13: BER performance of IDB decoding with other LDPC codes, along with MDD-BMP and OMS for comparison. All decoders use 6 bits of quantization. IDB uses the decoding schedule from Table 5–2, OMS and MDD-BMP use 20 and 45 maximum iterations respectively.

results, DB decoding algorithms should be used in conjunction with codes that lack low degree VNs.

5.8 Design Results For IDB

As with DD-BMP and MDD-BMP, we have created an ASIC standard cell design of an IDB decoder, and used post-layout simulations to estimate the throughput, power consumption, and energy efficiency. This design uses the $(2048, 1723)$ RS-LDPC code used in the 10 Gbps Ethernet specification, with 6 bits of quantization, and employs degeneration and relaunching according to the decoding schedule shown in Table 5–2. This design employs the same process technology (TSMC 65 nm 7LM single-V_t CMOS), software tools, design flow, and operating conditions as

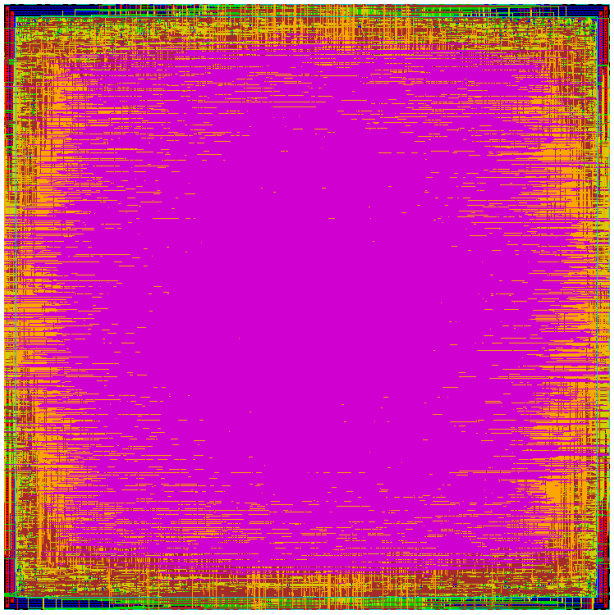


Figure 5–14: Layout view of the IDB decoder design.

specified in Section 5.4, and again post-layout simulations are used to obtain all circuit performance metrics. Figure 5–14 shows a screenshot of the layout view of the IDB decoder within Cadence EDI.

A top level block diagram of the designed IDB decoder is shown in Figure 5–15. Architecturally, the IDB decoder is highly similar to MDD-BMP, the algorithm from which it is based. As before, this decoder is not pipelined, and each iteration completes in a single clock cycle. The degeneration function is implemented internally by the accumulators. Relaunching requires a channel data buffer to retain the original values of L_v throughout the decoding process, which is labelled "L_v buffer" in the figure. The values for $M_v^{(0,p)}$ - that is, $L_v - F(p, v)$ - are computed in parallel with arithmetic modules between the L_v buffer and the accumulators. Thus each relaunch takes a single clock cycle and can be performed concurrently with the final iteration

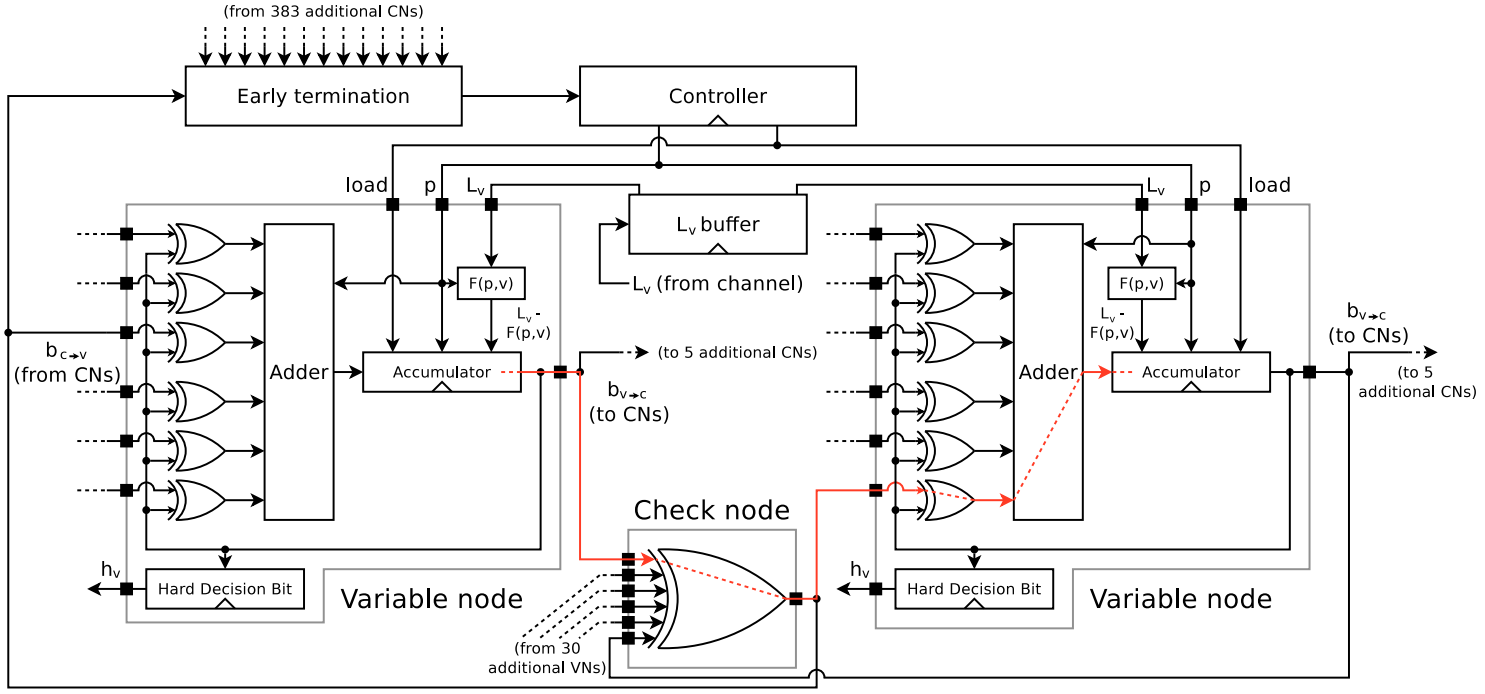


Figure 5–15: Top level block diagram of the IDB decoder, showing a check node, 2 neighbouring variable nodes, and other top-level components. The critical path is traced in red.

Table 5–3: IDB Decoder Critical Path (Typical Operating Conditions)

Component	Delay (ps)
Accumulator output (start point)	—
VN-CN buffers & interconnect	299
Check node	368
CN-VN buffers & interconnect	353
Input XOR	236
Adder	260
Accumulator (end point)	407
Total	1923

of the previous phase. The *load* control signal causes the accumulators in each VN to load $L_v - F(p, v)$ at the start of each decoding phase. The decoding phase p is generated by the controller state machine and distributed to all VNs, where it is used to determine $F(p, v)$ and s (v is constant for each given VN). As before, early termination is based on parity checks generated at the CNs. If all parity checks are satisfied, decoding stops and the next frame is loaded into the L_v buffer. Otherwise, decoding continues according to the schedule in Table 5–2.

The critical path of the IDB decoder is traced in red in Figure 5–15. It begins at the accumulator output of one VN, passes through a CN and on to a neighbouring VN, ending at the VN's accumulator registers. Table 5–3 details the critical path and the delay accrued through each segment. Despite the lack of pipelining, IDB achieves a much faster critical path than min-sum based decoders due to simpler node logic. In particular, the IDB check node is simply a d_c-input XOR gate, and does not need to find the smallest magnitude among the input messages. The VN adder tree is simpler, as it needs only to add d_v inputs of 1 bit each. In addition,

the smaller silicon area results in lower buffering and interconnect delay for the long inter-node connections along the critical path.

The post-layout results of the IDB decoder are summarized in Table 5–4, along with other state-of-the-art decoders using the $(2048, 1723)$ RS-LDPC code as comparison. IDB presents very large improvements in area, throughput, and energy efficiency. In terms of silicon area, IDB is 48% smaller than the next smallest decoder design, which is the partially-parallel layered OMS decoder of [90]. It is 57% smaller than MTFM-based stochastic [71], which is the next smallest fully-parallel architecture.

At $E_b/N_0 = 5.5$ dB, the average throughput of IDB is 171.8 Gbps, which is 7% greater than the throughput reported for relaxed half-stochastic (RHS) [72], and approximately 3.5 times greater than [60]. The average throughput of the IDB decoder at $E_b/N_0 = 4.55$ dB is 126.3 Gbps, which is an improvement of 36% over split-row min-sum at the same signal-to-noise ratio [64].

Despite the much lower computational complexity of the IDB algorithm, IDB achieves error correction performance comparable to min-sum and stochastic-based decoders. At a BER of 10^{-7}, correction performance is approximately 0.05 dB better than split-row min-sum, and 0.05 dB and 0.1 dB worse than MTFM-based stochastic and RHS, respectively. However, IDB requires an iteration limit of 315 to reach this level of performance, which is much higher than most other architectures. As a result, the worst-case latency and minimum throughput of IDB compare poorly. However, the MTFM-based stochastic decoder of [71], with an iteration limit of 400,

Table 5–4: Post-Layout Design Results for IDB and Comparisons With Other Works

	This work		[64]		[60]		[90]		[71]	[72]
Decoding algorithm LDPC code	IDB (2048,1723)		Split-row MS (2048,1723)		Offset MS (2048,1723)		Layered OMS (2048,1723)		Stoch. MTFM (2048,1723)	RHS (2048,1723)
Technology	65 nm		65 nm		65 nm		90 nm		90 nm	65 nm
Quantization bits	6		5		4		4		6	4
Area (scaled to 65 nm) (mm^2)	1.44		4.84		5.35		5.35 (2.79)		6.38 (3.33)	4.41
Utilization	95%		97%		84.5%		84.4%		95%	94.4%
Decoding iterations	315 (7×45)		11		8 + 6 post proc.		4		400	50
E_b/N_0 at BER = 10^{-7} (dB)	4.5		4.55		4.25		4.4		4.45	4.4
Supply voltage (V)	1.0	0.8	1.3	0.7	1.2	0.7	1.2	0.8	1.0	1.0
Clock frequency (MHz)	520	350	195	35	700	100	137	85	500	448
Min. throughput (Gbps)	3.38	2.28	36.3	6.52	14.9	2.13	11.7	7.23	2.56	9.18
Av. throughput (4.55 dB) (Gbps)	126.3	85.0	92.8	16.6	-	-	11.7	7.23	-	-
Av. power (4.55 dB) (mW)	462	192	1359	62	-	-	-	-	-	-
Av. energy (4.55 dB) (pJ/bit)	3.65	2.26	14.6	3.7	-	-	-	-	-	-
Av. throughput (5.5 dB) (Gbps)	171.8	115.6	-	-	47.7	6.67	11.7	7.23	61.3	160
Av. power (5.5 dB) (mW)	478	199	-	-	2800	144	-	-	-	-
Av. energy (5.5 dB) (pJ/bit)	2.78	1.72	-	-	58.7	21.5	-	-	-	-
Scaled energy (pJ/bit)‡	3.65* / 2.78†		8.64*		40.8†		-		-	-
Scaled throughput (Gbps/mm^2)§	87.7* / 119.3†		19.2*		8.92†		4.19†		18.4†	36.3†

* $E_b/N_0 = 4.55$ dB † $E_b/N_0 = 5.5$ dB ‡ Energy scaled to 1.0 V.

§ Average throughput per unit area scaled to 65 nm at constant frequency.

Table 5–5: Wiring Complexity of the Designed Decoders

Decoder	Tot. wire length (m)	Tot. interleaver wire length (m)	Interleaver wiring per edge (μm)
IDB (2048,1723)	10.957	5.055	411.36
DD-BMP (273,191)	7.920	3.586	772.69
MDD-BMP (273,191)	1.793	0.337	72.53
MDD-BMP (1023,781)	13.998	10.899	332.91
MDD-BMP (4095,3367)	225.687	207.517	791.81

is the highest out of all the surveyed decoders. Additionally, as noted in [108], a significantly higher minimum throughput can be guaranteed with input buffering.

Table 5–5 shows a comparison of the wiring complexity of the IDB decoder with the DD-BMP and MDD-BMP decoders. The total wiring length of the IDB decoder is 10.957 m, which is 64% lower than the 30.598 m reported for the same code in [60]. The reduction can be attributed to the smaller die size of IDB, as well as the use of broadcasting for inter-node (interleaver) wires. These results also clearly demonstrate the wiring advantage of MDD-BMP and IDB over DD-BMP. The DD-BMP $(273, 191)$, MDD-BMP $(1023, 781)$, and IDB decoders have similar die sizes and utilization factors, but DD-BMP has a much higher average interleaver wire length, due to the need to send unique VN-to-CN messages.

