Titre: Title:	Activity Reduction and Energy Modeling of Gallager-B LDPC Decoders
Auteur: Author:	Simon Brown
Date:	2023
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Brown, S. (2023). Activity Reduction and Energy Modeling of Gallager-B LDPC Decoders [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/54387/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Activity Reduction and Energy Modeling of Gallager-B LDPC Decoders

SIMON BROWN
Département de génie électrique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie électrique

Juillet 2023

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Activity Reduction and Energy Modeling of Gallager-B LDPC Decoders

présenté par **Simon BROWN**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Christian CARDINAL, président
François LEDUC-PRIMEAU, membre et directeur de recherche
Jean Pierre DAVID, membre

ACKNOWLEDGEMENTS

I would like to acknowledge the Natural Sciences and Engineering Research Council of Canada (NSERC) for their financial support.

I want to thank my supervisor François Leduc-Primeau for his unwavering support and availability and for believing in me since my undergraduate.

I must also thank Jérémy Nadal for always being there when I needed his support.

RÉSUMÉ

La réduction de la consommation d'énergie des décodeurs LDPC est une étape importante pour l'adoption de ces codes pour des applications ayant de fortes contraintes énergétiques telles que la protection de mémoires sur puce. Les décodeurs LDPC du type Gallager-B (GaB) sont bien adaptés aux applications nécessitant un haut débit grâce à leur faible complexité. Nous proposons un algorithme de décodage fonctionnellement identique à GaB qui dispose d'une probabilité de transition des messages réduite que nous nommons LA-GaB. Nous proposons aussi une technique dérivée de l'évolution de densité permettant d'estimer ses probabilités de transition de messages. Nous montrons une implémentation ASIC de LA-GaB dans une configuration déroulée qui réduit l'énergie dynamique de 50% tout en réduisant la surface de fabrication par rapport à l'implémentation équivalente de GaB standard, sans aucun impact sur la performance de décodage. Nous formulons aussi un modèle d'énergie de haut niveau pour cette implémentation déroulée basé sur les probabilités de transition des messages permettant d'explorer efficacement l'impact des divers paramètres de conception sur l'énergie.

ABSTRACT

Reducing the energy consumption of low-density parity-check (LDPC) decoders is an essential step towards increasing the relevance of these codes in energy-constrained applications such as protecting on-chip memories. Gallager-B (GaB) LDPC decoders are well suited for high throughput applications due to their low complexity. We propose the Low-Activity Gallager-B (LA-GaB) decoder, a functional equivalent to GaB with reduced message switching probability, and an adapted density evolution technique capable of predicting its message switching probabilities. We demonstrate an ASIC implementation of LA-GaB in an unrolled configuration achieving a reduction of more than 50% of dynamic energy and lower surface area compared to standard GaB, with no change in decoding performance. We also propose a high-level energy model of this unrolled implementation based on our activity model for rapid design exploration in the energy domain.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

AWGN additive white Gaussian noise. 40

BER bit-error rate. 2, 3, 31

BSC binary symmetric channel. 3, 40

CMOS complementary metal–oxide–semiconductor. 6, 7

CN check node. 4, 10, 13, 16, 18

CNU check node unit. 9, 11, 16–18, 33, 39, 40, 46

DE density evolution. 11, 18, 23, 42, 49

FEC forward error correction. 3

FER frame-error rate. 4

FPFU fully parallel - fully unrolled. 15–18

GaB Gallager-B. 1, 2, 8, 11

HDL hardware description language. 33

LA-GaB low-activity Gallager-B. 1, 2, 16

LDPC low-density parity-check. 1, 4

MS min-sum. 11

NMOS negative-MOS. 7

O-GaB original Gallager-B. 1, 13, 16

OMS offset min-sum. 1, 8, 11, 14, 40, 42

PCM parity-check matrix. 4, 8, 9, 16, 17, 38, 49

PMF probability-mass function. 23

PMOS positive-MOS. 6

QC-LDPC quasi-cyclic ldpc. 9–11

SNR signal to noise ratio. 41

SPA sum-product algorithm. 14

VN variable node. 4, 10, 13, 16, 18, 20

VNU variable node unit. 9, 11, 16–18, 33, 38–40, 46

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Low-density parity-check (LDPC) codes are among today's most commonly used error correction codes. They offer excellent error correction performance approaching the capacity of many channel models. They are included in various communication standards such as 5G, 10GBASE-T Ethernet, and 802.11ax (Wi-Fi 6) [1–3]. LDPC codes are also used in data storage applications such as magnetic hard drives [4] and NAND storage in SSDs [5,6]. Their energy efficiency is becoming a primary design constraint as the throughput and density requirements continuously increase, but power budgets do not, particularly for mobile devices [7].

Two families of LDPC algorithms exist. Soft-decision algorithms such as belief propagation [8], min-sum [9], and offset min-sum (OMS) [10], where internally exchanged messages include probability information, and hard-decision algorithms such as bit-flipping [11] and Gallager-B (GaB) [12], where internal messages are single bits. Hard-decision algorithms cannot achieve the same level of decoding performance, but they benefit from much lower hardware implementation complexity, which translates to lower energy consumption.

This thesis focuses on high-throughput, low-energy applications where hard-decision decoders such as GaB are well suited because of their low complexity. One such scenario is on-chip memories, where error correction can improve yields or increase performance in low-energy regimes.

In the CMOS technology commonly used to implement decoders, energy is consumed whenever a signal changes state as load capacitances are charged or discharged and signal transition times are non-zero, leading to some short-circuit current. We refer to this type of energy consumption as dynamic energy, which constitutes most of a decoder's energy in CMOS technologies [13]. Most LDPC decoder implementations suffer from high switching activity, thus, high dynamic energy. To address this, we propose the low-activity Gallager-B (LA-GaB) algorithm, which is functionally equivalent to the original Gallager-B (O-GaB) algorithm, but reduces the switching activity of messages thanks to a particular change of variable.

Obtaining accurate energy estimation of application-specific circuits is typically done using post-synthesis or post-layout power estimation tools. These tools can provide accurate energy estimations that consider the exact circuit topology, technology-specific implementation details, and approximate or exact information about the circuit layout. However, these tools can only be used after the system has been fully designed, which is a labor and compute-intensive task. Furthermore, to reach accurate results, the power estimation must be fed with

switching activity information obtained through time-consuming simulations of the system. As such, these power estimation methods cannot be used to optimize the design of LDPC codes and decoders. This is of particular concern when exploring design parameters to optimize energy consumption. High-level energy models alleviate this issue, making optimization in the energy domain feasible.

In this thesis, we propose an analytical framework to estimate message probabilities of LA-GaB, from which signal activities can be obtained. Furthermore, we implemented LA-GaB in TSMC 65nm GP technology in an unrolled configuration to validate energy and circuit area gains. Unrolled implementations have been shown to have higher efficiency in the sense of throughput per area than their iterative counterparts [14]. We propose an energy model for this unrolled implementation based on our message probability model, allowing rapid exploration of design parameters.

We will begin in Chapter 2 by reviewing some concepts essential for understanding our work and relevant existing research. Chapter 3 then presents our contributions, including a novel low-activity version of GaB, a probabilistic model for evaluating its signal activities, and a high-level energy model. Chapter 3 also describes the hardware architecture of the ASIC implementations developed to validate the contributions experimentally. Chapter 4 reviews our experimental frameworks for testing bit-error rate (BER), switching activity simulation, and energy measurements. It also presents our results to validate our energy consumption improvements and our activity and energy models.

Some of the material presented in this thesis is part of a paper submitted to the 2023 International Symposium on Topics in Coding [15].

CHAPTER 2 DEFINITIONS AND LITERATURE REVIEW

2.1 Definitions and Basic Concepts

2.1.1 Digital Data Communication

We model digital data communication with forward error correction (FEC) as a data source transmitting binary information vectors b of length K to a channel encoder that encodes b with some redundancy into codewords c of length N preparing them for transmission over a noisy channel, as shown in Fig. 2.1. This channel can be modeled in various ways, but we focus only on the binary symmetric channel (BSC) model of parameter p_e for our work, which will be defined below. Defining the bit-wise XOR operation $\oplus$, the received vector is $r = c \oplus e$, where e is a length N vector of independent and identically distributed random variables following a Bernoulli distribution of parameter p_e. Next, the channel decoder attempts to recover the original information vectors b from r using the included redundancy and outputs an estimate $\hat{b}$ to the user.

Figure 2.1 Block diagram of the basic digital communication model with FEC

The code rate R is the proportion of information bits within a codeword, defined as:

$$R = \frac{K}{N} \tag{2.1}$$

2.1.2 The Binary Symmetric Channel

The BSC is a simple channel model where each transmitted binary bit can be flipped with a crossover probability parameter p_e that fully characterizes the channel. With sent bits $c_i \in \{0, 1\}$ and received bits $r_i \in \{0, 1\}$ the channel transition probabilities are as follows:

$$P(r_i = 1|c_i = 0) = P(r_i = 0|c_i = 1) = p_e, \tag{2.2}$$

$$P(r_i = 1|c_i = 1) = P(r_i = 0|c_i = 0) = 1 - p_e \tag{2.3}$$

BER is the probability that the decoder output decision bit $\hat{b}_i$ is different from the source

bit b_i [16]:

$$P(\hat{b_i} \neq b_i) \ . \tag{2.4}$$

A channel decoder's BER must be considered in relation to the BSC crossover probability p_e to determine its decoding performance. Similarly, frame-error rate (FER) is the probability that one or more bits within a decoder output decision frame of length K contain an error.

2.1.3 Low-Density Parity-Check Codes

LDPC codes are a linear block-code class first invented by Gallager [17]. They are defined by a sparse $M \times N$ parity-check matrix (PCM) denoted $\boldsymbol{H}$ where each "1" defines an interconnection between a check node (CN) and a variable node (VN). Each of the N columns of $\boldsymbol{H}$ represents a single VN, and each of the M rows represents a CN.

Here is an example of a small PCM:

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \ . \tag{2.5}$$

This example is shown for illustration purposes only since the PCM of an LDPC code should, by definition, be sparse, composed mostly of zeros. The $\boldsymbol{H}$ matrix can be represented graphically with a Tanner graph where CNs are represented with squares and VNs as circles [16]. The equivalent Tanner graph representation of the PCM in Equation 2.5 is shown in Figure 2.2 where a single edge is highlighted in both representations showing the same edge, connecting a single CN to a VN.

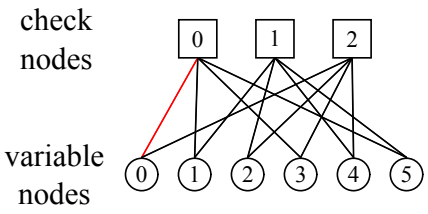

Figure 2.2 Tanner graph representation of $\boldsymbol{H}$

We define $\mathcal{V}_i$ as the set of VN indices connected to CN i and check node degree as $d_c(i) = |\mathcal{V}_i|$. Similarly, we define $\mathcal{C}_j$ as the set of CN indices connected to VN j and variable node degree as $d_v(i) = |\mathcal{C}_j|$. If all CNs have the same d_c and all VNs have the same d_v, then the code is called regular. Otherwise, it is irregular.

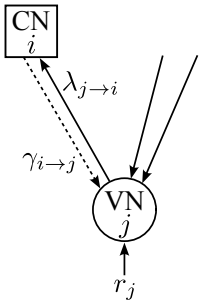

Figure 2.3 The VN update operation where VN j composes message $\lambda_{j\to i}$ using the codeword bit received from the channel r_j and γ-messages received from all of it's neighbours, excluding message $\gamma_{i\to j}$ from CN i

The decoding process starts with initializing each VN j with an estimation of a codeword bit received from the noisy channel r_j, where $j \in N$. Soft-decision algorithms represent channel information estimates as a likelihood of being a zero or a one, quantized to Q bits. Hard-decision algorithms such as GaB store this likelihood information on a single bit with $Q = 1$, resulting in a hard decision. We will refer to this likelihood information as messages which are then exchanged iteratively between VNs and CNs until an iteration limit or a stopping criterion is met. Messages sent from CNs to VNs are noted γ, and messages sent from VNs to CNs are noted λ. Each node performs an update operation defined by a specific message-passing algorithm.

The VN update operation is visualized in Figure 2.3 where VN j receives and sends messages to each CN within $\mathcal{C}_j$. A different message $\lambda_{j\to i}$ is sent to each CN, $i \in \mathcal{C}_j$. An extrinsic principle is followed whereby the message $\gamma_{i\to j}$ is excluded in updating message $\lambda_{j\to i}$. VNs also use the initial channel estimation r_j as an input to the update operation.

Similarly, the CN update operation is visualized in Figure 2.4, following the same extrinsic principle.

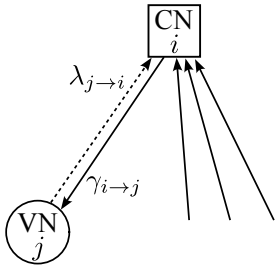

Figure 2.4 The CN update operation where CN i composes message $\gamma_{i\to j}$ using λ-messages received from all of it's neighbors, excluding message $\lambda_{j\to i}$ from VN j

The decoding steps are performed as follows:

1. The VNs are initialized with the channel estimations and broadcast these messages to all CNs to which they are connected since no other information is available at this step.
2. The CNs perform their update operation and send their response following the extrinsic principle shown in Figure 2.4.
3. The VNs perform their update operation and send their response following the extrinsic principle shown in Figure 2.3. The VNs also update their current estimate $\hat{b}$ using a slightly different decision operation that does not exclude extrinsic information and results in a hard decision. The decoding process stops if a stop criterion is met, such as a set number of iterations or a condition on the parity equations.
4. Go back to step 2.

The GaB message passing algorithm will be presented in Section 3.1.1 with its specific node update and decision operations. A numerical example of the GaB decoding algorithm is shown in Annex A.

2.1.4 Digital CMOS Energy Consumtion

In digital complementary metal–oxide–semiconductor (CMOS) technology, energy consumption can be divided into static and dynamic energy. Static energy results from the various leakage currents in the transistors and is typically small compared to dynamic energy. However, the ratio depends on the specific technology node, and there is a trend of increasing static energy in recent nodes [13]. Temperature is known to increase leakage current [18].

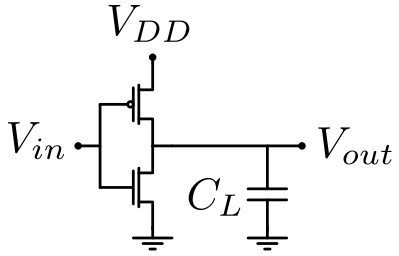

Figure 2.5 CMOS inverter

Figure 2.5 shows a CMOS inverter with a load capacitance C_L at its output, representing the sum of all parasitic and gate capacitances. The inverter is composed of a positive-MOS (PMOS) transistor shown with an inverting input, which allows the flow of current from

V_{DD} to the output when the input V_{in} is low, and a negative-MOS (NMOS) transistor which allows the flow of current from the output to ground when V_{in} is high. The inverter is the simplest CMOS logic function, composed of a single PMOS and NMOS pair, and illustrates how energy is consumed in more complex circuits.

We define dynamic energy as any energy resulting from signal transitions in a CMOS circuit. If we consider signal transition times to be zero, the energy E_{charge} taken from the power supply for the inverter in 2.5 during a full transition cycle is

$$E_{charge} = C_L \cdot V_{DD}^2 \,. \tag{2.6}$$

When V_{in} transitions from high to low, a current flows from V_{DD} threw the PMOS transistor charging the load capacitance C_L. The energy stored in the C_L capacitor equals $C_L V_{DD}^2/2$, meaning only half of the energy supplied by V_{DD} is stored on C_L, with the other half being dissipated by the PMOS transistor. Conversely, when V_{in} transitions from high to low, C_L is discharged to ground threw the NMOS transistor, with the other half of the energy initially supplied by V_{DD} being dissipated in the NMOS transistor.

However, in practice, rise and fall times are not zero, leading to a short period t_{sc} where both PMOS and NMOS transistors conduct current at the same time, which constitutes a direct short-circuit path from V_{DD} to ground. The short circuit energy dissipated in a switching cycle is given by

$$E_{sc} = t_{sc} V_{DD} I_{peak} \,, \tag{2.7}$$

where I_{peak} is the saturation current of the transistors.

Both the energy consumed by the charging of the load capacitance C_L and the short circuit energy are components of what we will refer to as dynamic energy. They are constant for a full switching cycle consisting of a low-to-high and a high-to-low transition. To accurately estimate the power of a digital CMOS circuit performing a given task, we must know the number of switching transitions occurring in one second. The total dynamic power is given by

$$P_{dyn} = \frac{1}{2}(C_L V_{DD}^2 + V_{DD} I_{peak} t_{sc}) f A \,, \tag{2.8}$$

where f is the circuit clock frequency and A is the activity factor, defined as the proportion of clock cycles where a low-to-high or high-to-low transition occurs at the output. The $\frac{1}{2}$ factor is necessary since the A factor counts both high-to-low and low-to-high switching transitions, whereas the previous equations modeled a full switching cycle including both transitions [19].

2.2 Literature Review

2.2.1 Types of LDPC Decoders

LDPC decoders are widely used in digital communication systems due to their decoding performance and the ease with which they can be parallelized. Several LDPC algorithm variants exist that achieve different tradeoffs among decoding complexity, decoding speed, and error-rate performance. These algorithms can be split into two broad categories: hard-decision and soft-decision, as outlined in Section 2.1.3. Soft-decision algorithms such as belief propagation [8], min-sum [9] and OMS [10, 20] can achieve the best error-rate performance but do so at the expense of great decoding complexity. In contrast, hard-decision algorithms have much lower decoding complexity at the expense of some error-rate performance. Hard-decision algorithms are better suited to the high-throughput low-energy problem we study in this thesis since the lower decoding complexity results in higher throughput and lower energy for a given surface area. Several hard-decision algorithms exist. The GaB, which will be presented in detail in Section 3.1.1 was proposed by Gallager [17]. Stochastic decoders [21] have the advantage of being able to represent message probabilities using single bits in the stochastic domain but require complex random generators and suffer from high latency due to the many cycles necessary to represent probabilities accurately, making them unattractive for real-world applications. The original bit-flipping concept was first introduced by Gallager [17], following a similar operating principle as GaB, but without excluding intrinsic messages. This type of decoder has very low complexity but suffers from low error-rate performance. Two non-deterministic algorithms, Gradient descent bit flipping and the probabilistic gradient descent bit flipping algorithms, were introduced in [22], greatly improving the decoding performance. A further improvement was achieved by [20] while improving throughput due to lower decoding complexity. However, the non-deterministic nature of these algorithms and the complex dependencies between messages due to the lack of an extrinsic information rule makes them hard to analyze. A random generator is also necessary for probabilistic decoders, requiring surface area and energy.

2.2.2 Code Flexibility

One of the challenges of decoder hardware implementations is the need for flexibility because communication standards such as WiMAX require support for a wide range of codes with different code lengths, code rates, node degree, and node degree regularity [23]. These problems exist regardless of the channel estimation method and the message-passing algorithm. Various methods are employed in the literature to manage this complexity. In theory, PCMs

built randomly achieve performance closest to the theoretical Shannon limit at the expense of hardware complexity [24]. Structured LDPC codes enable some reduction in hardware complexity as well as added flexibility, sometimes at the cost of coding gain degradation or higher error floors. Quasi-cyclic ldpc (QC-LDPC) codes are a form of structured codes built from a base graph where each element can be replaced with a square diagonal sub-matrix of fixed size z with a shifting parameter or an empty sub-matrix. The hardware nodes only need to support this basic matrix to support any PCM built from it. This does require additional control logic, such as permutation networks. A high degree of flexibility can be achieved using QC-LDPC with variable code rates and information-word lengths [25].

2.2.3 Parallelism methods

LDPC codes lend themselves well to hardware parallelism because nodes at a given iteration are independent, meaning multiple nodes can be processed in parallel. With QC-LDPC decoders, parallelism can be achieved while maintaining flexibility by instantiating up to z Check Node Units (CNUs) along with up to z Variable Node Units (VNUs) for processing of any base graph with that lifting size.

A fully parallel implementation involves mapping the entire Tanner graph to hardware. Fully parallel implementations suffer from poor hardware area utilization ratios as the Tanner graph is mapped to hardware using wires, leading to routing congestion and area utilization as low as 50% [26]. Hard decision decoders suffer less from this congestion issue than soft decision decoders since messages are only a single bit wide.

Parallelism can also be achieved by unrolling decoding iterations using a pipeline, where multiple codewords are processed simultaneously in a cascading manner, with each hardware block performing one decoding iteration. This parallel unrolled architecture reduces routing congestion because messages only travel in a single direction on the unrolled tanner graph, cutting the number of wires needed for a given interconnect in half [14]. Power consumption is also reduced because the lower routing congestion leads to shorter wires with lower capacitance. Very high throughputs are achievable using the unrolled architecture at the expense of some flexibility since the number of decoding iterations is fixed at fabrication time. Fully parallel unrolled implementations can reach the highest throughput of any configuration found in the literature, with throughputs beyond 1Tb/s. A fully unrolled implementation using a finite-alphabet message-passing algorithm in TSMC 90nm CMOS with five unrolled iterations achieves a throughput of 1665 Gbps [27]. A throughput of 1218 Gbps is achieved with a 28nm CMOS 802.11n Min-Sum decoder using five unrolled iterations [28].

2.2.4 Energy Reduction Methods

Increasing the energy efficiency of LDPC decoders is becoming critical as new standards require ever-increasing throughput, but power budgets remain the same [7]. In some applications, the decoding energy is greater than the transmission or storage energy.

The first approach to lowering energy consumption is decreasing the number of decoding iterations. The order in which nodes are updated contributes to the speed at which decoding converges, and different scheduling methods are possible. The most straightforward approach, sometimes called two-step or flooding, involves updating every edge with the CN update operation before updating every edge with the VN update, repeated for however many decoding iterations are necessary. In contrast, layered scheduling, first presented as turbo-decoding in [29], processes a subset of CNs and immediately performs the VN update operation with only a subset of the required edges having been updated [30]. Convergence is reached in only half as many iterations because the following CN subset receives partially updated messages. This scheduling method is used in [31] and [25] with QC-LDPC to improve the performance and flexibility of their respective decoders. Layered scheduling reduces the number of decoding iterations necessary to reach a given level of decoding performance, thus lowering the decoding energy. Dynamic adjustment of decoding iterations, also known as early termination, reduces the energy consumption as only as many iterations as necessary are performed [32]. However, the varying number of iterations results in variable latency, which requires buffers at the input and the output to absorb the latency variation, which reduces the overall energy efficiency. A similar technique was introduced by [7] for unrolled architectures where clock gating is used to reduce the energy consumption of pipeline registers for unneeded decoding iterations while maintaining fixed latency and throughput.

A second approach to lowering energy consumption is reducing message quantization. The complexity, and thus the area and energy cost of nodes, strongly depends on the number of bits on which messages are quantized. The routing congestion of parallel decoders is also affected by quantization. Lowering the quantization of channel estimations and messages comes at the cost of reduced decoding performance. Hard-decision decoders achieve the best throughput density and energy efficiency by quantizing messages on single bits. Bit-serial message transmission for a parallel OMS decoder is proposed in [32], alleviating the routing congestion issue without reducing quantization by transmitting messages on a single wire over several clock cycles. This approach is well suited for OMS decoders as both the minimum and sum operations are inherently bit-serial.

A third approach to lower decoding power is dynamic voltage and frequency scaling. The decoder power consumption has a quadratic relationship with supply voltage leading to con-

siderable power savings at the expense of maximum operating frequency. Used in conjunction with early termination, this method reduces the energy cost of a given decoding iteration when fewer iterations are required, minimizing the impact on latency and throughput. A mechanism to predict the number of iterations is required [33] to control the voltage and frequency. If variable latency and throughput are allowed, voltage and frequency can be adjusted based on the state of an input buffer [34].

A fourth approach is the reduction of signal activity, which will be the focus of this thesis. The switching activity generated by messages transmitted over an interleaver network is of particular interest because these messages' throughput exceeds the channel throughput, and messages transit over long wires due to routing congestion and the random nature of the Tanner graph. A pulse-width encoding for messages is proposed in [35] that reduces switching activity by using pulses of varying widths to represent different quantized message values with a fixed maximum transition count of one.

A probabilistic method called density evolution (DE) is presented in [36] for estimating the error performance of LDPC codes without time-consuming Monte Carlo simulation. The technique consists of propagating message probabilities iteratively across decoding iterations. This DE method is adapted in [37] to estimate signal activity analytically and model decoding energy.

2.2.5 Gallager-B Implementations

Focusing now on GaB decoders, few hardware implementations are documented in the literature. A QC-LDPC GaB decoder with flooded scheduling is implemented by [38] on FPGA. z CNUs and z VNUs are instantiated in hardware, with each unit processing edges in a serial manner. This implementation inherits the flexibility of QC-LDPC decoders, as any base graph with lifting size z can be supported with only the memory requirements changing. Since they use a flooded scheduling method, they chose to process two codewords simultaneously to avoid VNUs going idle when CNUs are busy and vice versa. A throughput of 342 Mbps is achieved with a WiMAX code with $R = 0.5, z = 96$ and a base matrix consisting of 24 columns and 23 rows.

A modified version of GaB is implemented on FPGA by [39], where random noise is added to messages if the decoder fails to converge within a set number of iterations. This noise improves the decoding performance, particularly in the error floor region. They manage to improve the decoding performance of GaB, bringing it closer to that of min-sum (MS) and OMS decoders while using fewer hardware resources. Two codes are implemented, one with $N = 1296, d_v = 4, R = 0.5$ and another with $N = 1296, d_v = 4, R = 0.75$. It is fully parallel,

with iterations being computed sequentially. Throughputs of 29.575 Gbps and 29.471 Gbps are achieved for the reference GaB decoder and the modified GaB decoder.

CHAPTER 3 LOW-ACTIVITY BINARY-MESSAGE LDPC DECODING

3.1 Gallager-B Decoding Algorithm

3.1.1 Original Gallager-B Decoding Algorithm

The O-GaB decoding algorithm for LDPC codes was proposed by Gallager [17]. It is a hard decision algorithm meaning that each symbol received from the communication channel is quantized on a single bit. Additionally, all messages exchanged in the algorithm are also single bits. These binary messages are iteratively exchanged between VNs and CNs following the Tanner graph interconnections and the extrinsic principal, both introduced in Section 2.1.3. Each VN carries the a posteriori estimated codeword bit value. At iteration ℓ, the message sent from CN index i to VN index j is denoted $\gamma_{i\to j}^{(\ell)}$. The message sent form VN j to CN i is denoted $\lambda_{j\to i}^{(\ell)}$.

We define the $\bigoplus$ operator as the multi-input XOR operation equivalent to a modulo-2 summation. For $\ell > 0$, the CN update is expressed as

$$\gamma_{i\to j}^{(\ell)} = \lambda_{j\to i}^{(\ell-1)} \oplus \left(\bigoplus_{j'\in\mathcal{C}_i} \lambda_{j'\to i}^{(\ell-1)} \right). \tag{3.1}$$

At the beginning of decoding, the λ-messages are initialized with $\lambda_{j\to i}^{(0)} = r_j, \forall j$, and $r_j \in \{0,1\}$ the bit of index j from the received channel codeword vector r. In the case of the Gallager-B decoding algorithm, the VN update function is determined by a threshold parameter t_h with $\lceil \frac{|\mathcal{V}|}{2} \rceil \leq t_h \leq |\mathcal{V}|$ and chosen to minimize the error-rate performance of the decoder. It is possible to use different threshold parameters at each decoding iteration and each variable node degree, although the optimal thresholds vary depending on the channel error probability p_e. The λ-messages takes the value $x \in \{0,1\}$ if more than $t_h - 1$ input γ-messages are equal to x. Otherwise, the message takes the a priori estimated codeword bit value r_j. One way to express the λ-messages is

$$\lambda_{j\to i}^{(\ell)} = \begin{cases} \bar{r_j} & \text{if } \Lambda_{j\to i}^{(\ell)} \geq t_h, \\ r_j & \text{otherwise,} \end{cases} \tag{3.2}$$

with

$$\Lambda_{j\to i}^{(\ell)} = \left| \left\{ i' \in \mathcal{V}_j \setminus \{i\} : \gamma_{i'\to j}^{(\ell)} = \bar{r_j} \right\} \right|. \tag{3.3}$$

This representation will be helpful later in understanding how to improve the decoding algorithm.

Finally, the output bits $\hat{b}_j^{(\ell)}$ are decided by slightly modifying (3.2) where the decision threshold t_d is always $t_d = \lceil \frac{|\mathcal{V}|}{2} \rceil$, and where the extrinsic message is not excluded:

$$\hat{b}_j^{(\ell)} = \begin{cases} \bar{r_j} & \text{if } \Lambda_j^{(\ell)} \geq t_d, \\ r_j & \text{otherwise,} \end{cases} \tag{3.4}$$

with

$$\Lambda_j^{(\ell)} = \left| \left\{ i \in \mathcal{V}_j : \gamma_{i \to j}^{\ell} = \bar{r_j} \right\} \right|. \tag{3.5}$$

Gallager decoders are generally suited for highly parallel implementation since the node update functions can be implemented using a few standard logic gates. The routing is simplified as each message corresponds to a 1-bit signal. Although the error-rate performance of Gallager decoders is higher than soft-output decoders such as sum-product algorithm (SPA) or OMS, their low complexity hardware architecture correlates to a lot of energy saving. Consequently, they are good candidates for applications requiring energy-efficient decoders.

3.1.2 Low-Activity Gallager-B

To reduce the switching activity of O-GaB, we propose a variable change that makes messages independent of the transmitted codeword, thus linking message switching probability only to the channel transition probability. We then demonstrate the implications of LA-GaB on an unrolled implementation since this configuration is ideally suited for high-throughput energy-constrained applications. However, it can also benefit any implementation where hardware nodes process independent messages on successive clock cycles.

We can first notice that (3.2) can be rewritten as

$$\lambda_{j \to i}^{(\ell)} = r_j \oplus \overset{\star}{\lambda}{}_{j \to i}^{(\ell)}, \tag{3.6}$$

with $\overset{\star}{\lambda}{}_{j \to i}^{(\ell)} = 1$ if $\Lambda_{j \to i}^{(\ell)} \geq t_h$, otherwise 0. Since, $\lambda_{j \to i}^{(0)} = r_j$, this implies that the initial value of the low-activity $\overset{\star}{\lambda}$-message is $\overset{\star}{\lambda}{}_{j \to i}^{(0)} = 0$. By substituting (3.6) into (3.1), we obtain a new equation for γ-messages given by $\gamma_{i \to j}^{(\ell)} = r_j \oplus \overset{\star}{\gamma}{}_{i \to j}^{(\ell)}$, with

$$\overset{\star}{\gamma}{}_{i \to j}^{(\ell)} = S_i \oplus \left(\bigoplus_{j' \in \mathcal{C}_i \setminus \{j\}} \overset{\star}{\lambda}{}_{j' \to i}^{(\ell-1)} \right), \tag{3.7}$$

where $S_i = \bigoplus_{j \in \mathcal{C}_i} r_j$ corresponds to the i^{th}-bit of the syndrome. Finally, we can substitute the γ-messages in (3.3) by the previously obtained expression to obtain

$$\Lambda_{j \to i}^{(\ell)} = \left| \left\{ i' \in \mathcal{V}_j \setminus \{i\} : \overset{\star}{\gamma}{}_{i' \to j}^{(\ell)} = 1 \right\} \right| = \sum_{i' \in \mathcal{V}_j \setminus \{i\}} \overset{\star}{\gamma}{}_{i' \to j}^{(\ell)} . \tag{3.8}$$

The decoder can then compute the low-activity messages instead of the original ones, since both $\overset{\star}{\gamma}$ and $\overset{\star}{\lambda}$-messages successively depend on each other, and the syndrome S_i is a constant term that is only determined at the start of the decoding process. Note that the syndrome value does not depend on the transmitted codeword. This property is also true for the messages of the first iteration since $\overset{\star}{\gamma}{}_{i \to j}^{(1)} = S_i$ and the $\overset{\star}{\lambda}$-messages only depend on the previously computed $\overset{\star}{\gamma}$-messages. We can deduce that message values are always independent of the transmitted codewords for all iterations through recursion.

Note that $\overset{\star}{\lambda}$-messages effectively represent intermediate a posteriori bit-flipping error estimations. If there are no errors in a given codeword, then the $\overset{\star}{\lambda}$-message will be a zero vector. Messages only depend on the BSC bit-flipping probability p_e, where the low-activity characteristics come from. Practical applications of decoders operate at low error rates where message vectors will be mostly filled with zeroes.

The a posteriori bit-flipping error estimations are applied to the received codeword at the decision stage. The t_d threshold replaces t_h and we obtain

$$\hat{b}_j^{(\ell)} = r_j \oplus \Lambda_j^{(\ell)} \geq t_d \, , \tag{3.9}$$

where $\Lambda_j^{(\ell)} = \left| \left\{ i \in \mathcal{V}_j : \overset{\star}{\gamma}{}_{i \to j}^{(\ell)} = 1 \right\} \right| = \sum_{i \in \mathcal{V}_j} \overset{\star}{\gamma}{}_{i \to j}^{(\ell)}$.
Annex A contains a numerical example of the O-GaB and LA-GaB decoding algorithms based on the small PCM shown in Section 2.1.3.

3.2 Hardware Implementation

3.2.1 Low-Activity Unrolled Implementation

To evaluate the impact of the changes to the LA-GaB algorithm, we will study a hardware implementation of a decoder using that algorithm and compare it to the equivalent decoder using O-GaB. We will first study the node level, which stays relevant for many types of decoder parallelism methods, then we will study a complete fully parallel - fully unrolled (FPFU) implementation.