IDB also demonstrates major improvements in the iterative decoder performance metrics defined in [79]. IDB achieves energy per decoded bit of 2.78 pJ/bit at E_b/N_0 = 5.5 dB, and 3.65 pJ/bit at E_b/N_0 = 4.55 dB. This is 14 times lower than the offset min-sum decoder of [60], and 58% lower than split-row min-sum at the same values of E_b/N_0. Finally, in terms of throughput per unit area, IDB's result of 119.3

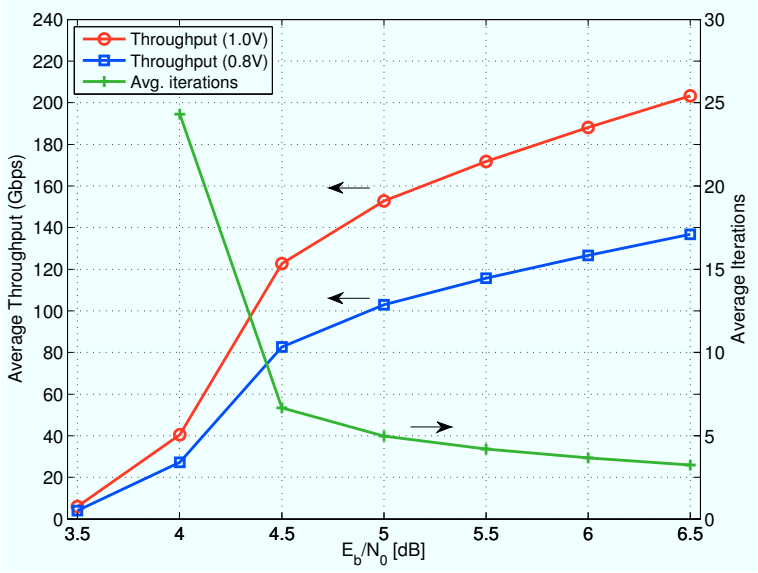


Figure 5–16: Average throughput and iteration count for the designed IDB decoder.

Gpbs/mm^2 at 5.5 dB is 3.3 times greater than RHS, and 87.7 Gpbs/mm^2 at 4.55 dB is 4.6 times greater than split-row min-sum.

If energy consumption due to off-chip I/O and data loading is taken into account using the method described in Section 4.4.1, the energy per decoded bit increases by 75.5 pJ/bit for $w = 1$, 5.06 pJ/bit for $w = 16$, and 2.52 pJ/bit for $w = 32$.* Since IDB decoding requires few iterations on average, the energy cost of loading is many times higher than the decoding process itself if w is small. However, we also note that [64] does not include data loading or off-chip I/O, and [60], while fabricated,

* The datasheets for the TSMC 65 nm I/O library used for this design do not include information on dynamic power consumption, so these results are estimated using values from the 130 nm library used in Chapter 4 [93], scaled quadratically by process size and supply voltage.

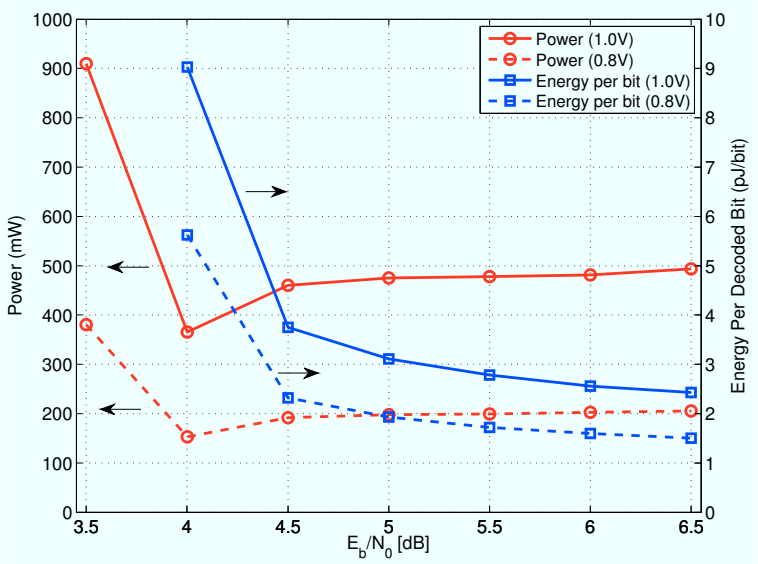


Figure 5–17: Average power consumption and energy per decoded bit for the designed IDB decoder.

contains on-die test and characterization logic that performs test vector generation and FER/BER, and so the energy consumption due to I/O and data loading is expected to be much smaller than if the L_v and h_v data were communicated off-chip. However, it is not clear whether this characterization circuitry is included in the reported results for power and energy, or whether the decoder core is on a separate power supply.

Figure 5–16 plots the average throughput and average number of decoding iterations for the designed IDB decoder over a range of values for E_b/N_0. Throughput is low at low values of E_b/N_0, due to the high decoding iteration limit and a large proportion of frames failing to decode. However, throughput increases greatly over the waterfall region, as the average number of decoding iterations decreases. This result demonstrates the large difference between average and worst-case performance

for IDB - despite having a maximum iteration limit of 315, the average is under 7 for E_b/N_0 above 4.5 dB.

The average power consumption and energy efficiency of the IDB decoder are plotted in Figure 5–17. Power consumption is high at low E_b/N_0, because degeneration in deadlocked VNs causes interleaver wires to toggle continuously between 0 and 1. Power is lower in the waterfall region, but afterwards increases monotonically due to the increasing frequency of loading new frames into the decoder, which consumes more power than ordinary decoding iterations. Energy per decoded bit decreases monotonically with E_b/N_0, due to the decreasing average number of decoding iterations (and thus computational effort) needed to decode each frame.

In addition, the low area and power consumption of IDB decoders make designs with multiple parallel decoders a practical possibility, which we refer to as a multi-core decoder. We note that an IDB decoder architecture with 2 or 3 cores would have a lower area than [64] or [60], assuming the impact of overhead circuitry is insignificant. If each processing core is assigned a different frame to decode, an n-core decoder would increase the average throughput by a factor of n over a single-core decoder, while throughput per unit area and energy efficiency remain roughly constant. Furthermore, because the relaunching technique of IDB consists of several independent decoding attempts, each core could also be assigned to work on different phases of the same frame, reducing the maximum decoding latency and increasing minimum throughput - an advantage that is not available to most other decoding algorithms.

5.9 Summary

This section has introduced the IDB algorithm, and presented LDPC decoders using the DD-BMP, MDD-BMP, and IDB algorithms. We demonstrate fully parallel designs of $(273, 191)$, $(1023, 781)$, and $(4095, 3367)$ FG-LDPC codes in 65 nm CMOS, overcoming the implementation complexity of these powerful codes. These decoders achieve BER performance up to 0.5 dB better than MSA, as well as low complexity, high throughput, and excellent energy efficiency. Our design using the $(2048, 1723)$ IEEE 802.3an LDPC code using IDB achieves BER performance within 0.25 dB of offset MSA, and has much lower area, higher throughput, and greater energy efficiency than other decoder architectures with similar error correction performance. The MDD-BMP decoder for the $(273, 191)$ code achieves an area of 0.28 mm^2, throughput of 37 Gbps, energy efficiency of 4.9 pJ/bit, and BER performance within 0.2 dB of MSA at a bit error rate of 10^{-6}. For the $(1023, 781)$ code, the MDD-BMP decoder achieves an area of 1.38 mm^2, throughput of 75 Gbps, energy efficiency of 13.2 pJ/bit, and BER performance 0.4 dB better than MSA. Finally, the decoder for the $(4095, 3367)$ code achieves an area of 15.37 mm^2, throughput of 141 Gbps, energy efficiency of 38.0 pJ/bit, and BER performance 0.5 dB better than MSA. Because these designs achieve very high throughputs, VFS can be applied to reduce their power consumption while still maintaining sufficient throughput for high-speed communications applications. Compared to other recent energy-efficient LDPC decoder designs, DD-BMP, MDD-BMP, and especially IDB offer favorable trade-offs for area, throughput, error correction performance, and power consumption. Due to

the low area of IDB relative to other designs, multi-core decoder architectures are feasible, and would increase both the minimum and average decoding throughput.

CHAPTER 6
Gear-Shift Decoder Designs

6.1 Introduction

As we have seen in previous works and in the previous chapters of this dissertation, one of the most effective ways to improve the energy efficiency of an LDPC decoder is to use a less complex decoding algorithm. Good heuristic decoding algorithms, such as split-row min-sum [64] and the IDB algorithm described in Chapter 5, can achieve greatly reduced complexity and increased energy efficiency at the cost of only a small amount of error correction performance. Very simple decoding algorithms, such as syndrome decoding and Gallager A, use the least possible amount of energy, but sacrifice a large amount of these codes' error correcting capabilities.

This compromise can be avoided by making a decoder capable of using multiple algorithms. Consider for instance a decoder implementing 2 algorithms: a low-complexity algorithm with poor error correction performance, and a high-complexity algorithm with excellent error correction performance. Decoding is first attempted with the low-complexity algorithm. If it fails, decoding is attempted with the high-complexity algorithm. Assuming failure of the first algorithm is a relatively uncommon event, the average energy cost per decoded frame of the second algorithm is very low, since it is rarely used. The composite decoder therefore has average energy efficiency similar to the low-complexity algorithm, while having the error correction performance of the high-complexity algorithm.

The concept of multi-algorithm iterative decoders was introduced in [106] as "gear-shift decoding". Gear-shift decoders are defined as decoders that can switch between a set of different decoding rules while a frame is being decoded. Several previous decoders that have multiple decoding "phases" thus qualify as gear-shift decoders, such as [60] and [71] with their post-processing phases. IDB employs the gear-shift concept with relaunching, as the VN update rule varies between phases. Even Gallager B, with its variable threshold parameter, can be considered a gear-shift algorithm.

Previous designs of gear-shift or multi-phase decoding have mostly had the goal of optimizing convergence time or improving error correction performance, and thus run the higher-complexity algorithms first, followed by the lower-complexity ones. For instance, the tri-mode decoder proposed in [109] employs three different algorithms with a wide range of complexity and a high degree of hardware commonality: a one-step majority algorithm, and two differential binary algorithms. However, it does not investigate operation as a gear-shift decoder. A brute-force approach to a multi-algorithm decoder can be found in [110], where multiple parallel decoders using different algorithms work simultaneously. A variable quantization scheme is proposed in [47], in which the dynamic range and precision of inter-node messages is variable from one iteration to another, improving BER performance. Since the total number of quantization bits is constant, this technique does not have a significant impact on energy consumption. We instead propose the reverse approach, with the goal of minimizing energy consumption: begin with lower-complexity algorithms, and switch to higher-complexity algorithms as decoding proceeds. To the best of our

knowledge, the only gear-shift decoder design focused on improving energy efficiency is the variable-precision split-row min-sum design presented in [111]. This decoder first attempts split-row min-sum decoding with 3 quantization bits, then switches to 4 bits after a small number of iterations.

One drawback of gear-shift decoding is that extra logic is needed to implement multiple decoding rules, especially in the straightforward case where each algorithm is implemented as a dedicated decoder. The split-row min-sum decoder mentioned above is an example - extra multiplexing logic and adaptations are required to support variable quantization, and the resulting decoder is 5% larger and 8% slower than a conventional split-row min-sum decoder [64], for a 9% reduction in energy per bit.

However, depending on the choice of decoder architecture and algorithms, different decoding rules can be implemented with minimal overhead. This can arise from hardware commonality between different algorithms, or an intrinsic ability of a given algorithm to use variable quantization without significant logic overhead. One example of the latter is bit-serial min-sum, where it was noted that varying the number of quantization bits in inter-node messages from one iteration to another could be done for little added cost [58]. However, this idea was not explored any further.

Likewise, two of the algorithms presented in this dissertation - IDB and PWM-MS - are highly amenable to gear-shift decoding. PWM-MS can support variable message width without any changes whatsoever to the node circuitry - we call this a GSP decoder, for "gear-shift pulse-width". Combination IDB/GSP (or IGSP, "IDB

with gear-shift pulse-width") decoders can also be implemented with modest additional computational complexity over a conventional PWM-MS decoder, and allow application of the gear-shift concept to greatly reduce average energy consumption.

The remainder of this chapter will describe gear-shift decoding schedules for the GSP and IGSP decoders, their hardware architectures, and ASIC post-layout design results.

6.2 Efficiency of Gear-Shift Decoding

Consider a gear-shift decoder with N "gears", or decoding stages. Assuming that the stages are attempted in order from lowest energy consumption to highest, and these stages are labelled from 1 to N, the average energy consumed in decoding a frame is:

$$E_{avg} = \sum_{n=1}^{N} P_n \cdot E_n \tag{6.1}$$

where P_n is the proportion of frames in which stage n is begun, and E_n is the average energy consumed during that stage when it is invoked. An alternate form of this equation is:

$$E_{avg} = E_1 + \sum_{n=2}^{N} \left(\prod_{\nu=1}^{n-1} FER_\nu \right) \cdot E_n \tag{6.2}$$

where FER_ν is the failure rate (or frame error rate) of stage ν, *given that all previous stages have been attempted and failed.* $P_1 = 1$ because stage 1 is always attempted for every frame.

From Equation 6.2, it is clear that in practical usage (i.e., the decoding algorithms, LDPC code, and decoder parameters are chosen so that most frames are

decoded successfully under given channel conditions), E_1 will be the dominant term as the FER products will greatly reduce the contributions from successive E terms. Thus, the average energy consumption of the overall decoder is determined by the energy consumption of the first stage. However, the error correction performance is determined by the individual stage with the best performance (BER/FER performance equal to or better than the best individual stage can be guaranteed by re-initializing at each stage with the initial channel inputs). Thus, the main appeal of gear-shift decoding is that it can achieve the average energy efficiency of a low-complexity algorithm, like IDB, while also having the error correction performance of a high-complexity algorithm, like offset MSA.