The CN i from the Tanner graph, functionally described by equations (3.1) for O-GaB

and (3.7) for LA-GaB is mapped to a hardware check node unit (CNU), capable of computing all $\gamma_{i\to j}\, j \in \mathcal{V}_i$ messages concurrently. Similarly, VN j from the Tanner graph, functionally described by equation (3.2) for O-GaB and (3.6) for LA-GaB is mapped to a variable node unit (VNU) capable of computing all $\lambda_{j\to i}\, i \in \mathcal{C}_j$ messages concurrently.

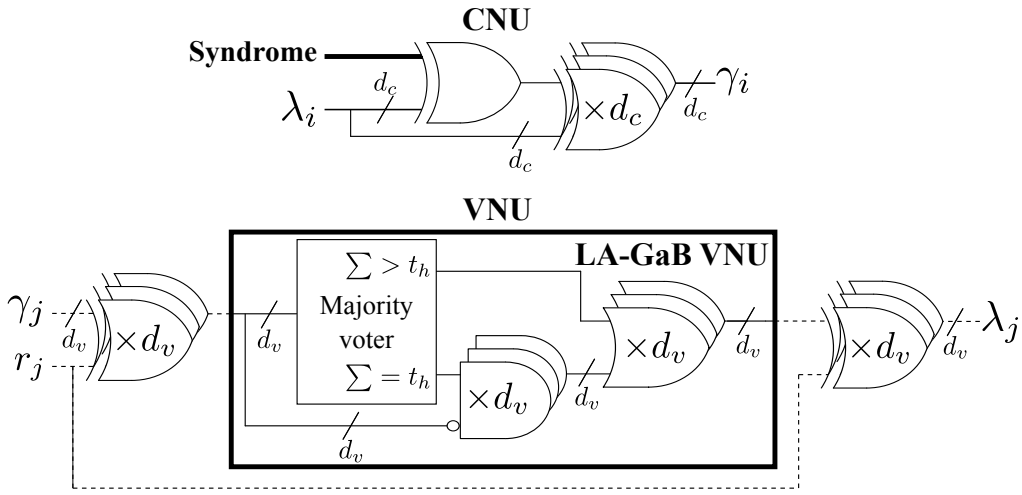

Figure 3.1 CNU and VNU comparison

Figure 3.1 shows the functional differences in CNUs and VNUs between O-GaB and LA-GaB. Compared to O-GaB, circuits added for LA-GaB are highlighted with dark lines, while circuits removed are drawn with dotted lines. The original VNUs can be seen as the low-activity VNUs with an added comparison with the channel bit before and after the extrinsic majority vote operation. Removing these comparisons saves $2 \times d_v$ XOR gates reducing the surface area and energy consumption of the VNU considerably, reducing LA-GaB decoders hardware resource requirements compared to O-GaB as VNUs are replicated many times in an FPFU decoder. Conversely, CNUs are slightly more complex because of the syndrome bit input, but this does not translate to much difference in surface area as there are fewer CNUs, particularly for higher code rates.

We consider the family of FPFU architectures, where the entire computation graph is flattened out by instantiating dedicated hardware components for each node and each iteration. For LDPC decoders, this implies that N VNUs and M CNUs are instantiated in parallel for each iteration. Both the VNUs and CNU inputs and outputs are mapped in the same way CNs and VNs are connected in the corresponding Tanner graph. Such a parallel unrolled architecture is not flexible, designed only to support one PCM and a fixed iteration count $\mathcal{L}$. The processing throughput is 1 codeword per clock cycle, and the clock frequency depends on the number of pipeline stages. Unrolled decoders achieve the maximal level of throughput found in the literature.

In a FPFU architecture, each successive clock cycle processes a new independent codeword. The bit value of each γ and λ message depends on the received bit values r_j, equal to 0 or 1 with equal probability regardless of p_e. As a result, dynamic energy consumption is high in all pipeline registers. Internally, VNUs and CNUs signal activities are partially correlated to p_e as intermediate syndromes are computed in CNUs and estimated codeword flipping decisions are computed in VNUs.

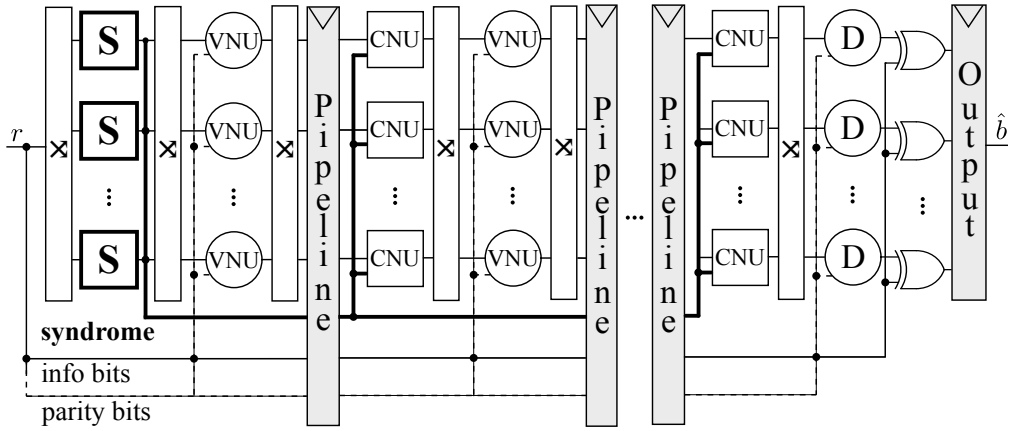

Figure 3.2 High-level Gallager-B FPFU architecture

Figure 3.2 compares the high-level architecture of O-GaB and LA-GaB FPFU decoders. New features introduced with LA-GaB are highlighted, and parts removed from O-GaB are shown with dashed lines. The blocks with arrow symbols are interleavers that implement the PCM interconnections with wires. The first iteration of CNU blocks is replaced with simpler syndrome blocks. The resulting syndrome vector is used in all subsequent iterations as an input to CNUs. Therefore, the syndrome vector is passed along the decoding pipeline. The VNUs no longer use channel bits as inputs. Channel bits are only required in the initial syndrome computation and for the final decision stage. However, only the information part of the codeword is of interest for the final decision. Therefore only K decision nodes are required along with the pipeline registers for the corresponding channel bits rather than the N registers previously required. The size of pipeline registers is equivalent between both versions since the added syndrome bit registers offset the removal of the parity bit registers. It is more advantageous in energy to pipeline syndrome bits rather than received parity bits because the syndrome bits have a much lower switching activity. This advantage is accentuated for lower code rates since more memory is allocated to storing the parity bits.

FPFU architectures for LDPC decoders provide interesting advantages compared to more serial approaches, such as low decoding latency that only depends on the number of pipeline stages. Indeed, for serial architectures, both VNUs and CNUs must be designed to support

the processing of messages having the highest CN/VN degrees, but this case only concerns a subset of the total processed CNs and VNs. Therefore, most of the hardware resources are not fully exploited. Meanwhile, for FPFU architectures, all VNUs and CNUs are specially designed to support their respective degree, so no resources are wasted. In addition, only the VNUs corresponding to information bits are implemented for the last iteration. However, these advantages must be weighed against the fact that the number of iterations must be fixed due to the unrolled nature of the decoder. Therefore, no early termination mechanism can be exploited, unlike serial architectures, so the area and energy efficiency are reduced.

With LA-GaB $\overset{\star}{\gamma}$ and $\overset{\star}{\lambda}$ messages depend only on the transmitted codeword's error probability which has the following benefits:

- Activity only depends on the error rate at a given iteration.
- Adding more iterations is less costly in energy than the typical implementation.
- Analytical signal activity characterization is achievable using modified DE methods.

3.2.2 Study of CNU Post-synthesis Topologies

Given the simplicity of CNU and VNU operations in GaB and the many instantiations of these components, we set out to understand the possible gate-level circuit topologies. They are the basic building blocks of the decoder, along with pipeline registers. Any change in the structure of these components can significantly affect both the surface area and the energy of the decoder, given the large replication count. Post-synthesis topologies can differ considerably from functional diagrams in Fig. 3.1 as modern synthesis tools consider propagation delays, surface area, and other technology-specific parameters. Knowledge of these post-synthesis topologies helps study internal signal activities and possibly design better hardware decoders as different topologies can be compared.

We used the `Genus` synthesis tool from Cadence along with standard cell libraries from TSMC 65nm process technology. Individual nodes were tested inside a wrapper component to provide signal latching and proper configuration representative of what would be generated in the decoder. The clock period constraint supplied to the synthesis tool was found to have a significant impact on the topology of the circuit and the choice of standard cells used. After synthesis and optimization, the resulting netlist was mapped to standard cells specific to the implementation technology. For CNU components, a range of d_c values were tested to identify patterns with the resulting topologies shown in Figures 3.3, 3.4 and 3.5.

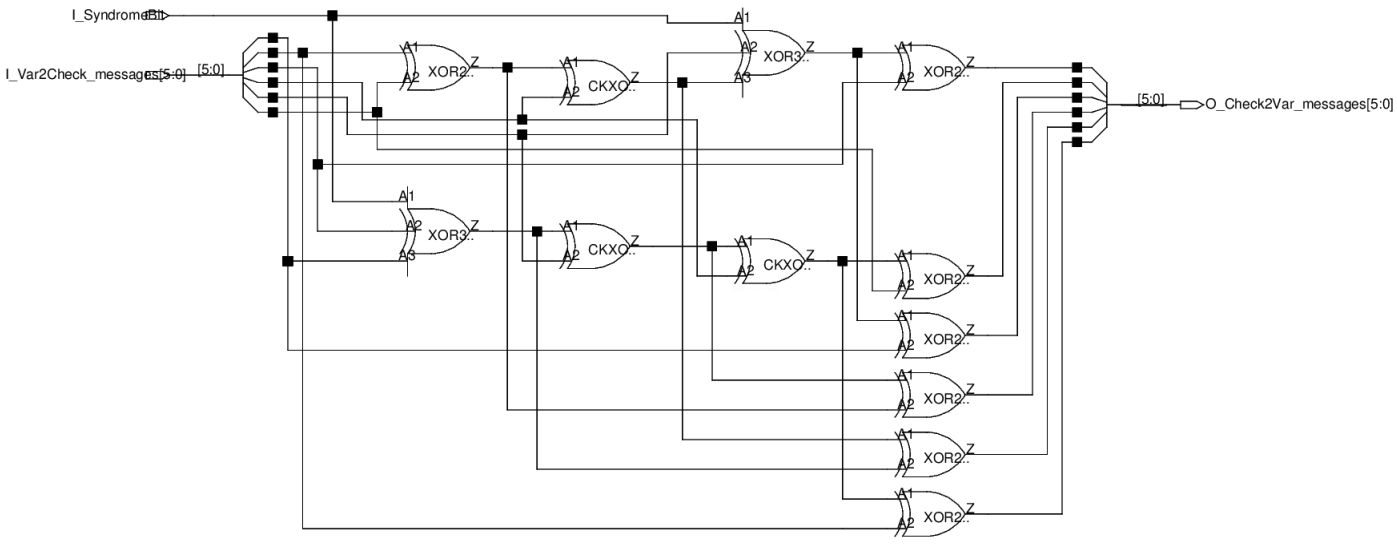

Figure 3.3 `Genus` post-synthesis circuit diagram of low-activity CNU with $d_c = 6$

We found that `Genus` uses different topologies depending on d_c, making the analysis less straightforward. The circuit synthesis rule can be described as follows for $5 \leq d_c \leq 8$:

1. Make a list of the λ input bits and the syndrome bit.
2. Make a reversed copy of the first list.
3. Truncate the last two entries of each list. The syndrome bit must not be truncated.
4. Make a XOR chain from each list, where the first two elements are XORed, then each additional element is XORed to the previous output until all elements are used.
5. The final CNU stage now consists of a XOR of two previously generated signals or inputs.

An example of this topology is shown in Fig. 3.3. We consider three-input XOR gates as two cascaded two-input XOR gates and four-input XOR gates as three cascaded two-input XOR gates. This is a rough approximation since, in practice, standard cells with more inputs have better characteristics than the equivalent circuit composed of smaller standard cells. Table 3.1 shows a typical XOR standard cell's characteristics in TSMC 65nm LP process technology with input count ranging from two to four. Switching power is shown as a function of load capacitance C_{load}, along with width and propagation delay. The four-input XOR gate is less than three times as wide, and its propagation delay is only about twice as long as the two

Table 3.1 TSMC 65nm LP Standard XOR cells comparison [40]

Standard cell	Cell Width (μm)	Average Pin Capacitance (pF)	Average Propagation Delay (ns)	Average Switching Power (pJ)
XOR2D1	2	0.0015	$0.08 + 4.0 \times C_{load}$	$0.005 + 0.0072 \times C_{load}$
XOR3D1	3.4	0.0012	$0.13 + 3.9 \times C_{load}$	$0.007 + 0.0091 \times C_{load}$
XOR4D1	5	0.0014	$0.16 + 4.5 \times C_{load}$	$0.008 + 0.0055 \times C_{load}$

input versions. Transistor-level optimizations possible in the layout of the standard cells account for these discrepancies.

With this topology, there will be $3 \times d_c - 4$ two-input XOR gates and a logic depth of $d_c - 1$. Fanout equals 2 for input and intermediate XOR signals (except the syndrome bit XORs with fanout equals 1) and 1 for the output stage of XORs.

For $9 \leq d_c \leq 15$, the topology is similar to that of smaller d_c values, except there are no two distinct chains of XOR gates as illustrated by Fig. 3.4. The reason for having these chains is to have access to all the necessary intermediate partial XORs needed for the final stage. When an intermediate signal is not required, using a cascaded tree of XOR gates is better because it results in a shorter propagation delay and fewer gates.

Above $d_c = 15$ `Genus` generates a simpler topology shown in Fig. 3.5 where first a global XOR is performed on all inputs, and then at the output stage, each λ input is XORed with the global XOR to form an extrinsic γ output.

This results in $2 \times d_c$ two-input gates and a logic depth of $\log_2(d_c + 1) + 1$. Fanout will be 2 for the input signals since they are used for the global XOR and the output stage. It will be 1 for the output stage as CN output drives either pipeline registers or VNs directly with an input fanout of 1. The XORs involved in the global XOR will also have a fanout of 1, except for the final XOR in the cascade that needs to drive d_c output stage XORs.

The number of gates will be equal between the two topologies at $d_c = 4$, with the second topology resulting in fewer two-input gates starting at $d_c = 5$. The logic depth will be smaller, with the second topology starting at $d_c = 5$. This suggests that the synthesis tool is not choosing the optimal topology as both gate count and gate depth would be improved by choosing the second topology for all $d_c \geq 5$. However, we should consider the fanout in conjunction with the gate count to estimate the surface area, power consumption, and critical path. The high fanout on the final XOR from the global XOR results in the synthesis tool choosing a larger standard cell to handle the higher output capacitance. The dynamic power

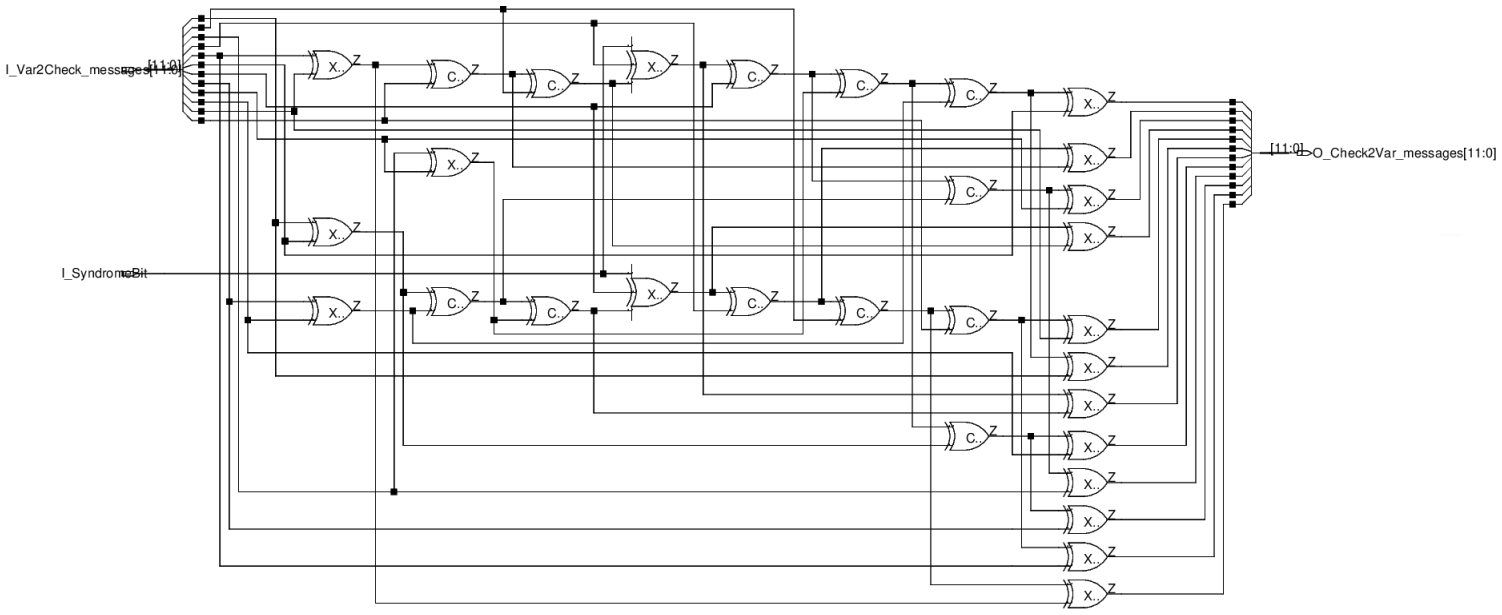

Figure 3.4 `Genus` post-synthesis circuit diagram of low-activity CNU with $d_c = 12$

of that cell is also dependent on its output capacitance and signal activity.

VNs tend to have smaller degrees than CNs. For the decoder configurations we have tested, only degrees 3 and 4 are used. Figure 3.6 shows the topology of a VNU with degree 3 composed of only three logic gates.

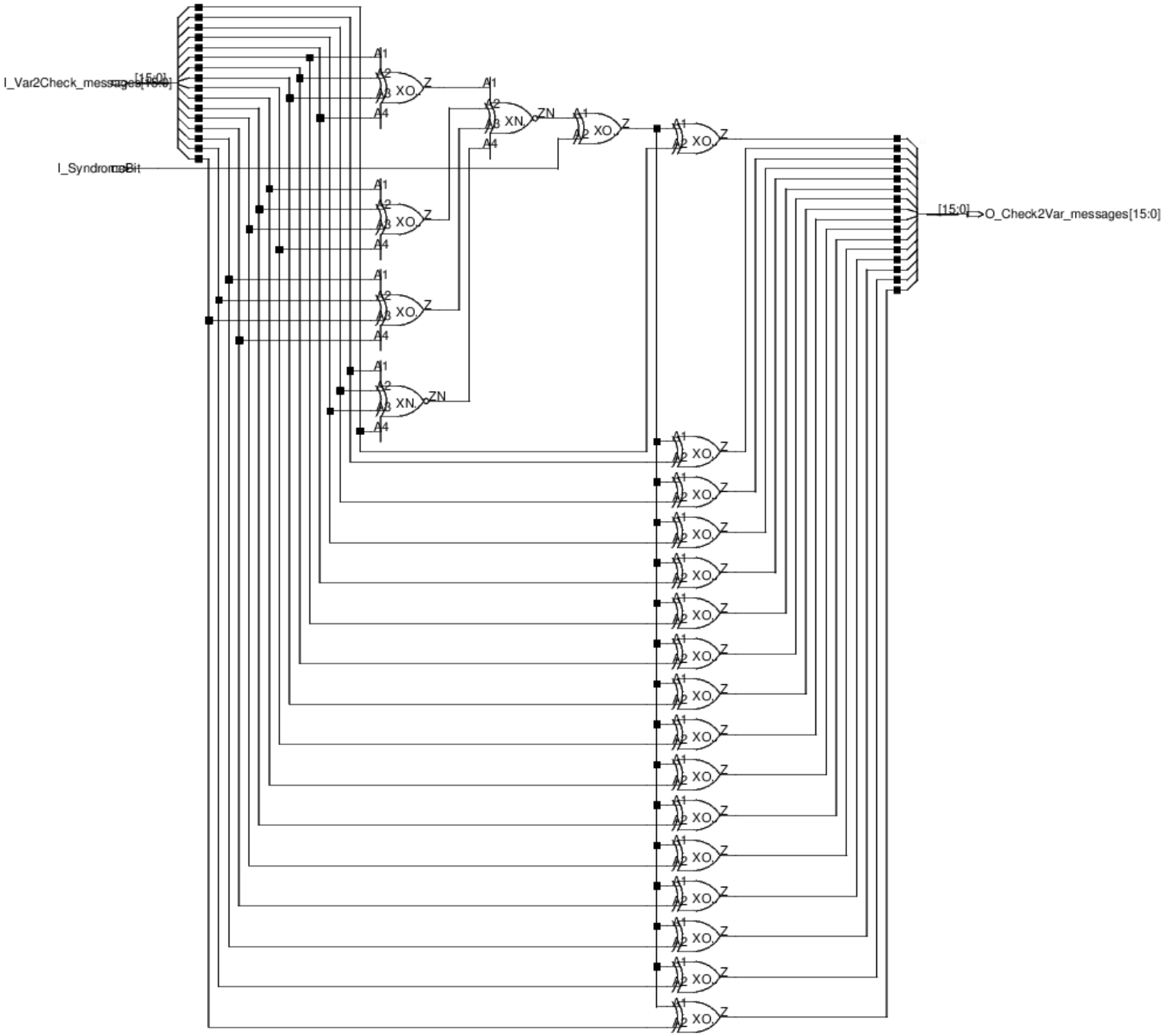

Figure 3.5 `Genus` post-synthesis circuit diagram of low-activity CNU with $d_c = 16$

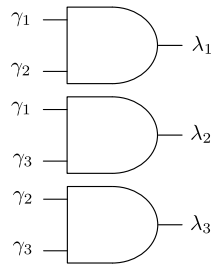

Figure 3.6 Topology of VNU with $d_v = 3$

3.3 Activity and Energy Modeling

We seek to characterize LA-GaB's message probability-mass function (PMF) by adapting O-GaB DE to validate the lowered switching probability independent of implementation architecture. Given a known hardware architecture, these PMFs can be used to calculate signal-switching activity. Based on the message-switching activities, we then construct an energy model for the unrolled architecture presented in Section 3.2.1.

3.3.1 Original Gallager-B Density Evolution

DE methods are generally used to study the asymptotic error-rate performance of codes. The principle is to propagate the probability density of the decoder inputs through its computation graph to deduce how the messages' probability density (or PMF for quantized messages) evolves through each iteration. This analysis relies on certain assumptions that allow considering that messages received at a node are independent, that errors are independent of each other, and that errors are independent of the choice of transmitted codeword [36]. Thanks to these assumptions, the graph computation becomes a tree. In such a case, the mathematical derivation of the message probability densities can be simplified by considering that the all-zero codeword is always transmitted.

To present the DE equations for GaB decoders, we introduce the notation $P_s[x]$, corresponding to the probability that the random variable x equals s. Since messages take binary values, knowing either $P_0[x]$ or $P_1[x]$ is enough to characterize their PMFs fully. The probability that the γ-messages take the value "0" corresponds to the likelihood of having an even number of "1" values in the input messages and can be expressed as

$$P_0\left[\gamma_{i\to j}^{(\ell)}\right] = \frac{1}{2}\left(1 - \prod_{j'\in\mathcal{C}_i\setminus\{j\}}\left(1 - 2P_1\left[\lambda_{j'\to i}^{(\ell-1)}\right]\right)\right). \tag{3.10}$$

For the λ-messages, it is more convenient to first derive the PMF of the Λ variable defined in (3.3). Since all-zero codewords are assumed to be transmitted, Λ can be rewritten as the sum of the input γ-messages. Therefore, the PMF of Λ is obtained by performing successive convolutions on the γ-messages and can be computed in the Fourier domain as

$$P_q\Big[\Lambda_{j\to i}^{(\ell)}\Big] = \sum_{l=0}^{|\mathcal{V}_j|-1} \omega_j^{ql} \prod_{i'\in\mathcal{V}_j\setminus\{i\}} \Big(P_0\Big[\gamma_{i'\to j}^{(\ell)}\Big] + P_1\Big[\gamma_{i'\to j}^{(\ell)}\Big]\omega_j^{-l}\Big), \tag{3.11}$$

with $\omega_j = \exp\Big[2\pi\sqrt{-1}/|\mathcal{V}_j|\Big]$.

The λ-messages take the value "1" when Λ meets or exceeds the threshold t_h. Therefore the λ-message PMF is expressed as

$$P_1\Big[\lambda_{j\to i}^{(\ell)}\Big] = \sum_{q=t_h}^{|\mathcal{V}_j|-1} P_q\Big[\Lambda_{j\to i}^{(\ell)}\Big]. \tag{3.12}$$

3.3.2 Low-Activity Gallager-B Message Activities

We propose to adapt the DE methods to derive the message probabilities of LA-GaB. From (3.6), we have

$$P_1\Big[\overset{\star}{\lambda}_{j\to i}^{(\ell)}\Big] = P_1\Big[r_j \oplus \lambda_{j\to i}^{(\ell)}\Big] \tag{3.13}$$

$$= \frac{1}{2}\sum_{b\in\{0,1\}} \Big(P_{b|\bar{b}}\Big[\Lambda_{j\to i}^{(\ell)} \geq t_h | r_j\Big]\Big), \tag{3.14}$$

where $P_{x|y}[X|Y] = P(X = x|Y = y)$ is the probability that $X = x$ given $Y = y$. We also have $r_j = c_j \oplus e_j$, and variable $\Lambda_{j\to i}^{(\ell)}$ is independent from e_j, the event that received bit j is wrong. Consequently, we have

$$P_{b|\bar{b}}\Big[\Lambda_{j\to i}^{(\ell)} \geq t_h | r_j\Big] = \sum_{b'\in\{0,1\}} P_{b|b'}\Big[\Lambda_{j\to i}^{(\ell)} \geq t_h | c_j\Big] P_{\bar{b}\oplus b'}[e_j], \tag{3.15}$$

with $P_1[e_j] = p_e$, and $\forall b'$. The probability term $P_{1|b'}\Big[\Lambda_{j\to i}^{(\ell)} \geq t_h | c_j\Big]$ can be straightforwardly calculated from (3.12) if the probability term $P_{q|b'}\Big[\Lambda_{j\to i}^{(\ell)} | c_j\Big], \forall q \in [0, |\mathcal{V}_j| - 1]$, is known. If $c_j = 0$, then the γ-messages coming to VN j, $\gamma_{i'\to j}^{\ell}$ $\forall i' \in \mathcal{V}_j \setminus \{i\}$, must equal 0 to satisfy the parity check equation, independently of the transmitted codeword. Otherwise, check node i' received an odd number of erroneous bits. If this event happens for q check nodes, then $\Lambda_{j\to i, c=0}^{(\ell)} = q$. Therefore, we have $P_{q|0}\Big[\Lambda_{j\to i}^{(\ell)} | c_j\Big] = P_q\Big[\Lambda_{j\to i}^{(\ell)}\Big]$ that can be calculated using the original DE equation (3.11). When $c_j = 1$, the messages must equal 1 unless bits of

adjacent VNs are erroneous. Therefore, we have $\Lambda^{(\ell)}_{j\to i,c=0} = |\mathcal{V}_j| - 1 - q$ and $P_{q|1}\Big[\Lambda^{(\ell)}_{j\to i}|c_j\Big] = P_{|\mathcal{V}_j|-1-q}\Big[\Lambda^{(\ell)}_{j\to i}\Big]$.

Following a similar mathematical development, the γ-message probability of the proposed LA-GaB decoder can be expressed as

$$P_1\Big[\overset{\star}{\gamma}{}^{(\ell)}_{i\to j}\Big] = \frac{1}{2}\sum_{(b,b')\in\{0,1\}^2} P_{b|b'}\Big[\gamma^{(\ell)}_{i\to j}|c_j\Big] P_{b\oplus b'}[e_j]. \tag{3.16}$$

By noting that $P_{b|b'}\Big[\gamma^{(\ell)}_{i\to j}|c_j\Big] = P_{b\oplus b'}\Big[\gamma^{(\ell)}_{i\to j}\Big]$, we have

$$P_1\Big[\overset{\star}{\gamma}{}^{(\ell)}_{i\to j}\Big] = (1-p_e)P_0\Big[\gamma^{(\ell)}_{i\to j}\Big] + p_e P_1\Big[\gamma^{(\ell)}_{i\to j}\Big]. \tag{3.17}$$

The probability of a binary syndrome signal S_i being set to "1" is the probability that the sum of received bits at indexes in the set $\mathcal{C}_i$ is odd, which simplifies down to (3.18). It is constant through decoding iterations as it is only calculated once at the first iteration.

$$P_1\Big[S_i\Big] = 1/2 + ((1-2p_e)^{d_c(i)})/2 \tag{3.18}$$

We are interested in deriving the signal activity for the FPFU architecture depicted in Figure 3.2. We introduce the notation $A\Big[s\Big]$ for activity, corresponding to the probability that the binary signal s toggles state between two successive clock cycles. Processed message bits belong to different i.i.d. codewords between successive clock cycles. Therefore, the CNU output bit activity associated with edge $i \to j$ is given by

$$A[\overset{\star}{\gamma}{}^{(\ell)}_{i\to j}] = 2P_0\Big[\overset{\star}{\gamma}{}^{(\ell)}_{i\to j}\Big] P_1\Big[\overset{\star}{\gamma}{}^{(\ell)}_{i\to j}\Big]. \tag{3.19}$$

The VNU output bit activity is obtained by replacing $\overset{\star}{\gamma}{}^{(\ell)}_{i\to j}$ variable by $\overset{\star}{\lambda}{}^{(\ell)}_{j\to i}$ in the above equation. Following this reasoning, the O-GaB message activities always equal $\frac{1}{2}$.

To improve the accuracy of the switching activity estimation, we take into account the variations in channel noise induced by the finite code length by computing the expectation of (3.19) with respect to the observed channel, using the same approach as in [37].

3.3.3 Activity-Based Energy Model

Given the effort required to simulate the energy of a single decoder configuration in hardware, we construct a model for dynamic energy based on post-synthesis activity-aware simulation of underlying components. Such a high-level energy model is helpful for rapidly exploring

design parameters and the tradeoff of decoding performance with energy consumption.

We derive a high-level energy model from the signal activity estimations from Section 3.3.2. The following assumptions are built into the model:

1. The channel error probability p_e is constant between successive codewords.
2. The decoder is operating in a range of channel conditions (p_e range) such that output BER is less than 10^{-3}.
3. The internal signal activity of components is proportional to their output activity. Internal signal activity is not modeled for simplicity.
4. The impact of parasitic signal switching, also known as glitching, that results from unequal propagation delays in CMOS circuitry is negligible.
5. All components of a given type consume the same energy.
6. DE-based activity estimation is representative of real circuit signal activities.
7. The dynamic energy consumed by the first iteration syndrome calculation and last iteration decision is identical to ordinary CNUs and VNUs from other iterations, respectively, adjusted for output activity.
8. Routing considerations such as wire length do not impact circuit dynamic energy consumption.

Assumption 2 is necessary because the dynamic energy of CNU and VNU components stops being linear with output activity outside of this range. Each output bit depends on multiple input bits, so the output activity becomes a multiple of input activity and reaches maximal activity of 0.5 long before input activity reaches 0.5. This phenomenon is accentuated with higher node degrees.

Assumption 7 is present to keep the number of model parameters low for simplicity, and we will show that a good git is achieved regardless. The actual energy consumption of the syndrome calculation is quite different than a regular CNU in that its input activity is always 0.5 since it receives codewords from the channel. It also has a different architecture than a regular CNU since it does not apply the extrinsic principle, making it significantly simpler. The last iteration decision nodes also differ from regular VNUs since they have different thresholds and do not exclude intrinsic information. This assumption is less impactful for decoders with larger decoding iteration counts, as the first and last iterations represent a smaller proportion of overall iterations.

Assumption 8 becomes less accurate for larger codes as the routing complexity increases, leading to longer wire lengths, greater capacitance, and more energy required for state changes. However, routing complexity is hard to model accurately, requiring knowledge of technology-specific parameters such as metal-layer count, and would not be appropriate in a high-level energy model.

We will start by defining the energy of each major decoder sub-component for a single unrolled decoding iteration ℓ. We will then sum all the component energies according to the decoder's iteration count $\mathcal{L}$.

Beginning with the CNU component, its output is characterized with the random variable $\overset{\star}{\gamma}$, with activity at iteration ℓ described as $A\left[\overset{\star}{\gamma}^{(\ell)}\right]$ according to equation (3.19). Since the decoder architecture is fully parallel, all M CNs in the Tanner graph are mapped to a hardware CNU instance, so the total energy of CNU components at a given iteration is the sum of each CNU component's energy. Considering the possibility of irregular codes, each CN of index m in the set of M can have a different degree $d_c(m)$. Our energy model is a linear fit of output activity with $\alpha_{\text{CNU}}^{(d_c)}$ the energy coefficient for a single check node of degree d_c specified in Table 4.3. The total energy of the CNU components of decoding iteration ℓ is given by

$$E_{\text{CNU}}^{(\ell)} = A\left[\overset{\star}{\gamma}^{(\ell)}\right] \times \sum_{m=1}^{M} \alpha_{\text{CNU}}^{(d_c(m))} \,. \tag{3.20}$$

Using a similar reasoning to CNU components, the output of VNU components is characterized by the random variable $\overset{\star}{\lambda}$, with activity at iteration ℓ described as $A\left[\overset{\star}{\lambda}^{(\ell)}\right]$. The total energy VNU components contribute in a given decoding iteration is the summation of each VNU of index n in the set of N VNs in the Tanner graph and of degree $d_v(n)$. Our energy model is a linear fit of output activity with $\alpha_{\text{VNU}}^{(d_v)}$ the energy coefficient for a single variable node of degree d_v specified in Table 4.3. The total energy of the VNU components of decoding iteration ℓ is given by

$$E_{\text{VNU}}^{(\ell)} = A\left[\overset{\star}{\lambda}^{(\ell)}\right] \times \sum_{n=1}^{N} \alpha_{\text{VNU}}^{(d_v(n))} \,. \tag{3.21}$$

The last major component in our decoder architecture is the pipeline register. A pipeline register is placed at the end of each decoding iteration after the VNUs, as illustrated in the block diagram in Figure 3.2. Three signal types get pipelined, each having different switching activity characteristics. However, they all use the same D flip-flop standard cells with the same energy characteristics. The energy model for these components is a linear fit to output

activity with coefficients α_{REG} and β_{REG} that are specified in Table 4.3. This linear fit is for a single-bit D flip-flop component. The constant offset β_{REG} is necessary because the flip-flops have some constant dynamic energy that results from clock activity rather than output activity.