However, this does not take into account the overhead incurred by implementing multiple decoding algorithms in hardware. In a straightforward design, in which stage is implemented as a separate decoder, most stages would sit idle for all but a small fraction of time. Since the later stages are the ones using more complex algorithms, and thus having greater silicon area, the resulting decoder would have very poor area efficiency. This also impacts energy efficiency, since the unused stages consume leakage power (power gating them would not be terribly practical, since they must be ready to operate quickly if they are needed). Thus, a high degree of hardware commonality between the different stages is important, but even if this is the case, hardware efficiency will be limited by the necessity of leaving portions of the circuit idle during decoding with the less complex stages.

6.3 The GSP and IGSP Decoding Algorithms

In order to support a variable maximum message width, the min-sum variable node update of Equation 2.14 is changed to:

$$M_{v\to c}^{(k)} = L_v + \sum_{c'\in C_i\setminus c} m_{c'\to v}^{(k-1)} \tag{6.3}$$

$$m_{c\to v}^{(k)} = sgn_r\left(M_{v\to c}^{(k)}\right)\cdot \min\left(m_{max}^{(k)},\ \left|M_{v\to c}^{(k)}\right|\right) \tag{6.4}$$

Note that Equation 6.3 is identical to Equation 2.14, save for $m_{v\to c}$ (the message from VN v to CN c) being replaced with $M_{v\to c}$ (the memory associated with that message). In other words, the internal computations function identically to PWM-MS, and it is only the inter-node messages that are affected. Equation 6.4 determines the actual message, reducing its magnitude to a maximum of $m_{max}^{(k)}$, which is the maximum message magnitude at iteration k. It is also worth nothing that no scaling is ever applied to L_v, regardless of the value of $m_{max}^{(k)}$. This is because simulations found that using the unscaled L_v resulted in better error correction performance.

In conventional PWM-MS, m_{max} is fixed at $2^q - 1 - \beta$, where β is the offset value for offset min-sum (our designs all use $\beta = 1$, which is a typical implementation value). In GSP, m_{max} is set to a low value for the first iterations, and is raised gradually until it reaches $2^q - 1 - \beta$, which is equivalent to offset min-sum with full q-bit precision. In other words, decoding begins with magnitude-limited message exchanges that consume less energy than using the decoder's full precision. If the frame does not converge to a valid codeword, the magnitude limit is incrementally increased on successive iterations, which increases the energy cost per iteration but

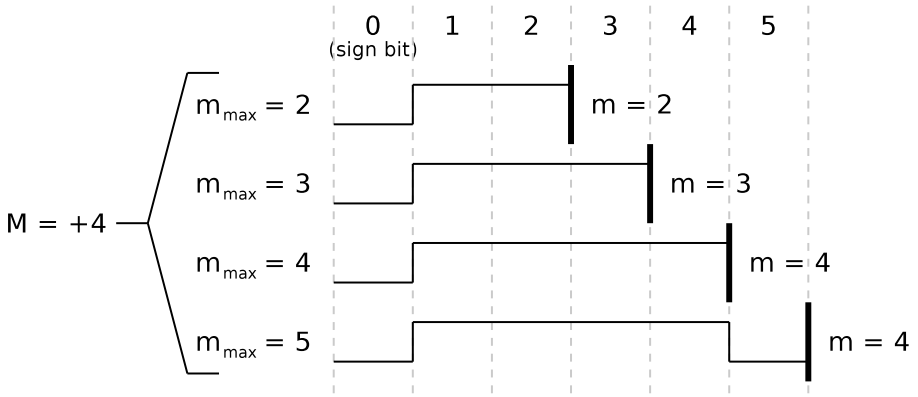


Figure 6–1: Example waveforms of inter-node messages in GSP decoding. A low value for m_{max} truncates high-magnitude messages, thus reducing information exchange and error correction performance, but also reduces the number of clock cycles per decoding iteration.

also the error correction capability of the decoder. Each of these precision levels can be considered a "stage" or "gear" n in Equations 6.1 and 6.2. Additionally, with lower values of m_{max}, iterations complete in fewer clock cycles. This is because on average, iterations with lower m_{max} have a higher impact on convergence normalized per clock cycle. In other words, using a lower m_{max} results in a lower convergence time than higher values of m_{max}.

Figure 6–1 shows examples of GSP messages, and demonstrates the truncating effect of m_{max} values that are lower than the maximum value that can be contained in the $M_{v\to c}$ memories. This figure also demonstrates how the pulse-width sign-magnitude format supports non-power-of-2 message widths, unlike conventional binary number encoding, which allows for more "gears" and more gradual transitions in message width.

The IGSP algorithm uses IDB as its first stage (or stages, if relaunching is used), then switches to GSP for the final stages. IDB offers considerable power savings over min-sum based algorithms, since it requires only a single memory in the VN compared to one per edge of the code's graph connecting to the VN - these memories can be disabled with clock gating while the decoder is operating in IDB mode. However, additional circuitry is required in the VNs to support IDB decoding, so we can expect IGSP decoders to have a higher silicon area for a given maximum clock frequency compared to GSP.

6.3.1 Parameter and Schedule Design

In general, gear-shift algorithms have a much larger optimization space than non-gear-shift algorithms, since the former may make use of multiple decoding algorithms, message-passing resolution levels, or internal arithmetic resolution levels.

One technique that can be used to analytically optimize an iterative decoder is density evolution [41]. Density evolution compares the Gaussian "densities" of messages from one iteration to the next, and so provides a quantitative measure of progress made in error correction. Density evolution has, in fact, been used to optimize parameters and compare the relative performance of different decoding algorithms [112], including optimization for low energy consumption [80].

It may be possible to use density evolution to optimize decoding schedules for the GSP and IGSP algorithms. However, the variable message resolution of these gear-shift algorithms causes difficulties with the use of "classical" density evolution. Adaptations may be possible, but this would constitute a major research effort, and furthermore the focus of this chapter is on circuit implementation issues - in

Table 6–1: GSP decoding schedule

Phase	Iterations	m_{max}	Clock cycles
1	6	2	18 (24)*
2	5	3	20 (25)*
3	4	4	20 (24)*
4	3	5	18 (21)*
5	2	6	14 (16)*
Total	20	—	90 (110)*

* Numbers in parentheses are for pipelined GSP.

Table 6–2: IGSP decoding schedule

Phase	Algorithm	Iterations	m_{max}	Clock cycles
1	IDB	15	—	15
Reset all $M_{v\to c}$ to original L_v				1
2	Offset MS	6	2	18
3	Offset MS	5	3	20
4	Offset MS	4	4	20
5	Offset MS	3	5	18
6	Offset MS	2	6	14
Total	—	35	—	106

particular, hardware commonality between the different decoding phases, and relative performance compared to other decoders. We therefore opt to leave this theoretical optimization as future work.

We have instead devised an empirical procedure for developing "good" decoding schedules, similar to the procedure for the IDB algorithm described in Section 5.7.1. These decoding schedules for GSP and IGSP using the $(2048, 1723)$ 10 gigabit Ethernet LDPC code are summarized in Tables 6–1 and 6–2 respectively. These schedules

are based around decoders with $q = 4$ (as in other min-sum decoders of this code such as [60] and [90]) and clipping threshold $T_{clip} = 8.0$.

Despite the very large design space for GSP and IGSP, two guidelines can be applied to drastically reduce it. The first is that, according to the gear-shift principle, decoding should begin with the simplest available algorithm, and end with the most complex. The second is that the simpler algorithms should be used until the next more complex one would be considerably more effective. This will minimize energy consumption and maximize throughput, as the vast majority of frames will be decoded by the earlier phases, which involve less computation and use fewer clock cycles per iteration.

In addition to minimizing energy consumption and maximizing throughput, we also require that error correction performance be equal to that of the most complex phase, or at least have no significant loss. The metric we use is that if the overall decoder has k_{max} iterations across all decoding phases, it should achieve similar performance to k_{max} iterations of the most complex algorithm.

One final consideration is maximum latency. This is a low priority in these designs, but it must still be constrained to a practical value. We chose 105 clock cycles as a soft upper limit, since this is the maximum latency of the PWM-OMS decoder presented in Chapter 4. The final schedules ended up exceeding this limit slightly, being 106 clock cycles for IGSP and 110 clock cycles for pipelined GSP.

The schedules of Tables 6–1 and 6–2 were then developed according to these guidelines, and optimized for average convergence time by trial and error. Although our highest priority optimization goal is energy efficiency, this cannot be accurately

modeled in such high-level simulations. Thus, energy consumption is modeled as the average number of clock cycles required for convergence. This optimization procedure has the additional benefit of maximizing throughput.

For the GSP schedule in Table 6–1, decoding begins with 6 iterations of OMS with $m_{max} = 3$. This is sufficient to successfully decode most frames in the waterfall region. Should decoding fail to converge, successive phases increase m_{max} while reducing the number of iterations. The final phase applies 2 iterations of full 4-bit OMS. GSP can be also pipelined in the same way as a standard PWM-MS decoder, which adds 1 clock cycle of latency per decoding iteration. Table 6–1 shows clock cycle counts for pipelined GSP in parentheses.

For the IGSP schedule in Table 6–2, decoding begins with 15 iterations of IDB. Again, this number was chosen so that most frames in the waterfall region successfully decode during this phase. The parameters used for IDB decoding are $s = 0.5$ and $d = 1$. The IDB relaunching technique is not used in this schedule, since including it would require additional hardware, as well as add too much latency to justify its performance increase. If IDB fails to decode the frame, all VN output counters are reset to the original values of L_v, and decoding proceeds using the GSP schedule from Table 6–1. The reset is beneficial, since when the IDB phase fails to decode the frame, it generally leaves the decoder in a state less likely to decode successfully using GSP, and also requires more iterations on average to complete decoding.

Figures 6–2 and 6–3 respectively show histograms of decoding iterations for GSP and IGSP at $E_b/N_0 = 4.25dB$, with annotations showing the mean, and points of interest in the cumulative distribution function D. A total of $6 \cdot 10^7$ frames were

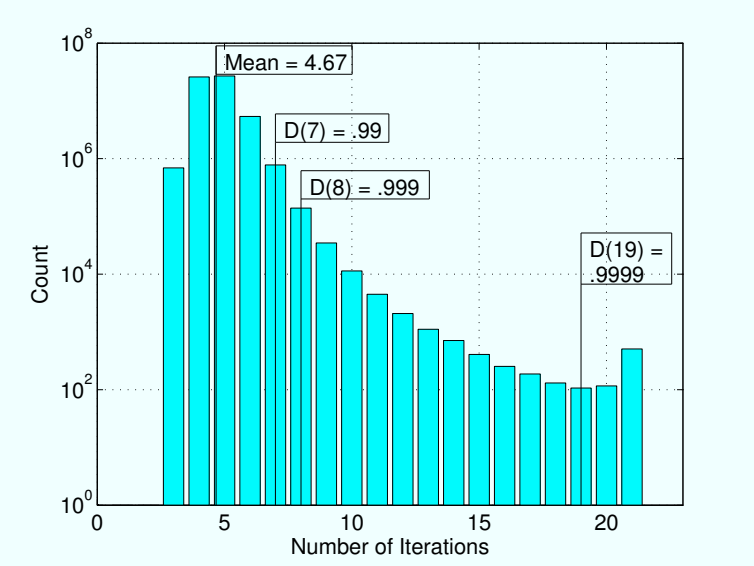


Figure 6–2: Iteration count histogram for GSP, using the decoding schedule in Table 6–1. The bar at 21 iterations shows frames that failed to decode within 20 iterations, and are counted as 20 in statistical calculations.

simulated for each histogram. These results demonstrate how each decoding phase is designed so that it successfully decodes most frames it begins, in accordance with the above guidelines. One particularly notable characteristic of Figure 6–3 is that the distribution has separate peaks for the IDB and GSP decoding portions. For IGSP, we found it worthwhile to continue IDB for several iterations beyond the peak, as IDB is very "cheap" in terms of time and energy compared to GSP. Other histograms similar to these, using different parameters, were used to aid in the optimization of these schedules by finding the set of parameters which resulted in the lowest mean without compromising error correction performance.