Each of the three pipelined signal types has a different activity. Therefore they are modeled separately. First of all, the energy of $\overset{\star}{\lambda}$-messages pipeline registers with activity $A\Big[\overset{\star}{\lambda}^{(\ell)}\Big]$ at iteration ℓ is described by

$$E^{(\ell)}_{\overset{\star}{\lambda}\text{REG}} = W \times \left(\alpha_{\text{REG}} \times A\Big[\overset{\star}{\lambda}^{(\ell)}\Big] + \beta_{\text{REG}}\right), \tag{3.22}$$

with W the Tanner graph edge count defined as $W = N \times \bar{d}_V = M \times \bar{d}_C$, where $\bar{d}_V$ and $\bar{d}_C$ correspond to the average variable and check node degree respectively. The edge count is the bit-width required to store all $\overset{\star}{\lambda}$ messages from one iteration, hence why it is a factor in equation (3.22) since the fit parameters only account for the storage of a single bit.

Next, codeword information bits are passed along the pipeline. These codewords are independent and randomly distributed at successive clock cycles; their activity is 0.5. There are K information bits in a codeword. Therefore the information bit pipeline register for a single decoding iteration comprises K single-bit flip-flop components. The energy for this single iteration information bit pipeline register is expressed as

$$E_{\text{bREG}} = K \times \left(\alpha_{\text{REG}} \times 0.5 + \beta_{\text{REG}}\right). \tag{3.23}$$

Finally, syndrome bits are passed along the pipeline. They have constant activity $A\Big[S\Big]$ throughout decoding iterations, described in equation (3.18). There are M syndrome bits, therefore the syndrome pipeline register for a single decoding iteration comprises M individual flip-flop components. The energy for a single iteration syndrome bit pipeline register is expressed as

$$E_{\text{SREG}} = M \times \left(\alpha_{\text{REG}} \times A\Big[S\Big] + \beta_{\text{REG}}\right). \tag{3.24}$$

To make a top-level energy model from these single-iteration component models, we must know what components are necessary for each iteration. CNU and VNU components are present at all iterations in our model since we consider syndrome nodes and decision nodes identical to CNUs and VNUs. Therefore $E^{(\ell)}_{\text{CNU}}$ and $E^{(\ell)}_{\text{VNU}}$ are summed over all $\mathcal{L}$ iterations. The $\overset{\star}{\lambda}$-message pipeline registers are necessary at the end of each decoding iteration, except for the last one since decision nodes do not produce $\overset{\star}{\lambda}$-messages. The information-bit pipeline

registers are required for all $\mathcal{L}$ iterations, including the final decoding iteration, because the decoded information bit outputs are latched. The syndrome pipeline registers are unnecessary in the final decision iteration since syndrome information is only required for CNUs, so $\mathcal{L}-1$ iterations are counted. The model could easily be adjusted to different placements of pipeline registers. The total energy for the decoder configured with $\mathcal{L}$ decoding iterations is

$$E_{\text{DEC}} = \sum_{\ell=1}^{\mathcal{L}} [E_{\text{VNU}}^{(\ell)} + E_{\text{CNU}}^{(\ell)}] + \sum_{\ell=1}^{\mathcal{L}-1} E_{\lambda_{\text{REG}}^{\star}}^{(\ell)} + \mathcal{L} \times E_{\text{bREG}} + (\mathcal{L}-1) \times E_{\text{SREG}} \,. \tag{3.25}$$

CHAPTER 4 RESULTS

This chapter presents our experimental methodology and results, validating the contributions presented in Chapter 3. Both software and ASIC test environments are discussed. Experimental energy results are compared to other decoders from the literature. Finally, our energy model is validated by comparing modeled energy to simulation estimates.

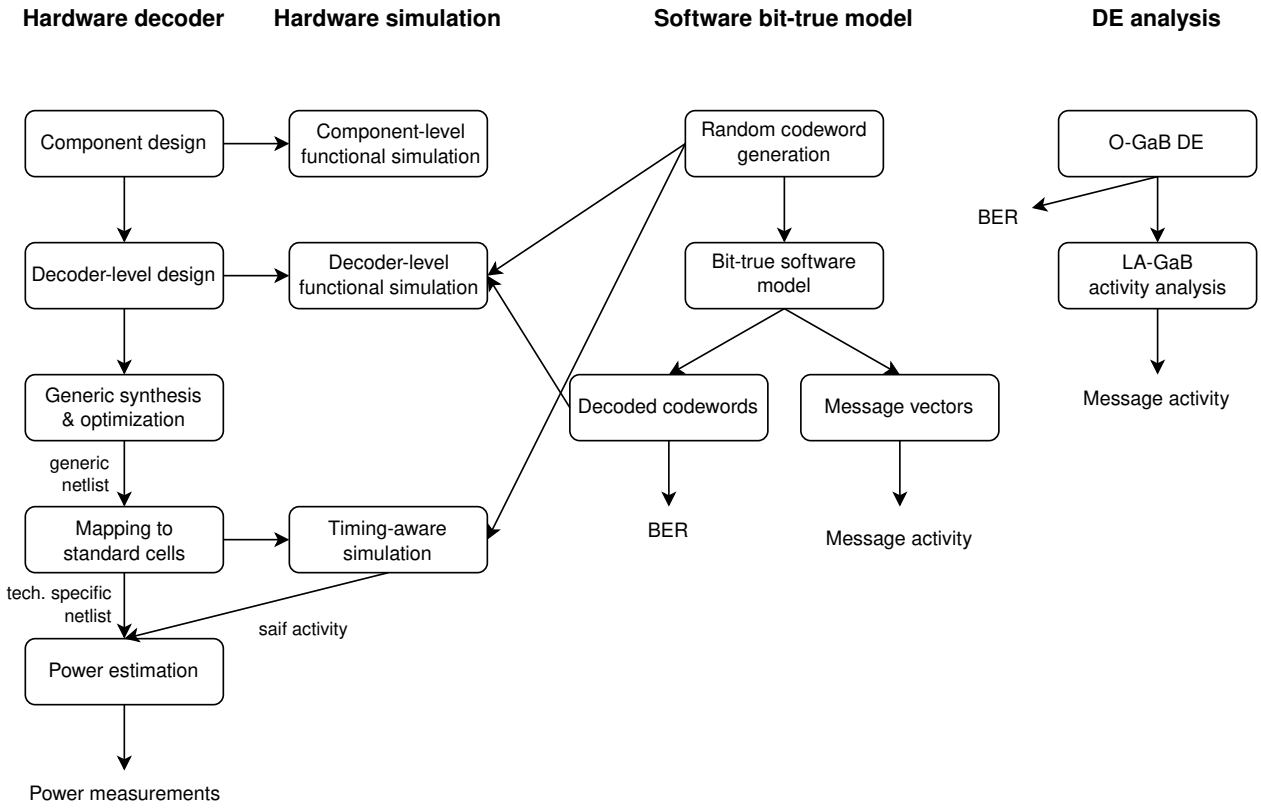

Figure 4.1 Experimental setup flowchart

4.1 Test Environment

4.1.1 BER and Switching Activity Simulation

Bit-true software models of both the original and low-activity Gallager-B message-passing algorithms were implemented in `Matlab`. These models have multiple usages, from comparing decoding performance and evaluating signal activity to generating test vectors to use in the hardware implementation for validation or power measurements. Figure 4.1 gives an overview of the experimental setup.

Binary information vectors b of size K are generated randomly with $P_1(b) = P_0(b) = 0.5$. These information vectors are then multiplied with G, the code's generator matrix, to form valid codewords c of length N. From there, a binary noise vector e of size N is generated with $P_1(e) = p_e$. Each element of e equal to "1" represents a channel bit flipping error. This noise vector is XORed onto the valid codeword to form the input r of the decoder. This step is labeled "Random codeword generation" in the experimental setup flowchart.

Codewords are organized into a matrix of dimensions $N \times batch\_size$ to allow for efficient computation because internal routines within `Matlab` are optimized to work on large arrays. The software model was designed with parallelism in mind using `Matlab`'s `parfor` parallel for-loop to process multiple batches of test vectors simultaneously. Computational efficiency is essential as Monte-Carlo simulation requires a very large number of test vectors to test small BER values accurately. As a rule of thumb, the number of samples should be on the order of $N_{samples} = \frac{10}{BER}$ [41]. Monte-Carlo simulation provides accurate estimates of both BER and signal activity but comes at the cost of high computational effort.

The CN update operation is performed by first calculating a global M-input XOR operation over all CN inputs, with the addition of the syndrome bit in the case of LA-GaB. Then each input is XORed separately with the global XOR to form each CN output. This step implements equations (3.1) for O-GaB and equation (3.7) for LA-GaB.

The VN update operation is performed by summing up all VN inputs and comparing the sum with the threshold parameter t_h. All outputs are set to "1" if the sum exceeds the threshold. If the sum equals the threshold, then each output is set according to the corresponding input, where the output is only set to "1" if that input was "0" because it did not contribute to the sum. If the threshold is unmet, all outputs are set to "0". Additional steps are required for O-GaB: all inputs are XORed with the channel bit prior to the summation and XORed again with the channel bit after the threshold comparison. These steps implement equation (3.2) for O-GaB and equation (3.6) for LA-GaB.

All intermediate messages are stored in a matrix of dimension $W \times batch\_size$ where W is the edge count in the Tanner graph, with functions to retrieve the set of relevant indexes for each node according to the parity check matrix. A copy of the matrix is kept for each decoding stage for future analysis.

In the first iteration of LA-GaB, the CN update operation is simplified; all outputs of a given CN output the global XOR of inputs. Messages are initialized with the channel codewords at the first iteration. The CN and VN update operations are repeated on the message matrix until the iteration count is reached.

The final VN update stage is modified using the t_d decision threshold parameter and outputting a single summation comparison bit. This decision bit is XORed with the channel bit for both O-GaB and LA-GaB versions and serves as the output of the decoder. This implements equation (3.4) for O-GaB and equation (3.9) for LA-GaB.

BER is evaluated by XORing the decoder output with the original noise-free information vector. Any flipped bit in the vector will be set to "1", so the BER is the total proportion of bits set to "1" in the vector.

Because this software model implements the same binary operations as the hardware implementation, the message vectors are bit-true. In an FPFU architecture, each copy of the message matrix from the different decoding stages corresponds to a different set of signals in the hardware decoder. Each row of a given message matrix is the state of that set of signals at a point in time. Each clock cycle processes a distinct codeword or row in the matrix. Therefore, signal activities can be derived from the message matrices at each stage in the decoder. Activity is considered to be the probability that a bit changes state from $(0 \rightarrow 1)$ or $(1 \rightarrow 0)$ at two successive points in time or rows in the matrix. It is evaluated by performing a XOR operation on each pair of consecutive message rows and counting the proportion of bits set to "1". The choice of successive message vectors to compare is architecture dependent and could be adapted to different parallelism models.

4.1.2 Energy Measurement Methodology

The `Genus` tool from Cadence can estimate the power consumption of a synthesized circuit based on the signal activities obtained from simulation and standard cell leakage and transition energy information supplied with the standard cell library of a given technology. These estimations are less accurate than those performed after the final circuit layout, but they do take into account many critical aspects of the circuit design. Topology, gate sizing, switching activity, and even approximate wire loads and delays are accounted for. After the layout stage, wire loads, circuit timing, and cell sizing are calculated more accurately, leading to improved energy estimations. The layout process is very time-consuming and depends on the exact circuit configuration, so it is not practical to include it in the parameter exploration stage of a system design.

The design and test steps of the hardware implementation are described in the experimental setup flowchart in Fig. 4.1. Functional simulations were performed at the component level on components such as the CNU, VNU, syndrome calculation, and decision nodes. For these simple circuits, testbenches were designed to iterate over all possible combinations of inputs and validate the circuit output by comparing them to expected outputs that are

either generated manually or calculated using a different method than the actual hardware implementation. This ensures the components function as intended. The exhaustive iteration is only feasible with components with a small number of inputs n as the total number of possible input combinations is 2^n.

Next, the individual components form a functional decoder using a parity check matrix for configuration and parameters such as the iteration count $\mathcal{L}$, GaB threshold t_h, and pipeline register placement. Since the implementation is fully parallel, each CN and VN is mapped to a CNU or VNU component and the PCM is used to route interconnections with wires. And as the implementation is unrolled, the mapped CNU and VNU components that form a single decoding iteration are replicated $\mathcal{L}$ times, with the first and last iterations adapted to account for syndrome calculation and decision nodes. This architecture is illustrated in Figure 3.2. Functional simulations were designed to validate the hardware description language (HDL) design. Test vectors were generated using the software bit-true model for random codewords with errors and decoder outputs. The same configuration parameters are used in both the software and hardware implementations. The testbenches compare the outputs of the hardware decoder with the expected output from the bit-true model and automatically report any discrepancy. An exhaustive simulation over all possible input combinations is not feasible because of the large set of possible combinations. Instead, several thousand randomly generated codewords with added noise are used to gain confidence in the design. Any errors in the software bit-true model or the hardware implementation were corrected at this stage before proceeding to synthesis.

`Genus` is used to synthesize the HDL code, optimize the logic and map the netlist to a library of standard cells. The cells are sized in a way that respects the fanout necessary for each signal and respects a clock period constraint while also minimizing surface area. The tools make a basic circuit layout to estimate area usage and wire loads at this stage. This results in a netlist file describing the elementary standard cells used and every interconnection.

More simulations are run using this post-synthesis netlist. Now that the actual standard cells are known, timing libraries for those standard cells are used in conjunction with the netlist, which enables timing-aware simulations that consider each standard cell's propagation delay properties. These simulations verify that the decoder has no timing violations and allow the recording of signal activities, including state changes that do not align with clock cycles resulting from propagation delay mismatches known as glitches. Noisy codeword test vectors with a known error probability from the software model are used again for the circuit simulations in the `modelsim` circuit simulation tool from Mentor Graphics. A large batch of these test vectors is generated for each p_e simulation point to obtain a meaningful average of

signal activity.

These timing-aware circuit simulations produce an activity report file where the total state transition counts are recorded for each signal in the circuit. It is important to activate power measurement during the exact time frame that the circuit under test is active and performing the operation of interest. For example, it was important to eliminate any period required to initialize the decoder and the time after the decoder stops receiving codewords not to skew the results. There is also a period where the pipeline fills at the start of decoding and a period of emptying at the end. We consider these periods with partial energy consumption negligible as they cover only a few clock cycles ($2 \times (pipeline\_depth - 1)$) compared to a large number of decoding clock cycles.

`Genus` is used again to estimate the dynamic and leakage power consumption of the circuit given as input the post-synthesis netlist, the simulation activity report, and the standard cell library database. Dynamic and static power was explained in 2.1.4. Reports are generated and then analyzed automatically using `Matlab` scripts to compile the data from various simulations and perform analysis.

4.2 Synthesis Constraint Optimization

There is a large design space of possible FPFU GaB decoder configurations. Code size and code rate directly impact throughput and surface area. The number of decoding iterations affects decoding performance and surface area. Once the code, iteration count, and throughput requirement are chosen for the intended application, pipeline registers can be placed to reduce the critical path to meet the throughput requirement. We have limited the scope of our testing to three different codes, a fixed iteration count $\mathcal{L} = 10$, and a fixed pipeline register configuration, shown in the block diagram in Figure 3.2 for all testing. The codes chosen will be presented in Section 4.3.1.

Given a fixed configuration, different throughput and area results can be obtained by modifying synthesis parameters. We have studied the effect of two major synthesis parameters and will present the results in the following sections.

4.2.1 Design Hierarchy Flattening

The decoder HDL design is fragmented into components replicated according to the desired configuration forming a design hierarchy. This design hierarchy can be flattened to help the synthesis tool optimize the design or specific modules can be specified to be left intact while the rest of the design is flattened. Any elements that are flattened disappear from

all subsequent area and power reports. Several module flattening arrangement combinations were tested as shown in table 4.1, and surprisingly preserving the design hierarchy yielded a better mix of area, timing, and energy consumption results. This also helped at the analysis stage since greater granularity was available in the power reports.

Table 4.1 shows several design hierarchy flattening configurations' synthesis results and subsequent power simulations. This study used the $area_{min} + 10\%$ method of clock period search described below, which could skew results. We deduct from these results that the best energy performance is achieved by preserving most of the decoder components while maintaining near-optimal throughput and with only a small penalty in surface area (4.26%) compared to the most compact implementation. The retained configuration is highlighted in gray.

4.2.2 Clock Period Constraint

When performing a timing-aware synthesis of a logic design, a period constraint needs to be provided along with timing information for all standard cells to be used. The synthesis tool will then select the standard cells with the timing characteristics necessary to meet the specified constraint while minimizing the surface area. This constraint can significantly impact the circuit's surface area, throughput, and energy consumption. Our first approach was to search for the synthesis period constraint that would render a circuit with an area equal to the minimal area possible plus 10% with the intent of obtaining a good tradeoff between surface area and throughput. The minimal area $area_{min}$ was determined by performing a synthesis with a large period constraint (30 ns). Then a binary search technique was used, with an upper and lower period constraint limit, to close in on the period constraint closest to the $area_{min} + 10\%$ criteria. These period searches' results are summarised in Figure 4.2. These plots show that the circuit area grows fast below a certain period threshold as the synthesis tool tries to meet the timing constraint by adding additional logic. The criteria of $area_{min} + 10\%$ results in an area already in this fast growth zone, where a small increase in throughput results in a large increase in area and energy. A single picosecond difference here can result in a several percent difference in energy consumption. Furthermore, each decoder will be optimized to a slightly different degree, with a very high sensitivity to synthesis period constraints. This difference in synthesis optimization dominates the circuits' energy performance, making comparing code configurations meaningless.

Because of these discrepancies, we manually selected a period constraint that results in a surface area close to the minimum surface area for all decoder configurations. According to Figure 4.2, $period = 1000$ ps is outside the fast growth region for all configurations and still results in a high operating frequency of 1 GHz. A constant synthesis period constraint

Table 4.1 LA-GaB, code 1024.512.3.103, synthesis results with different grouping parameters

	Period (ps)	Total area (μm)	Total Pwr (nW)		
			$p_e = 0$	$p_e = 0.025$	$p_e = 0.5$
Fully flattened	620	697586	470455	587320	1174760
CNU & VNU preserved	526	723437	550119	673401	1352131
Pipeline registers & LDPC iteration blocks preserved	518	702676	556867	684602	1305063
All major components preserved (CNU, VNU, pipeline registers, LDPC iteration blocks)	524	722581	466118	562235	1081680
No design flattening	521	693059	564025	682891	1250744

enables us to use the same simulation clock period across all decoder configurations, making throughput comparisons more straightforward. This is the period constraint that is used in all subsequent simulations. The resulting information throughput ranges from 48 Gbps to 1.772 Tbps depending on the code configuration.

However, once a designer settles on a given decoder configuration, some additional gains in performance could be achieved by optimizing the period constraint further. The fixed period constraint is sufficient for comparing and predicting the energy of different decoder configurations. We prefer to minimize any relative difference in performance that synthesis optimization can introduce rather than maximize throughput. To achieve the best trade-off between surface area, throughput, and energy, a post-synthesis activity-based energy simulation could be done after each synthesis using realistic test vectors, enabling the use *Energy/bit* as a performance metric to optimize, at the cost of some extra search time. But simulation time is short compared to synthesis time, especially for larger decoders, so adding that extra simulation step would be relatively inexpensive.

4.3 Results

4.3.1 Code Choices

Results are shown for three different codes, chosen to cover a wide range of code lengths and code rates:

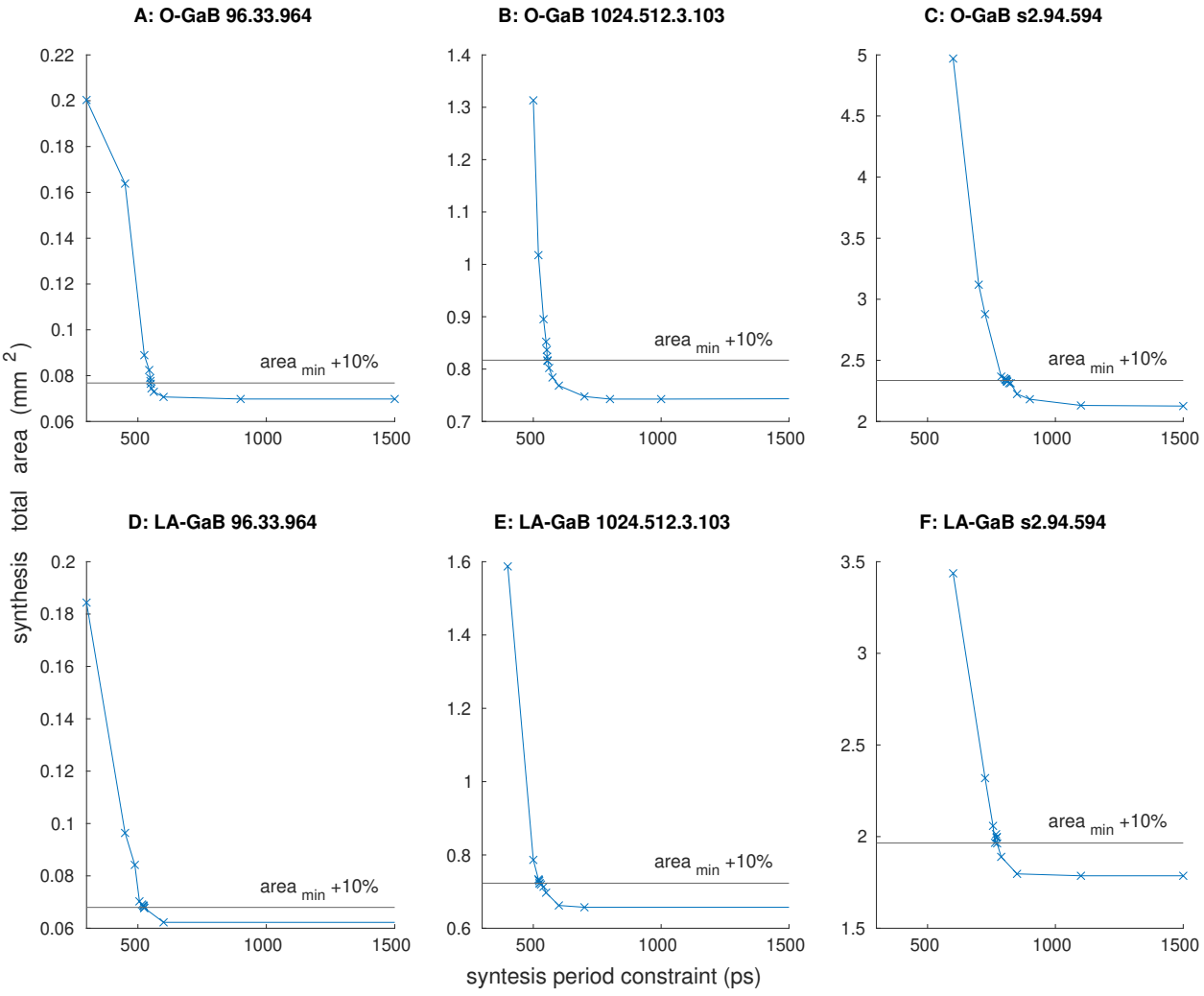

Figure 4.2 Area vs. synthesis period constraint exploration

- 96.33.964 with code length $N = 96$, code rate $R = 0.5$, CN degree $d_c = 6$, VN degree $d_v = 3$ from [42].
- 1024.512.3.103 with code length $N = 1024$, code rate $R = 0.5$, CN degree $d_c = 6$, VN degree $d_v = 3$. This code was randomly generated using the "code5" random code generator [43].
- s2.94.594 with code length $N = 1998$, code rate $R = 0.89$, CN degree $d_c = 36$, VN degree $d_v = 4$ from [42].

All the codes are regular, even though our implementation supports irregular codes. They are also all systematic, which allows for simplifying the last decoding iteration.

4.3.2 Surface Area Results

As discussed in Section 3.1.2, the complexity of VNUs is reduced with LA-GaB compared to O-GaB. Figure 4.3 shows synthesis area results of the sum of all instances of CNU, VNU, and message pipeline registers in both O-GaB and LA-GAB. Syndrome nodes from LA-GaB and decision nodes are included with CNUs and VNUs, respectively. Here, the VNU surface area is roughly cut in half for LA-GaB. Area differences for CNUs and message pipeline registers are minor. Figure 4.4 shows a comparison of the total area of each decoder configuration. We can see here that the gains in overall surface area are modest since VNUs are only one of several components. The codes achieve area reductions of 13.0%, 13.0%, and 19.5%, respectively. We also notice that code rate impacts the total benefit to the surface area since VNUs represent a more significant proportion of total surface area with higher code rates.

Any gain in surface area results in cost savings as fabrication costs scale with surface area. It is also noteworthy that in a fully parallel architecture, the PCM is fully implemented with wires which can lead to a very complex routing problem for large codes. This can result in poor surface area utilization as a lot of area can be used for routing rather than for logic [24]. This problem is reduced using an unrolled architecture and a hard-decision decoder since each message is a single bit. However, congestion can still be an issue for very large codes.

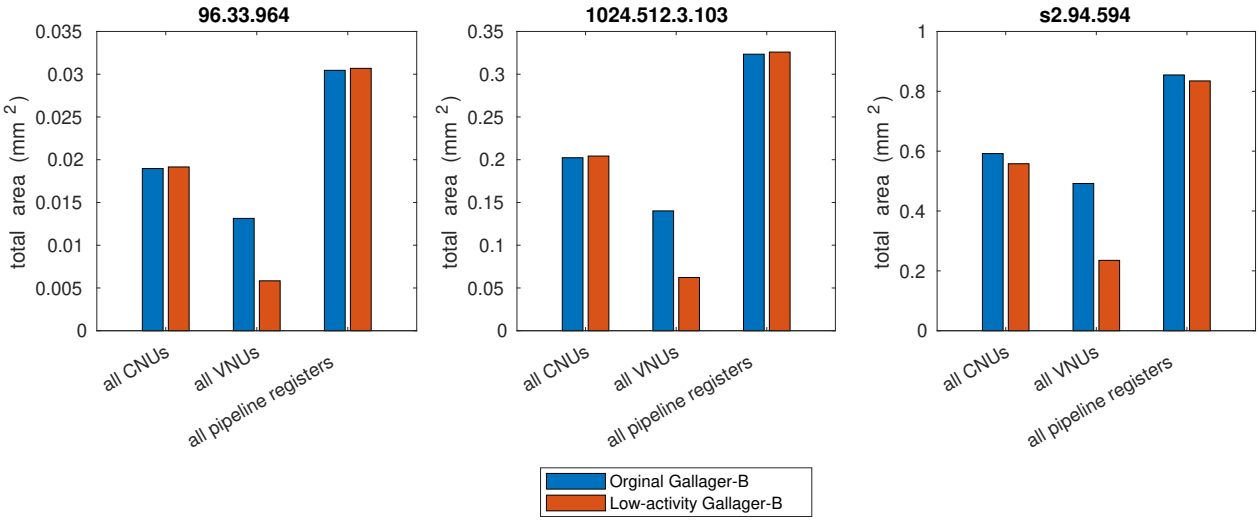

Figure 4.3 Component area comparison

4.3.3 Global Energy Results

Figure 4.5 shows the BER performance and dynamic energy of both O-GaB and LA-GaB ASIC implementations, relative to p_e, the BSC channel error probability. LA-GaB saves more

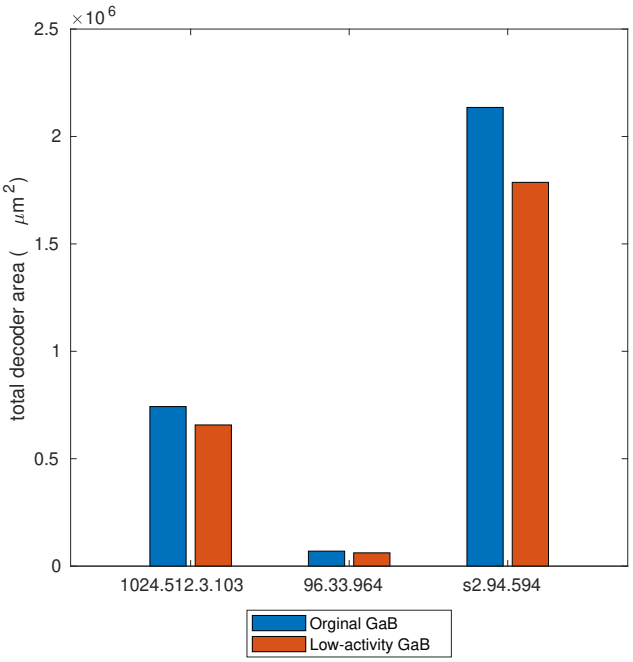

Figure 4.4 Total area comparison

than 50% of dynamic energy per information bit in all decoder configurations tested. This improvement can be attributed to the activity reduction internal to CNUs and VNUs, the storage of lower activity $\overset{\star}{\lambda}$-messages and syndrome vectors inside of pipeline registers, and the gate-count reduction of VNU components. We observe a slight correlation between dynamic energy consumption in O-GaB and p_e despite all λ and γ-messages having 0.5 activity. This is due to internal signals within nodes having an activity dependant on p_e, such as CNUs, where internally, a syndrome calculation is performed, followed by extrinsic exclusions. In that component, the internal syndrome signal, which has a high fanout, will have an activity tied to the error rate.

Figure 4.5 also shows a close match of the energy model to the post-synthesis simulation energy measurements. A more detailed analysis of the energy model fit will be discussed in Section 4.5.

4.3.4 Component Energy Results

Figure 4.6 shows the dramatic reduction in dynamic energy for the CNU and VNU components. Here the sum of energies of all instances of the components across all iterations is used, with the first iteration syndrome calculation included with the other CNUs, and the final decision nodes also included with the other VNUs. A p_e representative of the practical

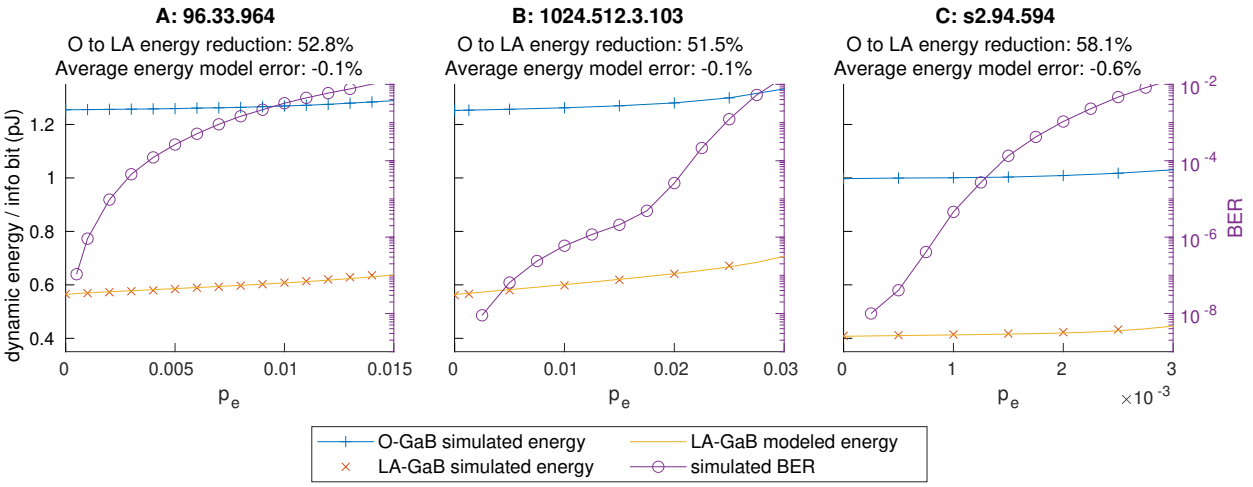

Figure 4.5 BER and dynamic energy/info bit simulation

operating range of each decoder configuration was used. The dynamic energy of CNUs and VNUs in LA-GaB now depends only on p_e and is dramatically reduced compared to O-GaB. The dynamic energy of $\overset{\star}{\lambda}$ pipeline registers depends on $\overset{\star}{\lambda}$-message activity, which is also reduced with LA-GaB. The codeword pipeline registers in Figure 4.6 store both the information and parity portion of received codewords. In LA-GaB, syndrome bits are stored instead of the received codeword parity bits, which have significantly less activity. The codeword information bits always have 0.5 activity in both configurations.

All pipeline registers have significant dynamic energy depending only on the clock activity.

The dynamic energy of CNU and VNU components is null when message activity is null, so their dynamic energy depends only on message activity.