Figure 6–4 plots the simulated BER/FER performance of GSP using this code and schedule, along with the IDB phase of IGSP, and a baseline of standard OMS

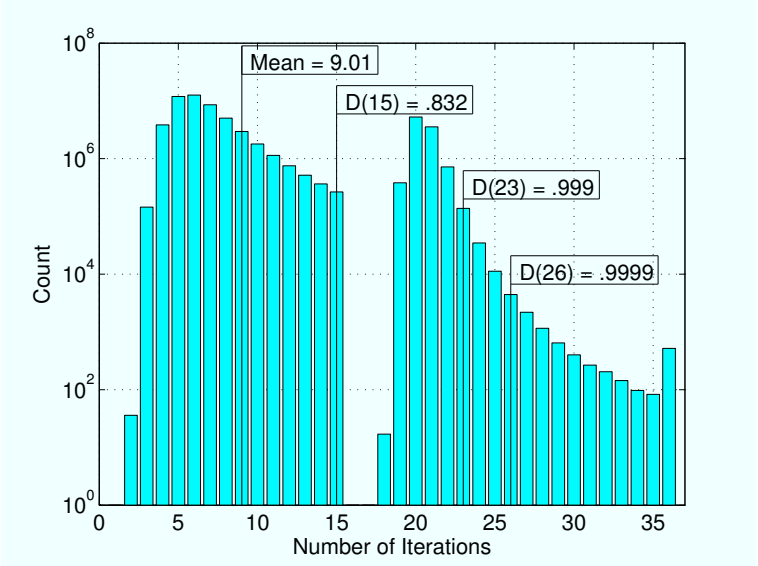


Figure 6–3: Iteration count histogram for IGSP, using the decoding schedule in Table 6–2. The bar at 36 iterations shows frames that failed to decode within 35 iterations, and are counted as 35 in statistical calculations.

with 20 iterations and $q = 4$. GSP actually achieves performance a very small amount better than OMS - this can be attributed to the message truncation having an effect similar to that of successive relaxation, and thus improving error correction performance [98]. IGSP achieves the same overall performance as GSP, since it resets to the original channel data after the IDB phase and then applies the same decoding schedule. Although the IDB phase has much poorer performance than GSP, its FER shows that it will successfully decode the vast majority of frames for $E_b/N_0 \geq 4.5dB$ and will therefore considerably increase energy efficiency in this region as per Equation 6.4.

The average decoding iterations and clock cycles required for GSP, IGSP, and OMS using these schedules are plotted in Figure 6–5. The clock cycle count for OMS

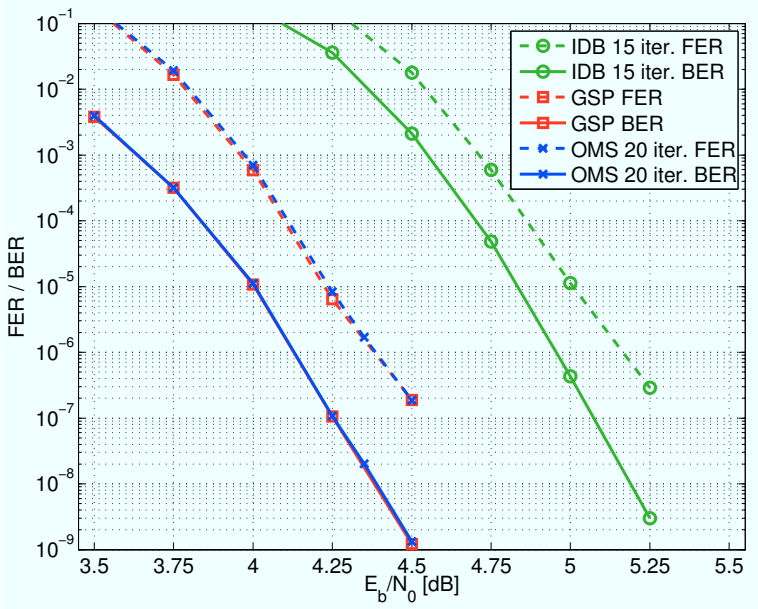


Figure 6–4: BER/FER for GSP, standard OMS, and the IDB phase of IGSP. The overall BER/FER performance of IGSP is equivalent to GSP.

is determined using a standard PWM-MS decoder. Clock cycle plots are displayed for both the unpipelined and pipelined versions of GSP and PWM-MS. Since pipelining adds 1 clock cycle of latency per iteration to both GSP and PWM-MS, the pipelined decoders require more clock cycles to converge. While the iteration counts for OMS and GSP are nearly identical, the average clock cycle count for GSP is approximately half that of OMS. Due to its IDB decoding phase, IGSP has the highest average iteration count, but also the lowest average clock cycle count for $E_b/N_0 \geq 4.5dB$, where the vast majority of frames complete decoding before reaching the min-sum phases.

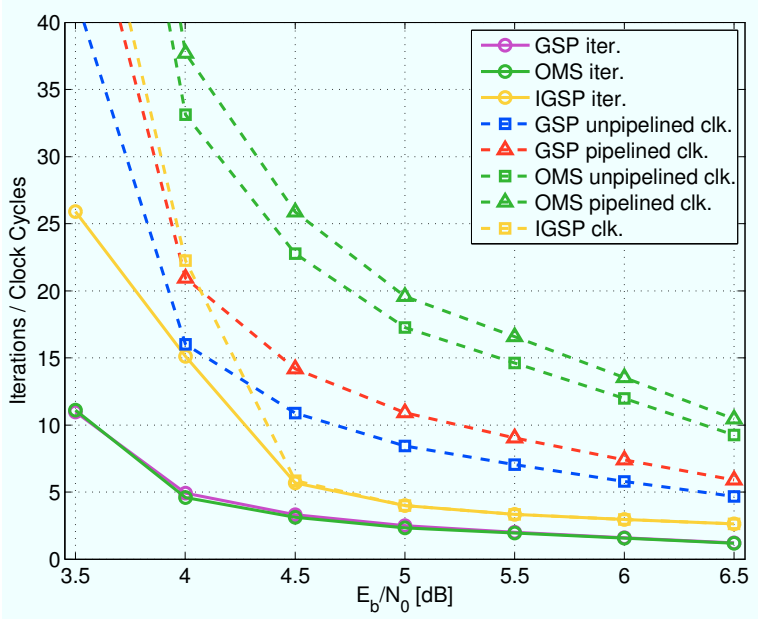


Figure 6–5: Average number of decoding iterations and clock cycles per decoded frame for GSP, IGSP, and standard OMS.

6.4 Circuit Architecture of GSP and IGSP Decoders

As mentioned previously, GSP has a unique advantage in gear-shift decoding in that the width of the inter-node messages can be varied without modifying the node hardware. The only difference between GSP and conventional PWM-MS is the controller state machine - the controller starts a new iteration after $m_{max} + 1$ clock cycles, where m_{max} is the maximum inter-node message magnitude for that iteration. Thus, for GSP, the VN and CN hardware is identical to PWM-MS.

For IGSP, the PWM-MS check node can also be used without modification, since the IDB check node function is equivalent to the CN sign computation phase in PWM-MS. However, the variable node requires additional hardware to support both IDB and GSP operation. Figure 6–6 shows the circuit schematic for an IGSP

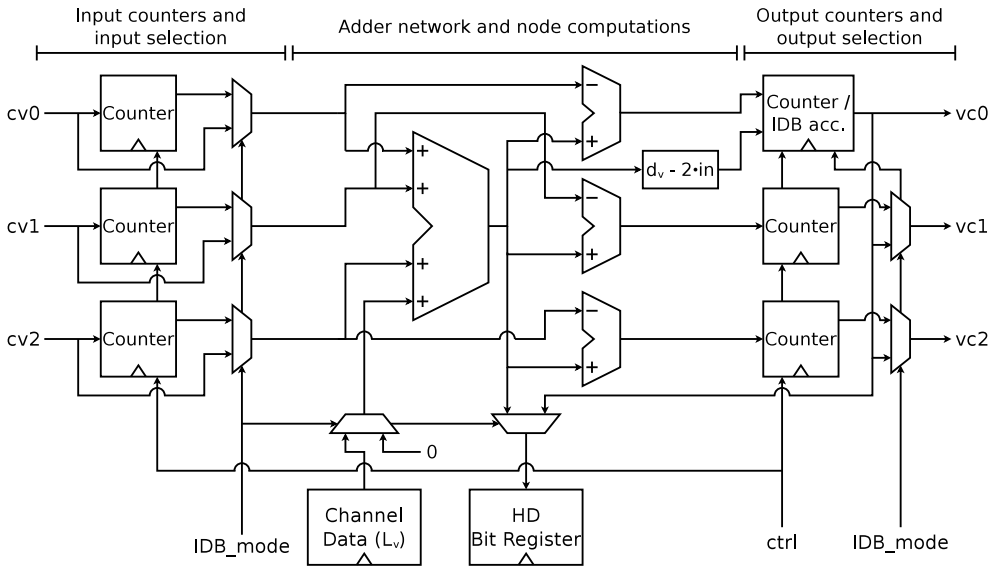


Figure 6–6: Schematic for an IGSP variable node with degree $d_v = 3$.

variable node with degree $d_v = 3$. When operating in IDB mode, the input counters are bypassed. The incoming messages $m_{c \to v}$ are zero-padded to q bits in width and input to the adder network. The delta value for the accumulator is calculated from the global sum with the function $\Delta = d_v - 2 \cdot in$. This converts the result of addition in the $\{0, 1\}$ domain to the $\{1, -1\}$ domain used in IDB messaging. One of the output counters is replaced with a combination IDB accumulator and GSP down counter. In IDB mode, this module functions as an accumulator, applying the input as a delta (minus the IDB degeneration factor) to its previously stored value, and outputting its sign bit, which is distributed to all outgoing edges. In GSP mode, it functions as a normal down counter. The adder network input for L_v is replaced with zero in IDB mode, since the initial channel data does not participate in the VN update in IDB. Finally, the hard decision bit d_v is taken from the sign of the

accumulator while in IDB mode, whereas in GSP mode it is taken from the sign of the global sum in the adder network.

The IGSP VN extensively uses clock gating to reduce dynamic energy consumption in IDB mode. Only the combination accumulator/counter and d_v register are active in IDB mode - all other output counters, all input counters, and the L_v register are clock gated, meaning only $q+1$ registers are enabled in each VN. For our designs using the $(2048, 1723)$ LDPC code (which has $d_v = 6$) and $q = 4$, this translates to 5 out of 53 registers active, for a reduction of 91%. In GSP mode, however, the full width of each counter is always enabled during decoding, even if it is not necessary to store the maximum possible message of m_{max}. This is because only 1 bit from each input counter can be gated in our GSP schedule, and doing so would require additional logic to support signed counting over multiple word widths, giving a poor tradeoff between area, speed, and energy savings.

This IGSP decoder can be pipelined in a manner similar to GSP by passing the VN inputs through the input UDCs, rather than bypassing them entirely. However, due to IGSP's IDB mode, this design would have additional drawbacks that are not present in a pipelined GSP decoder. One is that it requires an active register at each VN input. The designed decoder has 5 active registers per VN in IDB mode: 4 in the combination accumulator/counter, and 1 for the hard decision bit. Pipelining would add 6 more for $(2048, 1723)$ LDPC code, more than doubling the number of active registers in the decoder during IDB mode. Since our primary design goal is energy efficiency, we wish to keep the number of active registers to a strict minimum. Secondly, the added cycle of latency would have a much more pronounced

effect on IDB than on GSP, raising the number of clock cycles per iteration from 1 to 2. Alternatively, the IDB algorithm could be altered to accommodate this extra cycle of latency by using the $M_{v\to c}$ values from 2 cycles previously, and thus adding 1 clock cycle of latency to the beginning of the IDB decoding phase, rather than for every iteration. However, this would require additional logic to implement in hardware, and the effect of this alteration on the BER performance of IDB has not been investigated. Therefore, we do not investigate pipelined IGSP in this work.

6.5 Design Results

As with the DB decoders, we have designed the previously described GSP and IGSP decoders in TSMC 65 nm 7LM single-V_t CMOS. We present ASIC post-layout results for three different decoder designs: unpipelined and pipelined versions of GSP using the decoding schedule in Table 6–1, and an IGSP decoder using the schedule in Table 6–2. All decoders are fully parallel, use the $(2048, 1723)$ LDPC code from the 10 Gbps Ethernet specification, and $q = 4$ bits of quantization for channel input values. As before, post-layout simulations are used to characterize the silicon area, throughput, and power consumption of these decoders.

In addition, the two GSP decoders can be run in "standard PWM-MS" mode, in which all iterations use the full 4 bits of precision for message passing. Since this requires only a change to the controller state machine, the hardware overhead is negligible. Standard PWM-MS mode was used to collect results for OMS, which we present as a baseline for comparison with GSP and IGSP.

A detailed system diagram showing the critical paths of GSP is shown in Fig. 6–7. The red line traces the critical path of the unpipelined decoder, while the

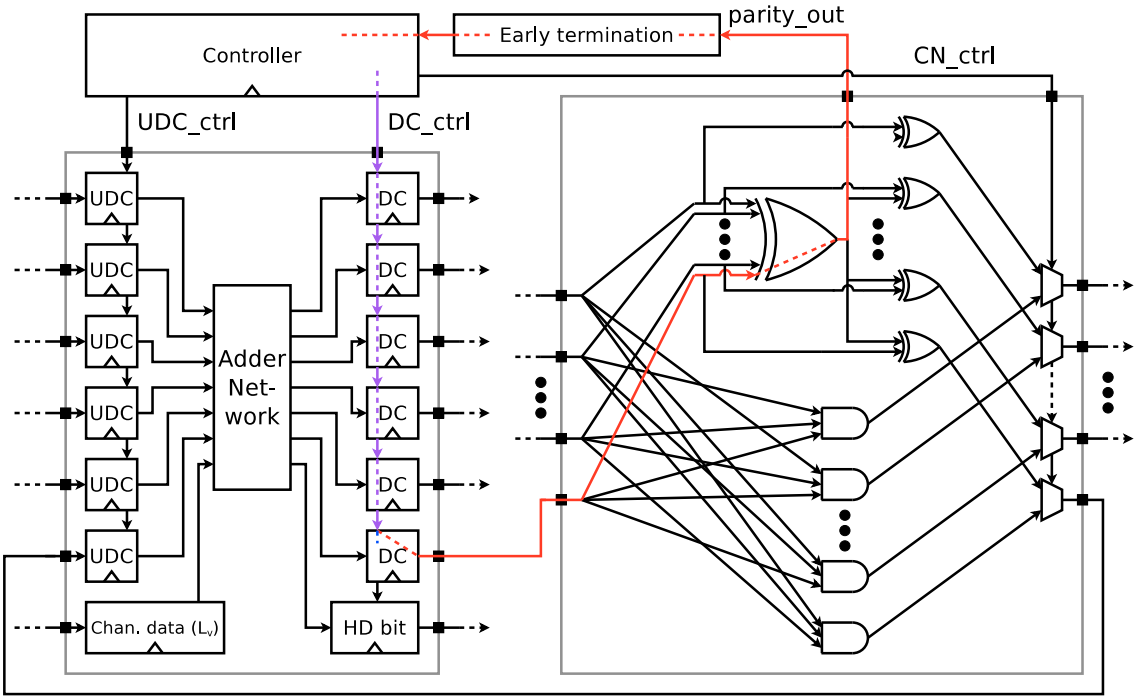


Figure 6–7: Detailed block diagram of the GSP decoders showing the critical paths of the unpipelined decoders (in red) and the pipelined decoder (in blue). A purple line is used where the two intersect.

purple line traces the segment in which the pipelined and unpipelined critical paths intersect. Although this decoder architecture is essentially identical to the PWM-MS decoder in Chapter 4, the critical paths are different due to the larger code and smaller manufacturing process used for GSP. For both the pipelined and unpipelined decoders, the critical path begins at the controller and goes through the control signal for loading the VN output counters. The critical path for the pipelined decoder ends here, while for the unpipelined decoder, it continues through to the check node, the early termination detection block, and back to the controller.