4.3.5 Comparison With the Literature

Given the lack of Gallager-B ASIC implementations in the literature, we compare our implementation results to an OMS [24] and a probabilistic bit-flipping decoder [20]. These decoders were characterized based on data transmission over an additive white Gaussian noise (AWGN) channel. In order to compare the performance with our hard-decision decoder, for which transmission is modeled as a BSC channel, we use a BSC channel having the same error probability p_e as the AWGN channel, given by

$$p_e = \frac{1}{2}\text{erfc}(\sqrt{\frac{E_b}{N_0}}) \ , \tag{4.1}$$

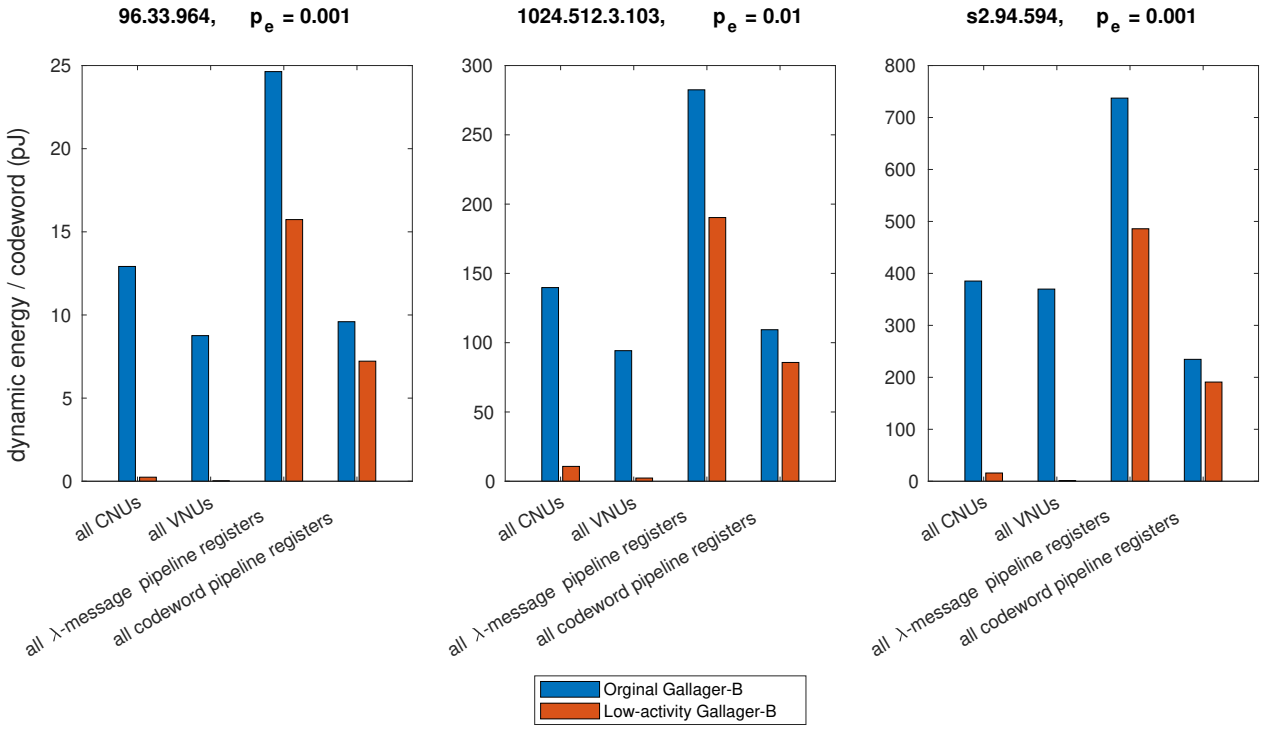

Figure 4.6 Component dynamic energy comparison

where $\frac{E_b}{N_0}$ is the energy per bit to noise power spectral density ratio, or signal to noise ratio (SNR) per bit.

Table 4.2 shows a comparison of various decoder parameters. Decoding performance is compared at a SNR of 4.5dB on the AWGN channel. The comparison with the OMS decoder poses challenges since the decoding algorithms are so different. This decoder class can achieve better decoding performance since more information is available from the channel. But this decoding performance comes at a high cost in hardware complexity. Our hard-decision Gallager-B decoder achieves far higher operating frequency, throughput, area efficiency, and energy/bit but has worse decoding performance. These results confirm the positioning of unrolled Gallager-B decoders as effective for high-throughput, low-energy applications. Compared to the hard-decision bit-flipping implementation of [20], our decoder achieves better throughput, area efficiency, and energy/bit. This probabilistic bit-flipping decoder uses a high maximum iteration limit, affording it excellent error rate performance. Unfortunately, [20] only provides energy information for a short code configuration. Additionally, our decoder has a fixed throughput with no early termination. In contrast, both other decoders have a variable decoding iteration count, which requires buffers at the input and output of the decoder to absorb the variable latency. The implementations presented in the literature do not include these buffers in their area and energy results. As such, we used the worst-case latency and

throughput data as a base for comparison since these are the operating conditions required to guarantee the claimed decoding performance without buffers.

Table 4.2 Comparison with other ASIC LDPC decoder implementations

	This work (LA)			[24]	[20]
Decoding algorithm	Gallager-B			OMS	PPBF
Technology	65nm			65nm	90nm
Stage	post-synthesis			post-fabrication	post-synthesis
Quantization bits	1			4	1
It_{max}	10			8 + 6 post proc.	300
Supply voltage (V)	1.0			1.2	1.2
Clock frequency (MHz)	1000			700	625
Max. latency (ns)	10			240	$>$ 480ns
LDPC code	(96,48)	(1024,512)	(1998,1772)	(2048,1723)	(155,93)
Code rate	0.50	0.50	0.89	0.84	0.60
FER @ 4.5dB	$8 \cdot 10^{-6}$	$4 \cdot 10^{-7}$	$8 \cdot 10^{-4}$	$< 10^{-6}$	$< 10^{-7}$
Min. throughput (Gb/s)	48	512	1772	8.53	0.323
Area (mm^2)	0.06	0.66	1.79	5.05	0.045
Area scaled to 65nm (mm^2)	0.06	0.66	1.79	5.05	0.023
TP/Area scaled to 65nm ($Gb/s/mm^2$)	776.9	775.7	989.9	1.69	13.8
Max. E/bit (pJ)	0.6	0.6	0.4	328.2	51.3

Our implementation is not optimized for maximum throughput, as a fixed clock frequency was used across all configurations. Some configurations can achieve close to double the clock frequency with no additional pipelining and minimal impact on the surface area, as seen in Figure 4.2. Alternatively, a lower supply voltage could have been used while achieving the same clock frequency with lower energy consumption, further improving our energy/bit metrics.

4.4 Validation of the Activity Model

We can measure the decoder's internal signal activities using Monte-Carlo simulation with the software model. We used these experimental activities to validate our analytical DE based activity model. Figure 4.7 shows that the analytical results are very close to the experimental ones for the large codes, validating that this analytical method is appropriate for estimating signal activities. A single p_e is shown for each code for clarity. For the smaller code in Figure 4.7-A, the more significant difference is attributable to the small code length. The DE method is based on the hypothesis that codes are cycle-free [36], which is valid only

for codes of infinite length. Thus, this assumption is less accurate for small codes and could explain the larger error in the DE prediction.

4.5 Validation of the Energy Model

An energy model is only presented for the LA-GaB decoder. Therefore this section only describes LA-GaB simulation results even when not specified. We will start by searching for α and β coefficients for use in the energy model outlined in Section 3.3.3 for each decoder sub-component. We will then compare the energy model to experimental measurements for each sub-component and the entire decoder.

4.5.1 Check Node Units

To find a generic α_{CNU} parameter for use in equation (3.20), we should verify that CNU energy is linear in relation to output activity in the region of interest. We set up an experiment to verify this. The CNU was simulated separately with input signal distributions similar to those during decoding.

In LA-GaB, CNUs have two types of input:

- a syndrome bit calculated at the first decoding iteration,
- $\overset{\star}{\lambda}$-messages (d_c of them).

Both of these input types have distributions that depend on p_e. The syndrome activity is constant across decoding iterations and relates to p_e through equation (3.18). On the contrary, $\overset{\star}{\lambda}$-messages have a distribution that changes at every iteration following equation (3.14). However, to simplify the experiment, we want to test the circuit with $\overset{\star}{\lambda}$-message distributions that depend only on p_e and not the decoding iteration. For this, we notice that $\overset{\star}{\lambda}$-messages are intermediate error estimations, therefore if the decoder converges to the transmitted codeword, the distribution of $\overset{\star}{\lambda}$ corresponds to the channel error distribution, that is $P_1(\overset{\star}{\lambda}) \approx p_e$.

Now we can perform energy simulations on the CNU component for a range of input activities and d_c values. Output test vectors are recorded and analyzed to derive output activity, as our energy model relies on output activity rather than input activity. We consider a range of output activity values from 0 to 0.2 since this corresponds to the range observed during Monte-Carlo simulations of the decoder when it converges to low BER values.

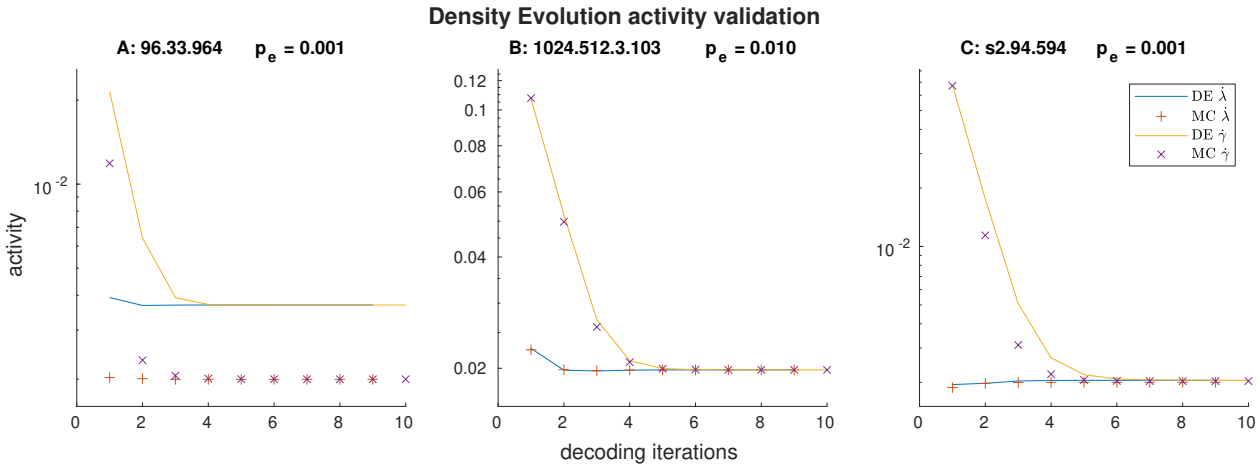

Figure 4.7 Comparison of Monte-Carlo measured activities to analytical Density Evolution results

Figure 4.8 shows that CNU dynamic energy is linear in relation to output activity in the range of interest and can be modeled using a linear fit with zero intersect. The slope parameter α_{CNU} is extracted from this linear fit and will be used in the energy model. Four CN degrees are shown to illustrate that this remains true across a wide range of d_c values.

Figure 4.9 shows a summary of α_{CNU} coefficients for a wide range of d_c values. We notice some discontinuities around $d_c = 16$ and $d_c = 32$. This matches the change in synthesis topology that we observed in Section 3.2.2, and corroborates the idea that the synthesis tool might not be choosing the optimal topology in terms of energy for lower degrees. These simulations consider realistic signal activities that the synthesis tool has no knowledge of and do not consider the circuit surface area. Because of these discontinuities, we have used separate α_{CNU} coefficients for each d_c rather than using a single fit across all degrees.

Figure 4.10 compares energy measurements from simulation to energy estimations based on our energy model presented in equation (3.20) and Monte-Carlo activity measurements. Here the sum of all CNU instances in the decoder is used. We can see that the model fits the data very well. It is important to remember that this model makes no special treatment of the first iteration syndrome calculation. Despite its different activity and architecture, it is treated like any other CNU stage. This is particularly apparent for the larger code in Fig. 4.10-C, where we see a constant offset. We expect zero dynamic energy for all iteration stages except the first when $p_e = 0$.

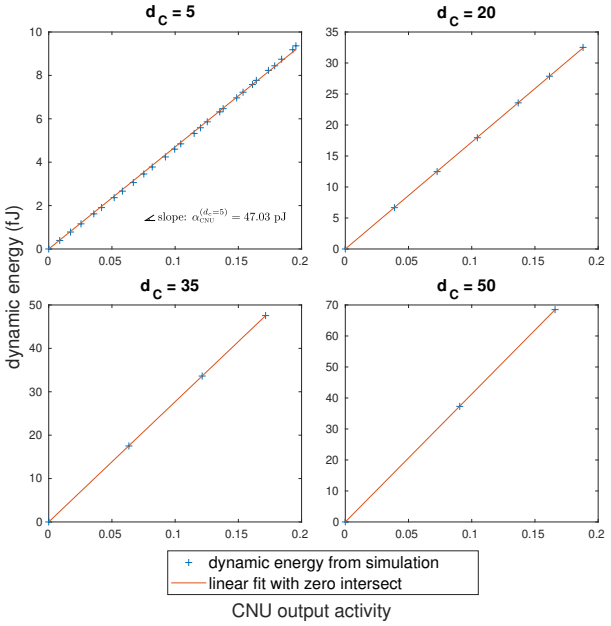

Figure 4.8 CNU energy vs. output activity

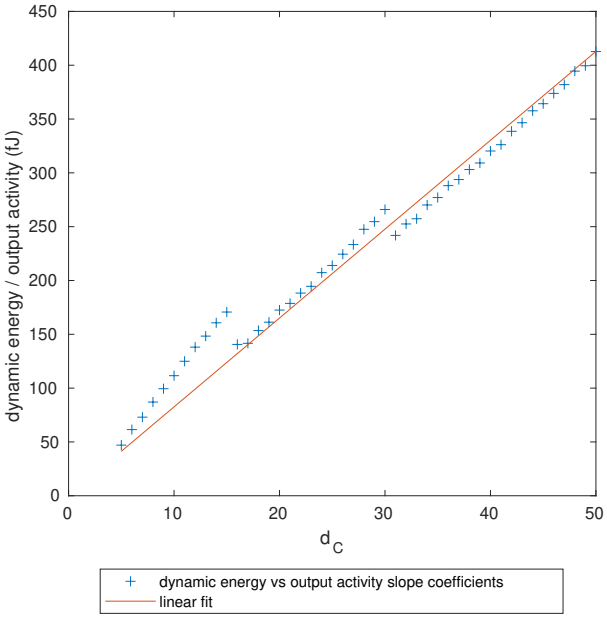

Figure 4.9 Compilation of α_{CNU} slope coefficients from Figure 4.8 for each d_c degree. CNU energy/output activity vs. d_c

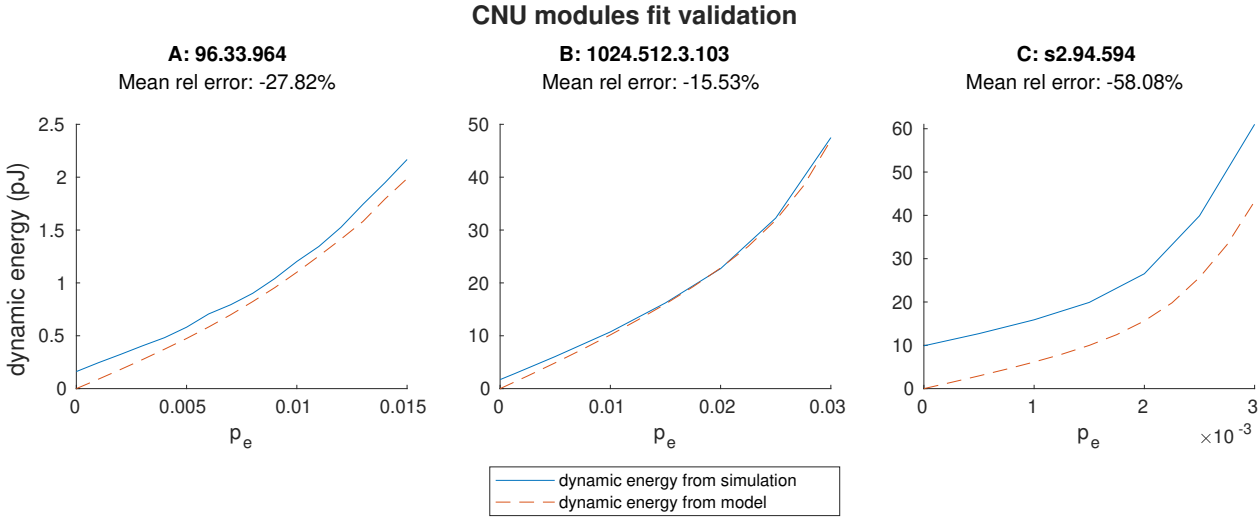

Figure 4.10 CNU energy model validation

4.5.2 Variable Node Units

We used the same methodology for VNUs as for CNUs without the added complexity of estimating syndrome activity. Again linearity is shown in the output activity dimension for a range of d_v values in Fig. 4.11. Figure 4.12 shows a lot of noise in the d_v dimension, making a linear fit not very useful. We also used discrete α_{VNU} fit coefficients for each d_v, just like for CNUs.