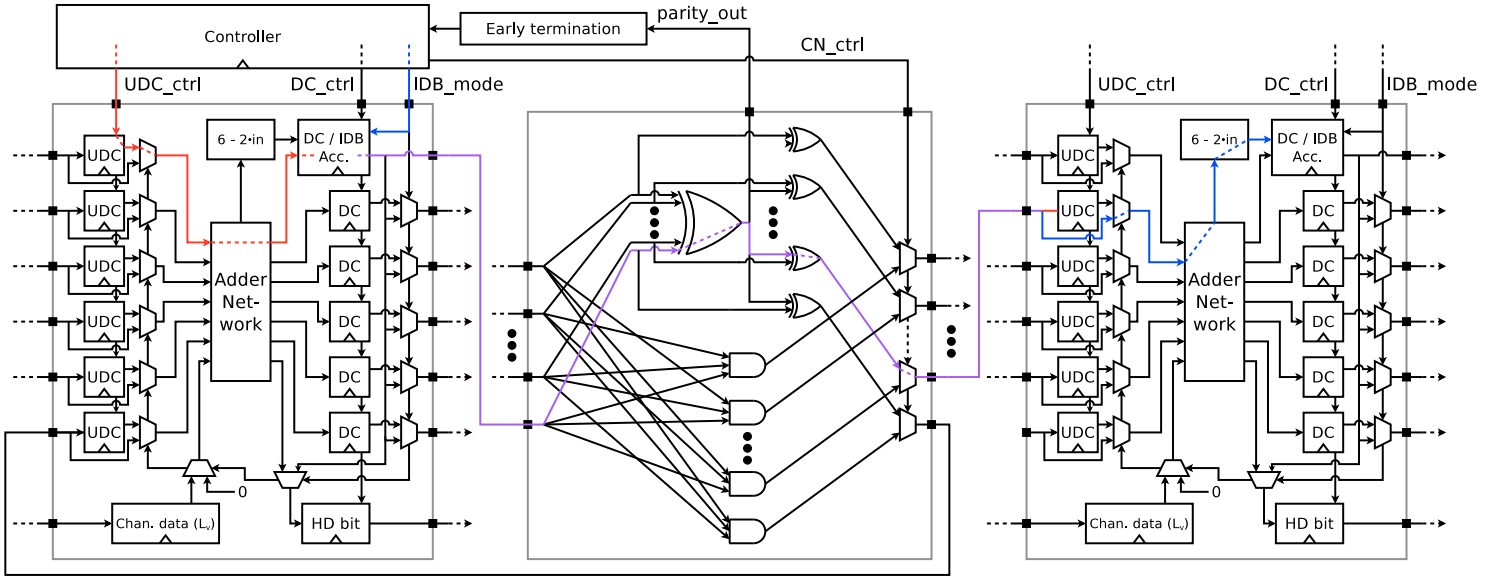


Figure 6–8: Top level block diagram of the IGSP decoder, showing a check node, 2 neighbouring variable nodes, and other top-level components. The critical path in GSP mode is traced in red, while the IDB mode critical path is traced in blue. A purple line is used for segments where they intersect.

Figure 6–8 shows the critical path for the IGSP decoder. The red line traces the critical path of the decoder while it is in GSP mode, and the blue line traces the critical path in IDB mode. The purple line shows where these critical paths intersect. Unlike the GSP decoders, the IGSP critical paths originate in a VN, pass through a CN and end in a different neighbouring VN. Since IGSP has extra logic and interconnect delays compared to GSP, this VN-CN-VN path has a longer total delay than the path through the early termination block.

The post-layout results for these designs are summarized in Table 6–3, along with data for relevant prior works. To the best of our knowledge, the only other LDPC decoder in the literature that employs the gear-shift approach to improve energy efficiency is the adaptive word-width split-row decoder in [111]. This decoder has a lower area and much better energy efficiency than GSP and IGSP, but the heuristic split-row min-sum algorithm results in a BER performance loss of 0.4 dB. The other decoders, [60] and [90], are included because they implement the offset min-sum algorithm without additional heuristics (although [90] has worse BER performance due to its very low iteration limit).

Compared to the offset min-sum decoder of [60], unpipelined GSP achieves 39% higher scaled throughput and an 8% improvement in scaled energy per bit, while pipelined GSP achieves an 85% improvement in scaled throughput and 4% lower energy per bit. Notably, these 3 decoders are highly similar in terms of silicon area and utilization.

IGSP further improves average throughput and energy efficiency, but at the cost of having 11% higher silicon area and a 10% lower clock frequency than unpipelined

Table 6–3: Post-Layout Results for GSP and IGSP Decoders, Including Comparisons With Other Works

	This work						[111]		[60]		[90]	
Decoding algorithm LDPC code	GSP (2048,1723)		GSP (2048,1723)		IGSP (2048,1723)		Adaptive split-row (2048,1723)		Offset MS (2048,1723)		Layered OMS (2048,1723)	
Technology	65 nm		65 nm		65 nm		65 nm		65 nm		90 nm	
Quantization bits	4		4		4		6		4		4	
Area (scaled to 65 nm) (mm^2)	5.29		5.29		6.00		5.10		5.35		5.35 (2.79)	
Utilization	87%		88%		91%		96%		84.5%		84.4%	
Decoding iterations	20		20		35		15		8 + 6 post proc.		4	
E_b/N_0 at BER = 10^{-7} (dB)	4.25		4.25		4.25		4.65		4.25		4.4	
Supply voltage (V)	1.0	0.8	1.0	0.8	1.0	0.8	1.3	0.7	1.2	0.7	1.2	0.8
Clock frequency (MHz)	290	200	475	320	262	175	185	40	700	100	137	85
Min. throughput (Gbps)	6.53	4.50	8.76	5.90	4.98	3.35	25.3	5.4	14.9	2.13	11.7	7.23
Av. throughput (Gbps)	65.8	45.4	88.1	59.4	100.3	66.9	85.7	13.5	47.7	6.67	11.7	7.23
Av. power (mW)	2465.7	1059.0	3465.8	1445.9	1464.3	608.0	1172	73	2800	144	-	-
Av. energy (pJ/bit)	37.5	23.3	39.3	24.3	14.6	9.1	13.6	3.9	58.7	21.5	-	-
Scaled energy (pJ/bit)*	37.5†		39.3†		14.6†		8.05‡		40.8†		-	
Scaled throughput ($Gbps/mm^2$)§	12.4†		16.7†		16.6†		16.8‡		8.92†		4.19†	

* Energy scaled to 1.0 V † $E_b/N_0 = 5.5dB$ ‡ Operating condition not specified

§ Average throughput per unit area scaled to 65 nm at constant frequency

GSP. Despite the lower clock frequency, due to its IDB decoding phase it needs fewer clock cycles on average to decode a frame (see Figure 6–5). Its average throughput of 100.3 Gbps at $E_b/N_0 = 5.5dB$ is the highest of all surveyed decoders, but due to its higher area, it has scaled throughput almost identical to pipelined GSP and [111]. Among offset min-sum decoders, it has the best energy efficiency by a significant margin - its scaled energy is 14.6 pJ/bit, which is 64% lower than [60] and 61% lower than unpipelined GSP.

If energy consumption due to off-chip I/O and data loading is taken into account using the method described in Section 4.4.1, the energy per decoded bit increases by 53.8 pJ/bit for $w = 1$, 3.49 pJ/bit for $w = 16$, and 1.68 pJ/bit for $w = 32$ (all three designs produced nearly identical results). As with IDB decoder presented in Chapter 5, the most relevant prior works also ignore these factors (as in [111] and [64]), or do not communicate L_v and h_v off-chip (as in [60]).

Figures 6–9, 6–10, and 6–11 respectively plot the average throughput, power consumption, and energy efficiency of the GSP and IGSP decoders, as well as baseline performance for standard OMS that were obtained by running the GSP decoders in PWM-MS mode. As before, these results were obtained via post-layout simulations using design kit libraries for both nominal ($V_{DD} = 1.0V$) and reduced ($V_{DD} = 0.8V$) supply voltages, including the clock tree, and using back-annotated delay and extracted RC parasitics.

The GSP decoders achieve considerably higher throughput than their PWM-MS counterparts, with the pipelined version showing 68% improvement at $E_b/N_0 = 5.5dB$, and the unpipelined version 84%. Power consumption for GSP increases

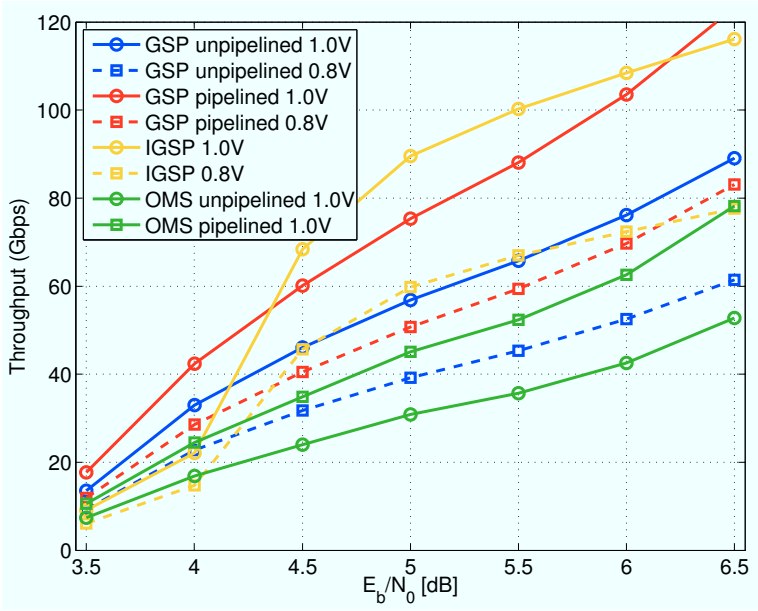


Figure 6–9: Average throughput for GSP, IGSP, and standard OMS.

monotonically with E_b/N_0, as the frequency of loading new frames increases. Energy efficiency is roughly 20% improved in GSP compared to PWM-MS. This is not as dramatic as the throughput improvement, since the sign computation phase and frame loading consume much more energy than the magnitude phase, and this cost is fixed between PWM-MS and GSP.

Somewhat surprisingly, pipelined GSP consumes only about 5% more energy than unpipelined GSP, despite having an additional clock cycle per iteration. However, both decoders perform the same computation, so the only extra energy expended in the pipelined decoder is in the clock tree and registers during the extra clock cycle. Furthermore, the higher clock frequency means leakage energy per decoded bit is lower in the pipelined decoder, which offsets some of the higher dynamic energy.

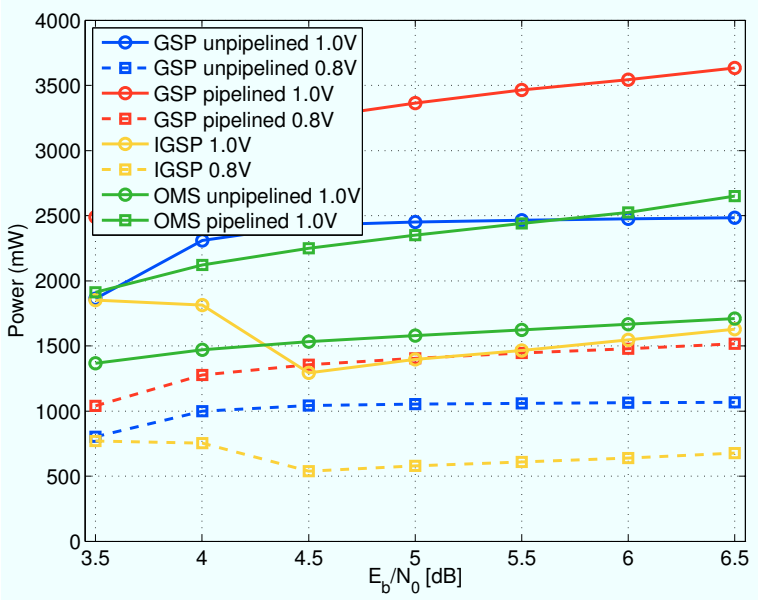


Figure 6–10: Average power consumption for GSP, IGSP, and standard OMS.

The IGSP decoder produces some interesting results because of its IDB decoding mode. Throughput is low and power consumption is high at low E_b/N_0, because IDB decoding fails for nearly all frames. At 4.5 dB, throughput suddenly shoots upward and power consumption goes downward. Referring back to Figure 6–4, approximately 98% of frames decode successfully during the IDB phase at this point, and so it can be said that this is the "turning point" at which IDB begins to dominate the throughput, power, and energy performance of the decoder. Beyond this point, IGSP has much lower energy consumption than GSP, although the extra overhead related to the inactive GSP components cause it to have 5.25 times as much energy consumption as a dedicated IDB decoder. Interestingly, at high values of E_b/N_0, pipelined GSP overtakes IGSP in throughput - this is due to GSP's higher clock frequency, and a rapid drop in the average number of iterations and clock cycles per frame for GSP.

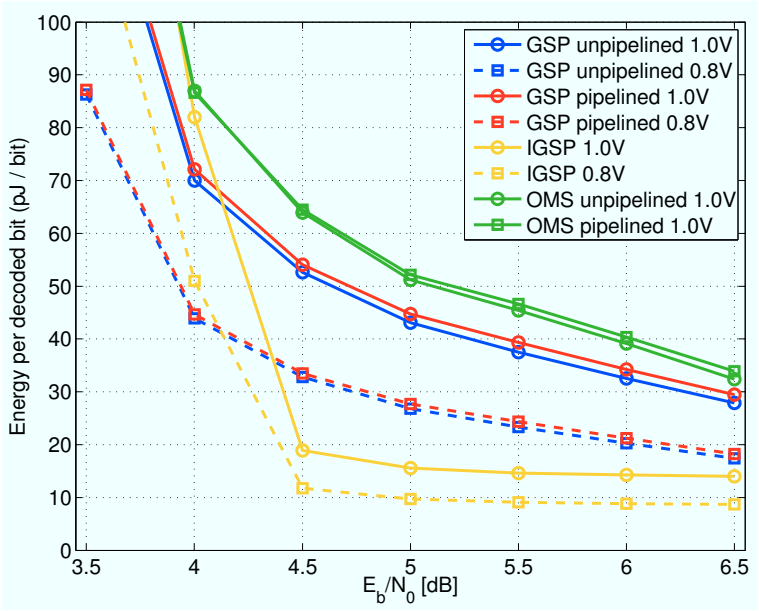


Figure 6–11: Average energy per decoded bit for GSP, IGSP, and standard OMS.

6.6 Summary

This chapter presented LDPC decoding algorithms and architectures using gear-shift techniques to improve energy efficiency. Gear-shift decoders can use multiple update rules during decoding, and therefore allow for energy reduction strategies in which simple, lower-energy algorithms are attempted first, followed by more complex algorithms if earlier attempts fail.

PWM-MS and IDB are naturally amenable to these techniques, because there is a high degree of hardware commonality between IDB and PWM-MS, and the message resolution of PWM-MS can be varied without any additional hardware in the processing nodes. This advantage is critical to achieving high area and energy efficiency. Without a high degree of commonality between different decoding phases, it is necessary to have hardware logic blocks that are idle while not being used, which

is a highly inefficient use of silicon. Hence, these two algorithms form the basis of our GSP (gear-shift pulse-width min-sum) and IGSP (IDB with GSP) decoding algorithms.

Our ASIC designs of GSP decoders achieve large improvements in area efficiency, and slight improvements in energy efficiency compared to other recent decoders which employ the offset min-sum algorithm without additional heuristics. IGSP achieves much larger improvements in throughput and energy efficiency without compromising error correction performance, but at the cost of having higher silicon area.

CHAPTER 7
Conclusion and Future Work

7.1 Advances

This dissertation has presented four separate works related to the energy efficient decoding of LDPC codes. These works present valuable novel contributions to decoder architecture and decoding algorithms, and make significant advances to the state of the art in these fields.

First, we presented a bidirectional interleaver design based on transmission gates and tristate buffers. This design allows the same shuffle networks and physical wires to be used for variable-to-check node message passing and check-to-variable node message passing. While not a great success in improving energy efficiency, this design succeeds in improving area efficiency by 28% compared to a design using conventional unidirectional CMOS logic. Furthermore, this pass-transistor based design has potential for use in future research with ulta-low supply voltage circuits (see Section 7.2.1 below).

Next, we presented a PWM-MS (pulse-width modulated min-sum) LDPC decoder. In this design, inter-node message passing occurs in a sign-magnitude pulse-width format. This message format has the advantage of low switching activity, as well as allowing the complex minimum-finding check node computation to be accomplished with a single AND gate. Our ASIC design of this decoder achieved a 19%

energy efficiency improvement over the similar bit-serial min-sum decoder architecture.

For the third project, we investigated and designed decoders based on differential binary (DB) algorithms. Analysis of one of these algorithms, MDD-BMP (modified differential decoding with binary message passing), led to the development of an IDB (improved differential binary) decoding algorithm. Despite having much lower computational complexity, IDB achieves error correction performance similar to the split-row min-sum and stochastic decoding algorithms. Our ASIC design of an IDB decoder achieved very large improvements in silicon area, throughput, and energy efficiency compared to previous decoder designs in the literature.

Finally, we introduced energy efficient algorithms and decoder designs based on the gear-shift concept. In gear-shift decoding, decoders can choose from multiple different node computation rules during decoding. This can be used to reduce energy consumption by first applying simpler decoding algorithms, then switching to more complex ones if they fail. PWM-MS and IDB are naturally amenable to gear-shift decoding, and formed the basis of our GSP (gear-shift pulse-width min-sum) and IGSP (IDB with GSP) algorithms. ASIC designs of GSP achieved moderate increases in area efficiency and slight increases in energy efficiency compared to offset min-sum decoders, without compromising error correction performance. IGSP achieved much larger improvements in throughput and energy efficiency, but at the cost of higher silicon area.

7.2 Possibilities for Future Research

A large amount of research interest continues to be shown in both LDPC codes and energy efficient circuits. This section lists a few of the more promising approaches to future research in these fields.

7.2.1 Ultra-Low Voltage Circuits

As mentioned in Chapter 2 of this dissertation, the dynamic energy consumption of a digital circuit can be greatly reduced by reducing the supply voltage. Recent research has examined the design and operation of circuits with supply voltages near or below the transistor threshold voltage V_t. While circuits operate very slowly at such voltages, they can theoretically have much greater energy efficiency than circuits operating at the supply voltages typically used in CMOS technology. LDPC decoding is naturally suited for these ultra-low voltage circuits, since its inherently parallel nature allows it to achieve high decoding throughput even at low clock frequencies. Some early work with simple LDPC decoders (using syndrome decoding and the Gallager A algorithm) operating at sub-threshold supply voltage can be found in [113].

There are, however, many challenges with such circuits. One of the most important ones is that leakage energy becomes increasingly significant as supply voltage is reduced, and becomes the dominant source of energy consumption below a certain threshold. One proposed solution to this is to use logic blocks based heavily around pass transistors [114] - since pass transistors are unpowered, they have zero leakage. The bidirectional transmission gate interleaver presented in this dissertation could possibly be used in an LDPC decoder based on this technique.

7.2.2 Asynchronous Decoders

Asynchronous decoders are another possible area of interest. Significant amounts of energy are expended in the clock tree of synchronous circuits, as well is in the internal cycling of sequential elements. An asynchronous design would eliminate both, but the issues of excessive handshaking circuitry and lack of design tools for asynchronous circuits present a problem.

Some initial work in asynchronous min-sum and stochastic decoders has been produced in [94] and [115]. Future efforts might focus on asynchronous stochastic decoding, as the probabilistic nature of these decoders might allow handshaking overhead to be considerably reduced (an erroneous or chronically slow bit caused by inadequate handshaking or state transition enforcement would not have a catastrophic effect on the operation of a stochastic decoder).

Asynchronous circuitry has also been proposed to solve the problem of variation in ultra-low voltage circuits [116] [117], so these two approaches have some overlap.

7.2.3 Statistical Computing

One final area of interest is a relatively new field known as statistical (or non-deterministic) computing. Statistical computing is the study of inexact computing methods and circuits, and their applications in tasks that are tolerant towards any errors these methods might introduce [118]. By abandoning determinism, energy consumption can be reduced through reduction of design margins, and the use of probabilistic circuit elements that are smaller and consume less energy than their deterministic counterparts (but only theoretically, as such elements have yet to be demonstrated). Error correction coding is a natural application of this, since a certain

probability of error is expected - much like in a stochastic decoder, a small number of internal errors caused by probabilistic circuitry can be tolerated. LDPC codes, with their high error correction performance, could also be used as a check on other results of statistical computing, assuming the energy savings from probabilistic methods are larger than the overhead incurred from encoding and decoding.

Appendix A: Decoder Design Flow

This section describes the design and ASIC design flows of the LDPC decoders presented in this dissertation.

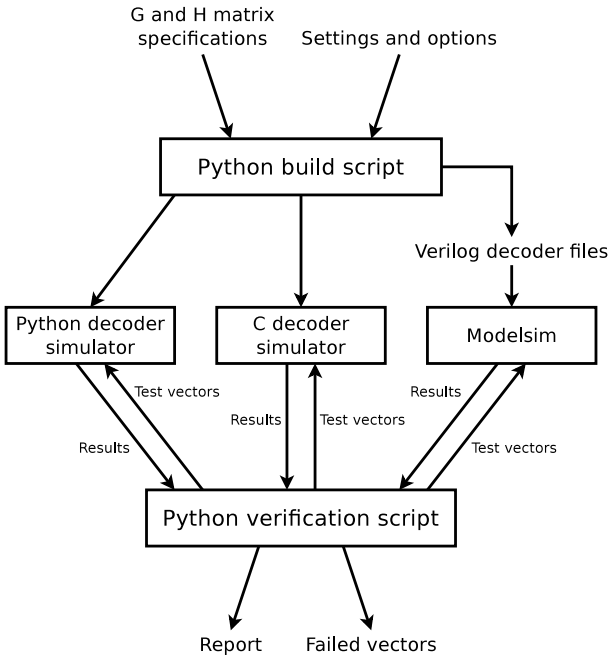


Figure A-1: LDPC decoder design and verification flow.

Figure A-1 shows the design and verification flow. A Python script reads the **H** matrix data for the LDPC code, along with a list of configuration options. The **G** matrix data can also optionally be provided for test vector generation. This script produces as output Verilog HDL for the decoder, as specified by **H** and the configuration. It also produces simulators for verifying the produced HDL: a Python simulator, which is primarily used for debugging, and C simulator, used for deep BER simulations.

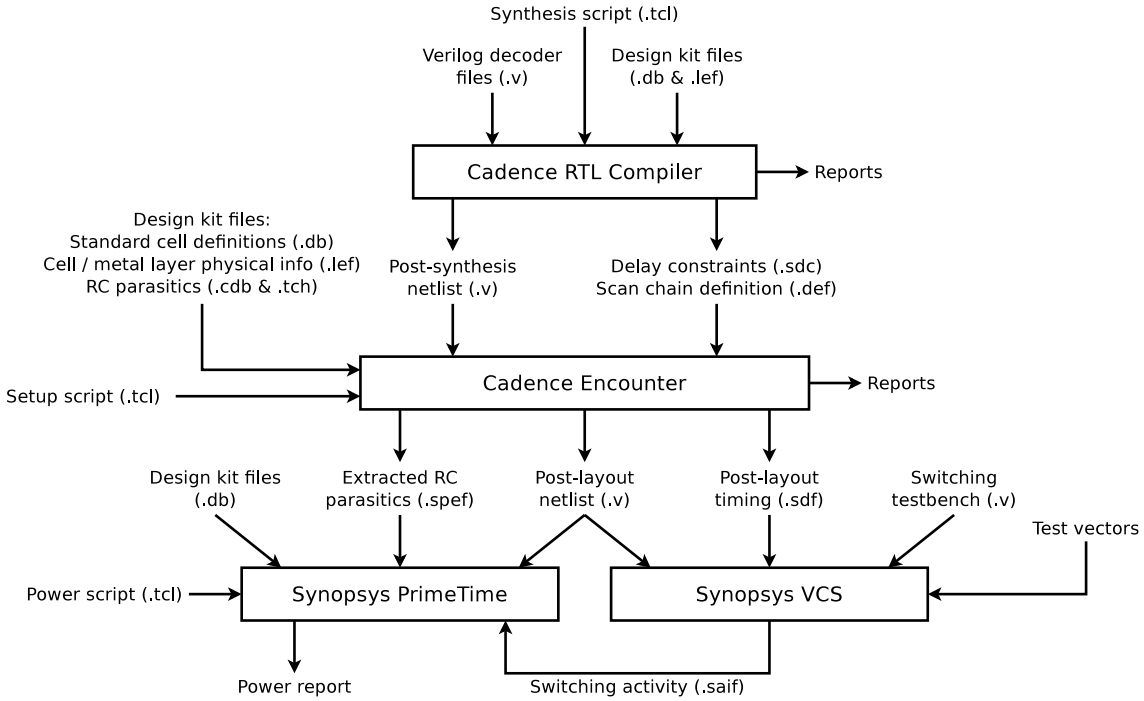


Figure A-2: LDPC decoder ASIC design and power characterization flow.

Another Python script performs automated regression testing and verification between the HDL and the two simulators. This script can produce test vectors for any specified value of E_b/N_0, as well as read fixed vectors from a file. These tests ensure that the HDL is error-free, and that the simulators are bit- and cycle- accurate representations of the HDL.

Once the HDL is generated and verified, the ASIC design flow begins. Figure A-2 provides an overview of this process. The first step is logic synthesis, which is performed using Cadence RTL Compiler Ultra. The synthesis tool is driven by a basic script in Tool Command Language (TCL). The relevant portions of the design kit are also provided as input (these are the standard cell specifications and metal

layer parasitic data).

The next step of the design flow is layout, which is performed using Cadence Encounter. The synthesized netlist and other design specification files produced by RTL Compiler serve as input, along with the relevant design kit info. The silicon area and clock frequency of the decoder is obtained once layout is complete - along with software simulation data, this is sufficient to compute the decoding throughput, as well. The output of this step is the post-layout Verilog netlist, post-layout timing information, and extracted RC parasitics.

To obtain accurate power estimations, the switching activity of all nodes in the circuit must be known. This information is obtained using post-layout simulations with Synopsys VCS. A special Verilog testbench with 200 test vectors per data point is used for these simulations. This step produces as output the switching activity of the circuit in SAIF format.

The final step is to obtain power consumption estimations using the output of the previous steps. The software used to compute the estimations is Synopsys Primetime. This tool takes as input the post-layout netlist, extracted RC parasitics, switching activity log, and the design standard cell specifications. The output is a power report, which contains the power figures reported in this dissertation. Energy efficiency for each decoder, measured as energy per decoded bit, is derived by dividing power by throughput.

List of software tools and design kits used in this flow:

- Cadence RTL Compiler Ultra (versions v9.10-p104_1 and 11.20-s017_1)
- Cadence Encounter (version 09.12-s159_1)

- Cadence IC (version IC.5.1.41)
- Synopsys VCS (version VCS D-2010.06)
- Synopsys Primetime (version D-2010.06-SP2)
- Mentor Graphics Modelsim SE (version 6.3)
- IBM 130nm CMR8SF-RVT Process SAGE-X Standard Cell Library (version 2.0)
- IBM CMRF8SF Process 1.2V/2.5V/3.3V Tolerant Inline GPIO Library (version 1.0)
- TSMC TCBN65GPLUS 65nm Standard Cell Library (version 140B)
- TSMC TPFN65GPGV2OD3 65nm Standard I/O Library (version 200C)
- STMicro CORE90GPSVT 90nm Standard Cell Library (version 2.2)
- STMicro CORX90GPSVT 90nm Standard Cell Library (version 4.3)

References

[1] D. Costello, J. Hagenauer, H. Imai, and S. Wicker, "Applications of error-control coding," *IEEE Trans. Inf. Theory*, vol. 44, no. 6, pp. 2531–2560, 1998.

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

[3] R. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, January 1962.

[4] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. ICC 1993*, vol. 2, May 1993, pp. 1064–1070.

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

[6] D. MacKay and R. Neal, "Near Shannon limit performance of low density parity check codes ," *Electronics Letters*, vol. 32, no. 18, pp. 1645–, 29 Aug 1996.

[7] D. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, Mar 1999.

[8] S.-Y. Chung, J. Forney, G.D., T. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 db of the shannon limit," *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 58–60, 2001.

[9] IEEE Standards Association, "IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1," *IEEE Std 802.16e-2005 and IEEE Std 802.16-2004/Cor 1-2005 (Amendment and Corrigendum to IEEE Std 802.16-2004)*, pp. 1–822, 2006.

[10] IEEE Standards Association, "IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pp. 1–2793, 2012.

[11] IEEE Standards Association, "IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. Part 15.3: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wireless Personal Area Networks (WPANs) Amendment 2: Millimeter-wave-based Alternative Physical Layer Extension," *IEEE Std 802.15.3c-2009 (Amendment to IEEE Std 802.15.3-2003)*, pp. 1–187, 2009.

[12] A. Morello and V. Mignone, "DVB-S2: The second generation standard for satellite broad-band services," *Proc. of the IEEE*, vol. 94, no. 1, pp. 210–227, Jan. 2006.

[13] IEEE Standards Association, "IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications," *IEEE Std 802.3an-2006 (Amendment to IEEE Std 802.3-2005)*, pp. 1–167, 2006.

[14] IEEE Standards Association, "IEEE Standard for Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications," *IEEE Std 1901-2010*, pp. 1–1586, 2010.

[15] International Telecommunication Union. (2004, Feb.) Series G: Transmission Systems And Media, Digital Systems And Networks - Digital sections and digital line system Optical fibre submarine cable systems - Forward error correction for high bit-rate DWDM submarine systems. [Online]. Available: http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-G.975.1-200402-I!!PDF-E&type=items

[16] G. Dong, N. Xie, and T. Zhang, "On the Use of Soft-Decision Error-Correction Codes in NAND Flash Memory," *IEEE Trans. Circuits Syst. I*, vol. 58, no. 2, pp. 429–439, 2011.

[17] A. Blanksby and C. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, Mar 2002.

[18] H. Hamann, A. Weger, J. Lacey, Z. Hu, P. Bose, E. Cohen, and J. Wakil, "Hotspot-Limited Microprocessors: Direct Temperature and Power Distribution Measurements," *IEEE J. Solid-State Circuits*, vol. 42, no. 1, pp. 56–65, 2007.

[19] G. Van der Plas, P. Limaye, I. Loi, A. Mercha, H. Oprins, C. Torregiani, S. Thijs, D. Linten, M. Stucchi, G. Katti, D. Velenis, V. Cherman, B. Vandevelde, V. Simons, I. De Wolf, R. Labie, D. Perry, S. Bronckers, N. Minas, M. Cupac, W. Ruythooren, J. Van Olmen, A. Phommahaxay, M. de Potter de ten Broeck, A. Opdebeeck, M. Rakowski, B. De Wachter, M. Dehan, M. Nelis, R. Agarwal, A. Pullini, F. Angiolini, L. Benini, W. Dehaene, Y. Travaly, E. Beyne, and P. Marchal, "Design Issues and Considerations for Low-Cost 3-D TSV IC Technology," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 293–307, 2011.

[20] S. Kamat, "Thermal management in embedded systems: A software approach," *Potentials, IEEE*, vol. 32, no. 1, pp. 23–26, 2013.

[21] M. Pedram, "Energy-Efficient Datacenters," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 10, pp. 1465–1484, 2012.

[22] K. Rajamani, C. Lefurgy, S. Ghiasi, J. Rubio, H. Hanson, and T. Keller, "Power management solutions for computer systems and datacenters," in *Low Power Electronics and Design (ISLPED), 2008 ACM/IEEE International Symposium on*, 2008, pp. 135–136.

[23] C. Winstead and Y. Luo, "Error correction circuits for bio-implantable electronics," in *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on*, 2012, pp. 158–161.

[24] N. Mobini, A. Banihashemi, and S. Hemati, "A differential binary message-passing LDPC decoder," *IEEE Trans. Commun.*, vol. 57, no. 9, pp. 2518–2523, september 2009.

[25] K. Cushon, W. Gross, and S. Mannor, "Bidirectional interleavers for LDPC decoders using transmission gates," in *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*, 2009, pp. 232–237.

[26] K. Cushon, C. Leroux, S. Hemati, S. Mannor, and W. Gross, "A Min-Sum Iterative Decoder Based on Pulsewidth Message Encoding," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 11, pp. 893 –897, Nov. 2010.

[27] K. Cushon, S. Hemati, C. Leroux, S. Mannor, and W. Gross, "High-Throughput Energy-Efficient LDPC Decoders Using Differential Binary Message Passing," *IEEE Trans. Signal Process.*, vol. 62, no. 3, pp. 619–631, Feb 2014.

[28] K. Cushon, S. Hemati, S. Mannor, and W. J. Gross, "Energy-efficient gear-shift LDPC decoders," in *Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference on*, June 2014, pp. 219–223.

[29] C. Spagnol, E. Popovici, and W. Marnane, "Hardware Implementation of GF(2^m) LDPC Decoders," *IEEE Trans. Circuits Syst. I*, vol. 56, no. 12, pp. 2609–2620, 2009.

[30] G. Sarkis, S. Hemati, S. Mannor, and W. Gross, "Relaxed half-stochastic decoding of LDPC codes over GF(q)," in *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, 2010, pp. 36–41.

[31] X. Zhang and F. Cai, "Reduced-Complexity Decoder Architecture for Non-Binary LDPC Codes," *IEEE Trans. VLSI Syst.*, vol. 19, no. 7, pp. 1229–1238, 2011.

[32] A. Pusane, A. Feltstrom, A. Sridharan, M. Lentmaier, K. Zigangirov, and D. Costello, "Implementation aspects of LDPC convolutional codes," *IEEE Trans. Commun.*, vol. 56, no. 7, pp. 1060–1069, 2008.

[33] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[34] N. Wiberg, *Codes and decoding on general graphs*. Linköping University, 1996.

[35] F. Kschischang and B. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 219–230, Feb 1998.

[36] J. Pearl, "Fusion, propagation, and structuring in belief networks," *Artif. Intell.*, vol. 29, no. 3, pp. 241–288, 1986.

[37] S. Papaharalabos, P. Sweeney, B. Evans, P. Mathiopoulos, G. Albertazzi, A. Vanelli-Coralli, and G. Corazza, "Modified sum-product algorithms for decoding low-density parity-check codes," *Communications, IET*, vol. 1, no. 3, pp. 294–300, 2007.

[38] A. Anastasopoulos, "A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," in *Proc. GLOBECOM 2001*, vol. 2, 2001, pp. 1021–1025.

[39] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb 2001.

[40] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, no. 5, pp. 673–680, May 1999.

[41] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[42] J. Chen, R. Tanner, C. Jones, and Y. Li, "Improved min-sum decoding algorithms for irregular LDPC codes," in *Proc. ISIT 2005*, Sept. 2005, pp. 449–453.

[43] Z. Wang, Z. Cui, and J. Sha, "VLSI Design for Low-Density Parity-Check Code Decoding," *IEEE Circuits Syst. Mag.*, vol. 11, no. 1, pp. 52–69, 2011.

[44] F. Zarkeshvari and A. Banihashemi, "On implementation of min-sum algorithm for decoding low-density parity-check (LDPC) codes," in *Proc. GLOBECOM 2002*, vol. 2, Nov. 2002, pp. 1349–1353 vol.2.

[45] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, "Reduced-Complexity Decoding of LDPC Codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, 2005.

[46] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright, "Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices," *IEEE Trans. Wireless Commun.*, vol. 57, no. 11, pp. 3258–3268, 2009.

[47] C.-H. Chung, Y.-L. Ueng, M.-C. Lu, and M.-C. Lin, "Adaptive quantization for low-density-parity-check decoders," in *Information Theory and its Applications (ISITA), 2010 International Symposium on*, 2010, pp. 13–18.

[48] M. Mansour and N. Shanbhag, "Architecture-aware low-density parity-check codes," in *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, vol. 2, 2003, pp. II–57–II–60 vol.2.

[49] Z. Wang and Z. Cui, "A Memory Efficient Partially Parallel Decoder Architecture for QC-LDPC Codes," in *Signals, Systems and Computers, 2005. Conference Record of the Thirty-Ninth Asilomar Conference on*, 2005, pp. 729–733.

[50] M. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1788–1793, 2004.

[51] E. Yeo, B. Nikolic, and V. Anantharam, "Architectures and implementations of low-density parity check decoding algorithms," in *Circuits and Systems, 2002. MWSCAS-2002. The 2002 45th Midwest Symposium on*, vol. 3, 2002, pp. III–437–III–440 vol.3.

[52] D. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *Signal Processing Systems, 2004. SIPS 2004. IEEE Workshop on*, 2004, pp. 107–112.

[53] J. Zhang and M. Fossorier, "Shuffled iterative decoding," *IEEE Trans. Commun.*, vol. 53, no. 2, pp. 209–213, 2005.

[54] M. Mansour and N. Shanbhag, "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 684–698, 2006.

[55] H.-C. Lee, Y.-L. Ueng, S.-M. Yeh, and W.-Y. Weng, "Two Informed Dynamic Scheduling Strategies for Iterative LDPC Decoders," *IEEE Trans. Commun.*, vol. 61, no. 3, pp. 886–896, 2013.

[56] A. Darabiha, A. Carusone, and F. Kschischang, "Block-interlaced LDPC decoders with reduced interconnect complexity," *IEEE Trans. Circuits Syst. II*, vol. 55, no. 1, pp. 74–78, Jan. 2008.

[57] Z. Cui and Z. Wang, "Efficient Message Passing Architecture for High Throughput LDPC Decoder," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, 2007, pp. 917–920.

[58] A. Darabiha, A. Chan Carusone, and F. Kschischang, "Power reduction techniques for LDPC decoders," *IEEE J. Solid-State Circuits*, vol. 43, no. 8, pp. 1835–1845, Aug. 2008.

[59] T. Brandon, R. Hang, G. Block, et al., "A scalable LDPC decoder ASIC architecture with bit-serial message exchange," *Integr. VLSI J.*, vol. 41, no. 3, pp. 385–398, 2008.

[60] Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "An efficient 10GBASE-T ethernet LDPC decoder design with low error floors," *IEEE J. Solid-State Circuits*, vol. 45, no. 4, pp. 843–855, April 2010.

[61] J. Sha, Z. Wang, M. Gao, and L. Li, "Multi-Gb/s LDPC Code Design and Implementation," *IEEE Trans. VLSI Syst.*, vol. 17, no. 2, pp. 262–268, 2009.

[62] J. Sha, J. Lin, Z. Wang, L. Li, and M. Gao, "Decoder Design for RS-Based LDPC Codes," *IEEE Trans. Circuits Syst. II*, vol. 56, no. 9, pp. 724–728, 2009.

[63] T. Mohsenin and B. Baas, "Split-row: A reduced complexity, high throughput LDPC decoder architecture," in *Proc. ICCD 2006*, Oct. 2006, pp. 320–325.

[64] T. Mohsenin, D. Truong, and B. Baas, "A low-complexity message-passing algorithm for reduced routing congestion in LDPC decoders," *IEEE Trans. Circuits Syst. I*, vol. 57, no. 5, pp. 1048 –1061, May 2010.

[65] N. Miladinovic and M. Fossorier, "Improved bit-flipping decoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 4, pp. 1594 – 1606, april 2005.

[66] P. Zarrinkhat and A. Banihashemi, "Threshold values and convergence properties of majority-based algorithms for decoding regular low-density parity-check codes," *IEEE Trans. Commun.*, vol. 52, no. 12, pp. 2087 – 2097, dec. 2004.

[67] A. Nouh and A. Banihashemi, "Bootstrap decoding of low-density parity-check codes," *IEEE Commun. Lett.*, vol. 6, no. 9, pp. 391 – 393, sep 2002.

[68] M. Jiang, C. Zhao, Z. Shi, and Y. Chen, "An improvement on the modified weighted bit flipping decoding algorithm for ldpc codes," *IEEE Commun. Lett.*, vol. 9, no. 9, pp. 814 – 816, sep 2005.

[69] C. Winstead, V. Gaudet, A. Rapley, and C. Schlegel, "Stochastic iterative decoders," *Proc. ISIT 2005*, pp. 1116–1120, Sept. 2005.

[70] S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "Fully parallel stochastic LDPC decoders," *IEEE Trans. Signal Process.*, vol. 56, no. 11, pp. 5692–5703, Nov. 2008.

[71] S. Sharifi Tehrani, A. Naderi, G.-A. Kamendje, S. Hemati, S. Mannor, and W. Gross, "Majority-Based Tracking Forecast Memories for Stochastic LDPC Decoding," *IEEE Trans. Signal Process.*, vol. 58, no. 9, pp. 4883–4896, Sept. 2010.

[72] F. Leduc-Primeau, A. J. Raymond, P. Giard, K. Cushon, C. Thibeault, and W. J. Gross, "High-Throughput LDPC Decoding Using The RHS Algorithm," in *Design and Architectures for Signal and Image Processing (DASIP), 2012 Conference on*, Oct. 2012, pp. 1–6.

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

[74] C. Winstead, N. Nguyen, V. Gaudet, and C. Schlegel, "Low-voltage CMOS circuits for analog iterative decoders," *IEEE Trans. Circuits Syst. I*, vol. 53, no. 4, pp. 829–841, April 2006.

[75] N. Weste and D. Harris, *CMOS VLSI design: A circuits and systems perspective*. Boston, USA: Pearson Addison-Wesley, 2005.

[76] C. Auth, M. Buehler, A. Cappellani, et al., "45nm High-k+Metal Gate Strain-Enhanced Transistors," *Intel Tech. J.*, vol. 12, no. 2, pp. 77–85, 2008.

[77] X. Wu, F. Wang, and Y. Xie, "Analysis of subthreshold finfet circuits for ultra-low power design," in *SOC Conference, 2006 IEEE International*, 2006, pp. 91–92.

[78] A.-M. Ionescu, L. De Michielis, N. Dagtekin, G. Salvatore, J. Cao, A. Rusu, and S. Bartsch, "Ultra low power: Emerging devices and their benefits for integrated circuits," in *Electron Devices Meeting (IEDM), 2011 IEEE International*, 2011, pp. 16.1.1–16.1.4.

[79] F. Kienle, N. Wehn, and H. Meyr, "On Complexity, Energy- and Implementation-Efficiency of Channel Decoders," *IEEE Trans. Commun.*, vol. 59, no. 12, pp. 3301–3310, December 2011.

[80] V. Gaudet, C. Schlegel, and R. Dodd, "LDPC decoder message formatting based on activity factor minimization using differential density evolution," in *Proc. ITW 2007*, Sept. 2007, pp. 571–576.

[81] 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 Commun. Lett.*, vol. 7, no. 7, pp. 317–319, Jul. 2003.

[82] T. Brack, F. Kienle, and N. Wehn, "Disclosing the LDPC code decoder design space," *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 1, pp. 6 pp.–, 6-10 March 2006.

[83] P. Radosavljevic, A. de Baynast, M. Karkooti, and J. R. Cavallaro, "Multi-Rate High-Throughput LDPC Decoder: Tradeoff Analysis Between Decoding Throughput and Area," *Personal, Indoor and Mobile Radio Communications, 2006 IEEE 17th International Symposium on*, pp. 1–5, Sept. 2006.

[84] J. Lin, Z. Wang, L. Li, J. Sha, and M. Gao, "Efficient Shuffle Network Architecture and Application for WiMAX LDPC Decoders," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 56, no. 3, pp. 215–219, March 2009.

[85] C.-H. Liu, C.-C. Lin, H.-C. Chang, C.-Y. Lee, and Y. Hsua, "Multi-mode message passing switch networks applied for QC-LDPC decoder," May 2008, pp. 752–755.

[86] M. Rovini, G. Gentile, and L. Fanucci, "Multi-size circular shifting networks for decoders of structured LDPC codes," *Electronics Letters*, vol. 43, no. 17, pp. 938–940, August 16 2007.

[87] V. Eisele, B. Hoppe, and O. Kiehl, "Transmission gate delay models for circuit optimization," Mar 1990, pp. 558–562.

[88] STMicroelectronics N.V., *CORE90GPSVT_1.00V 2.2 Standard Cell Library User Manual & Databook*, Geneva, Switzerland, May 2006.

[89] STMicroelectronics N.V., *CORX90GPSVT_1.00V 4.3 Standard Cell Library User Manual & Databook*, Geneva, Switzerland, May 2006.

[90] A. Cevrero, Y. Leblebici, P. Ienne, and A. Burg, "A 5.35 mm2 10GBASE-T Ethernet LDPC decoder chip in 90 nm CMOS," in *Solid State Circuits Conference (A-SSCC), 2010 IEEE Asian*, Nov. 2010, pp. 1–4.

[91] A. Ghazel, E. Boutillon, J.-L. Danger, G. Gulak, and H. Laamari, "Design and performance analysis of a high speed AWGN communication channel emulator," in *Proc. PACRIM 2001*, vol. 2, 2001, pp. 374–377.

[92] N. Onizawa, T. Hanyu, and V. Gaudet, "Design of high-throughput fully parallel LDPC decoders based on wire partitioning," *IEEE Trans. VLSI Syst.*, vol. 18, no. 3, pp. 482–489, March 2010.

[93] ARM Inc., *IBM CMRF8SF Process 1.2V Core, 2.5V I/O, 3.3V-Tolerant General Purpose In-Line (MA metal stack) I/O Library Databook*, Sunnyvale, CA, USA, March 2007.

[94] N. Onizawa, V. Gaudet, and T. Hanyu, "Low-Energy Asynchronous Interleaver for Clockless Fully Parallel LDPC Decoding," *IEEE Trans. Circuits Syst. I*, vol. 58, no. 8, pp. 1933–1943, 2011.

[95] Y. Kou, S. Lin, and M. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2711 –2736, nov 2001.

[96] J. Cho, J. Kim, and W. Sung, "VLSI Implementation of a High-Throughput Soft-Bit-Flipping Decoder for Geometric LDPC Codes," *IEEE Trans. Circuits Syst. I*, vol. 57, no. 5, pp. 1083 –1094, may 2010.

[97] Richardson, T. J., "Error-floors of LDPC codes," in *Proc. 41st Annu. Allerton Conf. Communications, Control and Computing*, Oct. 2003, pp. 1426–1435.

[98] S. Hemati and A. Banihashemi, "Dynamics and performance analysis of analog iterative decoding for low-density parity-check (LDPC) codes," *IEEE Trans. Commun.*, vol. 54, no. 1, pp. 61–70, 2006.

[99] D. MacKay and M. Postol, "Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes," *Electron. Notes in Theor. Comp. Sci.*, vol. 74, pp. 97–104, Oct. 2003.

[100] C. Di, D. Proietti, I. Telatar, T. Richardson, and R. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inf. Theory*, vol. 48, no. 6, pp. 1570–1579, Jun. 2002.

[101] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic, "Analysis of Absorbing Sets and Fully Absorbing Sets of Array-Based LDPC Codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 1, pp. 181–201, Jan. 2010.

[102] C. Schlegel and S. Zhang, "On the Dynamics of the Error Floor Behavior in (Regular) LDPC Codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 7, pp. 3248–3264, July 2010.

[103] X. Chen, J. Kang, S. Lin, and V. Akella, "Hardware Implementation of a Backtracking-Based Reconfigurable Decoder for Lowering the Error Floor of Quasi-Cyclic LDPC Codes," *IEEE Trans. Circuits Syst. I*, vol. 58, no. 12, pp. 2931–2943, Dec. 2011.

[104] F. Leduc-Primeau, S. Hemati, S. Mannor, and W. J. Gross, "Dithered Belief Propagation Decoding," *IEEE Trans. Commun.*, vol. 60, no. 8, pp. 2042–2047, Aug. 2012.

[105] Y. Han and W. Ryan, "Low-floor decoders for LDPC codes," *IEEE Trans. Commun.*, vol. 57, no. 6, pp. 1663–1673, Jun. 2009.

[106] M. Ardakani and F. Kschischang, "Gear-shift decoding," *IEEE Trans. Commun.*, vol. 54, no. 7, pp. 1235–1242, July 2006.

[107] F. Leduc-Primeau, S. Hemati, W. Gross, and S. Mannor, "A Relaxed Half-Stochastic Iterative Decoder for LDPC Codes," in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, Dec. 2009, pp. 1–6.

[108] S. Sweatlock, S. Dolinar, and K. Andrews, "Buffering requirements for variable-iterations LDPC decoders," in *Information Theory and Applications Workshop, 2008*, Feb. 2008, pp. 523–530.

[109] X. Chen, Q. Huang, S. Lin, and V. Akella, "Fpga-based low-complexity high-throughput tri-mode decoder for quasi-cyclic ldpc codes," in *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on*, 2009, pp. 600–606.

[110] I. Tsatsaragkos, N. Kanistras, and V. Paliouras, "Multiple LDPC decoder of very low bit-error rate," in *Digital Signal Processing (DSP), 2011 17th International Conference on*, 2011, pp. 1–6.

[111] T. Mohsenin, H. Shirani-mehr, and B. M. Baas, "LDPC Decoder with an Adaptive Wordwidth Datapath for Energy and BER Co-Optimization," *VLSI Design (Hindawi Journal of)*, vol. 2013, no. 1, pp. 1–14, 2013.

[112] S.-Y. Chung, T. Richardson, and R. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a gaussian approximation," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 657–670, Feb 2001.

[113] C. Winstead and J. Rodrigues, "Ultra-Low-Power Error Correction Circuits: Technology Scaling and Sub-Vt Operation," *IEEE Trans. Circuits Syst. II*, vol. 59, no. 12, pp. 913–917, 2012.

[114] D. Markovic, C. Wang, L. Alarcon, T.-T. Liu, and J. Rabaey, "Ultralow-Power Design in Near-Threshold Region," *Proc. IEEE*, vol. 98, no. 2, pp. 237–252, 2010.

[115] N. Onizawa, V. Gaudet, T. Hanyu, and W. Gross, "Asynchronous Stochastic Decoding of Low-Density Parity-Check Codes," in *Multiple-Valued Logic (ISMVL), 2012 42nd IEEE International Symposium on*, 2012, pp. 92–97.

[116] T.-T. Liu, L. Alarcon, M. Pierson, and J. Rabaey, "Asynchronous Computing in Sense Amplifier-Based Pass Transistor Logic," *IEEE Trans. VLSI Syst.*, vol. 17, no. 7, pp. 883–892, 2009.

[117] B.-H. Gwee, J. S. Chang, Y. Shi, C.-C. Chua, and K.-S. Chong, "A Low-Voltage Micropower Asynchronous Multiplier With Shift-Add Multiplication Approach," *IEEE Trans. Circuits Syst. I*, vol. 56, no. 7, pp. 1349–1359, 2009.

[118] J. Rabaey, H. DeMan, M. Horowitz, T. Sakurai, J. Sun, D. Dobberpuhl, K. Itoh, P. Magarshack, A. Abidi, and H. Eul, "Beyond the horizon: The next 10x reduction in power - Challenges and solutions," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, 2011, pp. 31–31.