Figure 4.13 compares the dynamic energy from simulation with the energy model presented in equation (3.21) using Monte-Carlo activity measurements, for the sum of all VNU instances, including the final iteration decoding stage. The model fits reasonably well but tends to underestimate the dynamic energy by a constant factor. Our fit coefficients were obtained by simulating the VNUs out of the circuit using a simple wrapper with signal latching. Herefore it is not unexpected to see some differences between this simple test setup and the full decoder. In particular, the test circuit has minimal wire lengths. In contrast, the actual decoder implements the full parity check matrix with wires resulting in wire lengths and subsequent capacitive loads growing with the size of the parity check matrix.

There could also be an impact from the choice of drive strength of the standard cells chosen by the synthesis tool to compensate for this wire length, although this did not appear to be the case in our verification of the synthesis netlists. Fanout is not a significant factor since, from our study of node synthesis topologies, the message signals only drive up to two different gates in subsequent nodes. Critical path optimizations can significantly affect the energy consumption of any component as the synthesis tool searches for a tradeoff between area, energy, and propagation delay. Our component topology search noticed significant design changes in topology and standard cell choices when tight timing constraints were set. We have largely mitigated this effect by opting for a synthesis period constraint sufficiently large to avoid this. A factor could be added to the energy model to compensate for code size. Nether the less, the energy estimates for the VNU are still helpful for a high-level energy model.

4.5.3 Pipeline Registers

Figures 4.14 and 4.15 show the linearity of banks of flip-flop components that constitute pipeline registers with different bit-widths W. As expected, pipeline registers' energy depends partially on the clock activity, such that there is dynamic energy consumption even with no message signal activity. Therefore we need to fit both an α_{REG} and a β_{REG} coefficient in the W direction.

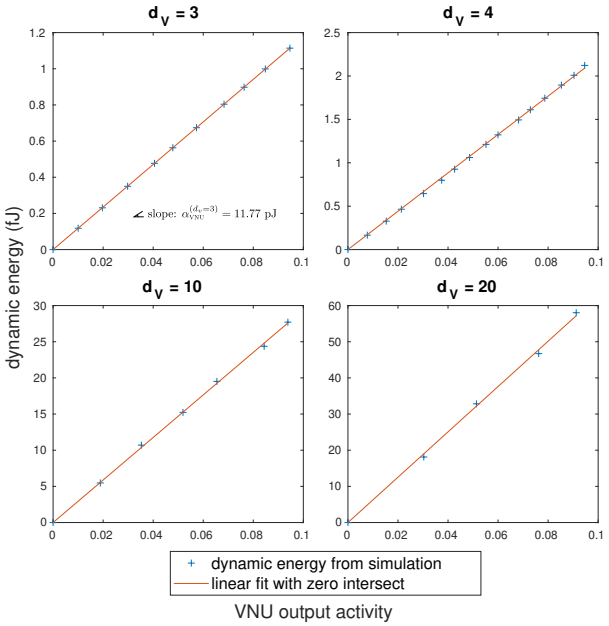

Figure 4.11 VNU energy vs. output activity

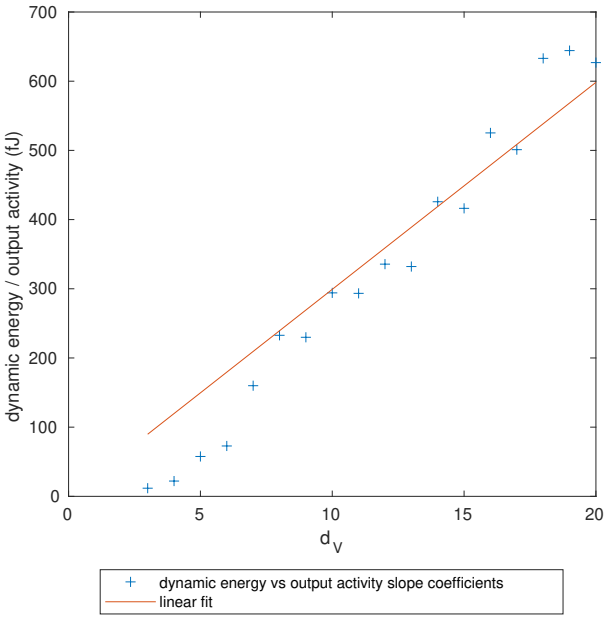

Figure 4.12 Compilation of α_{VNU} slope coefficients from Figure 4.11 for each d_v degree. VNU energy/output activity vs. d_v

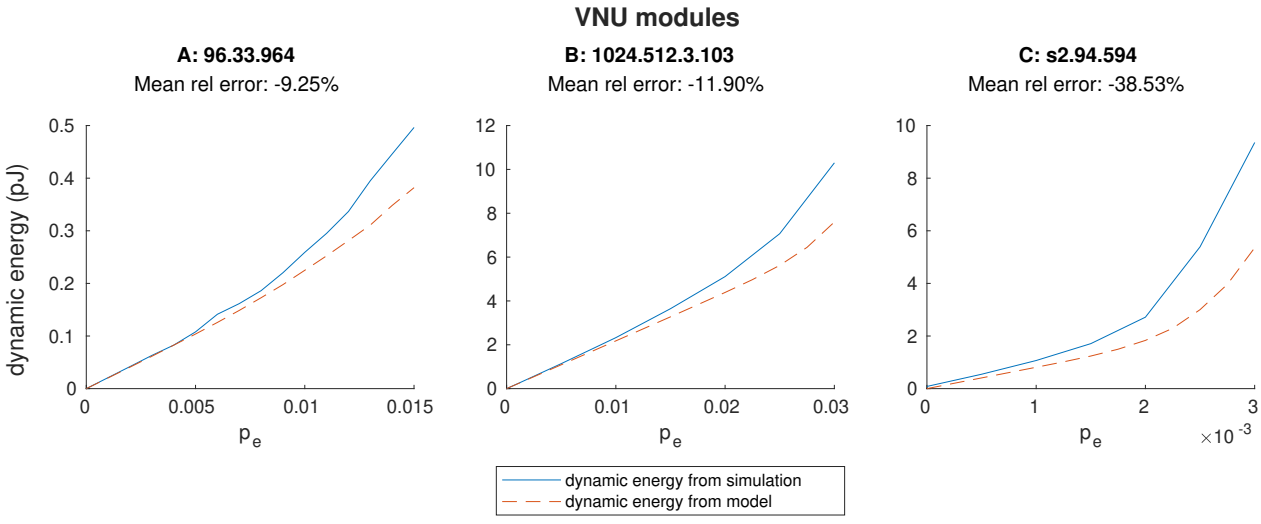

Figure 4.13 VNU energy model validation

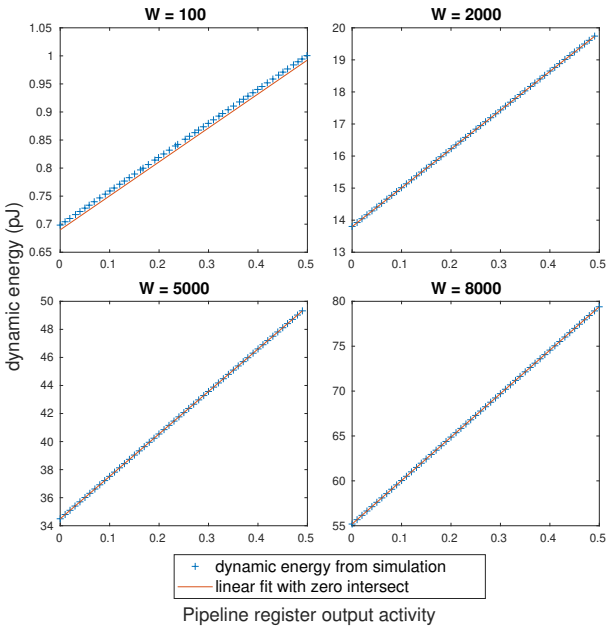

Figure 4.14 Pipeline registers energy vs. output activity

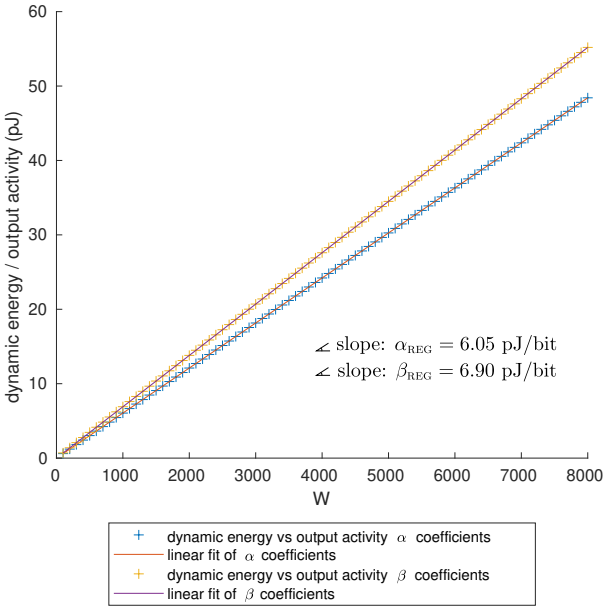

Figure 4.15 Compilation of α_{REG} slope coefficients and β_{REG} offset coefficients from Figure 4.14 for each register bank width W. Pipeline registers energy/output activity vs. W

Some clock-gating logic implemented for pipeline registers at each decoding iteration is controlled by an enable signal propagated through the pipeline. Since all pipeline registers of a given iteration use the same control signal, this clock-gating logic is shared across all pipeline register types. This logic's energy is constant while the decoder is in operation, as its activity depends only on the clock's activity and that of the control signal. Since this logic is shared, we must compare the sum of all pipeline registers plus this clock-gating logic to the fit model to obtain comparable results. Figure 4.16 shows that our model presented in equations (3.22), (3.23), and (3.24) fits closely to the hardware decoder's dynamic energy for pipeline registers.

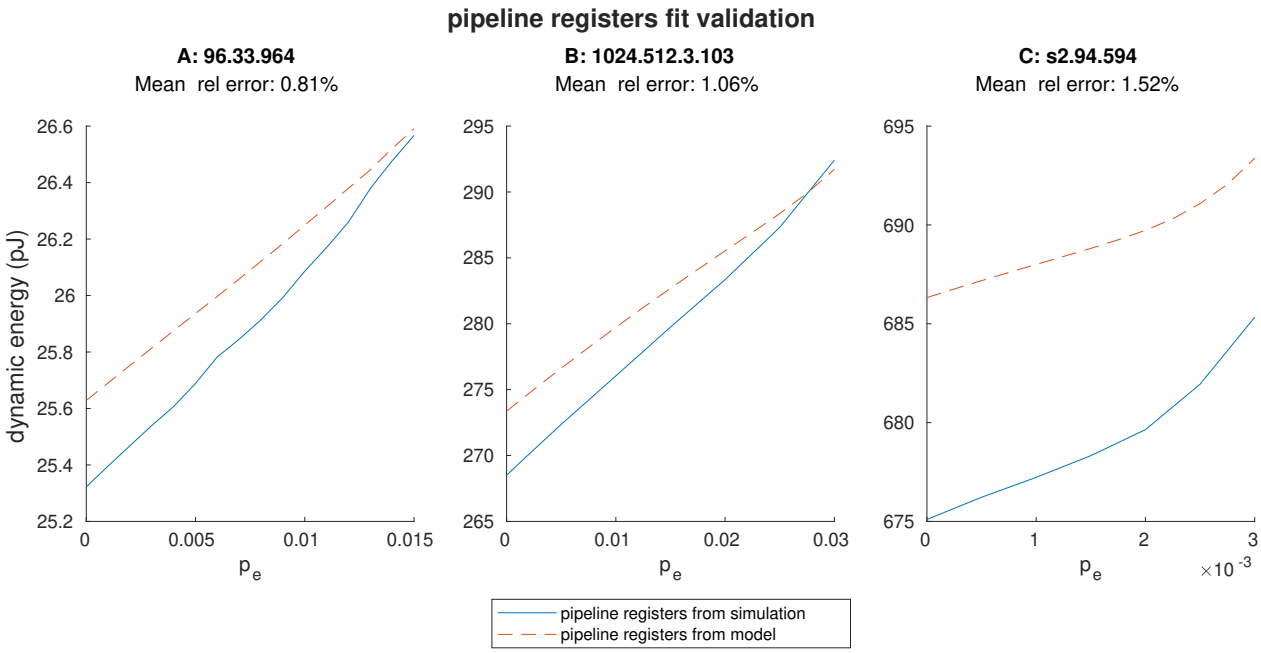

Figure 4.16 Pipeline registers energy model validation

4.5.4 Bringing It All Together

Table 4.3 summarizes the linear fit coefficients from Sections 4.5.1, 4.5.2 and 4.5.3 for the degrees used in the codes we are testing. These coefficients are used in the energy model equations (3.20) through (3.24).

Combining the energy models of CNUs, VNUs, and pipeline registers, we obtain an energy model of the entire decoder, as described by equation (3.25). Figure 4.17 compares this global energy model to the dynamic energy of the simulated decoder and shows that the model fits the data very closely. The previous component fit validations all used Monte-Carlo-based activity measurements, but this time we incorporate DE based activities. The fitment errors for pipeline registers and node units have opposite signs, which aid in getting such small global error figures.

To summarize, the steps to recreate these energy estimations are:

1. Perform energy simulations on individual VNUs, CNUs, and flip-flop banks with degree representative of the codes of interest in the implementation technology of choice. Perform linear fits on these results to obtain the technology-dependant α and β parameters described in this energy model.

2. Choose a PCM to study, a decoding iteration count, and a GaB threshold.

Table 4.3 Energy model fit coefficients for TSMC 65nm GP

Component	Degree	α (fJ)	β (fJ)
CNU	$d_c = 6$	61.45	–
	$d_c = 36$	288.14	–
VNU	$d_v = 3$	11.77	–
	$d_v = 4$	31.33	–
REG	–	6.05	6.90

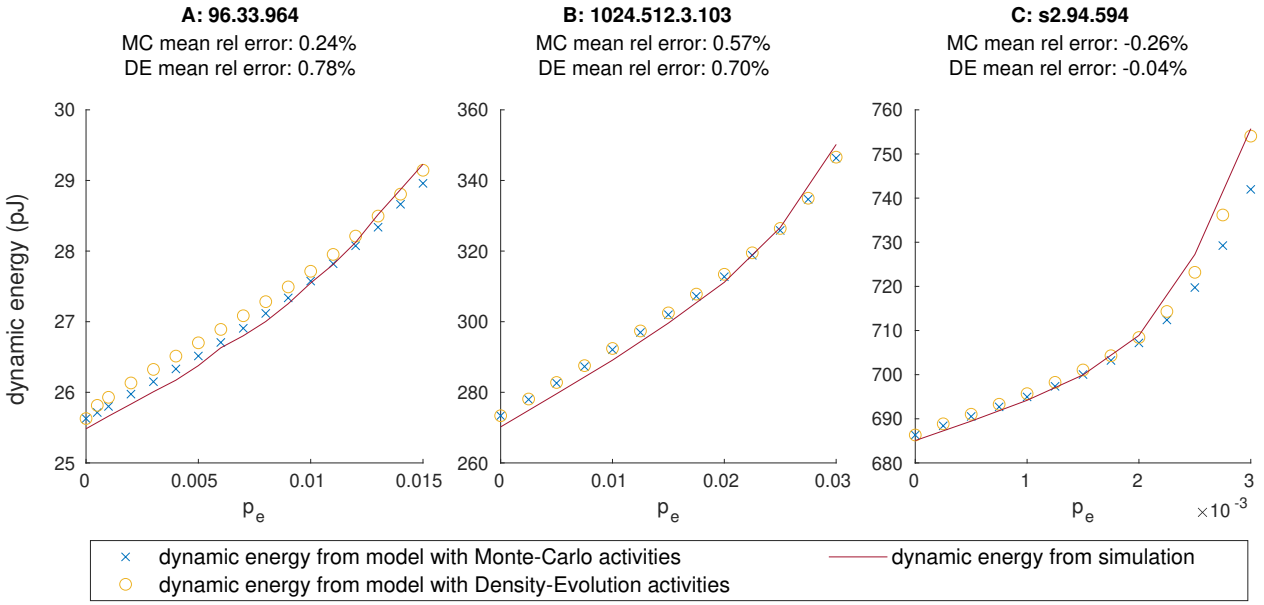

Figure 4.17 Global energy model validation

3. Perform analytical DE analysis using those parameters to obtain message activities.
4. Apply the energy model using the chosen parameters and DE activities.
5. Repeat steps 2 - 4 until the desired tradeoff between circuit energy and decoding performance has been achieved.

This process is a great improvement from what would have previously been required, which would have involved implementing a hardware decoder in the technology of choice, then running synthesis and energy simulation for each set of parameters. The synthesis and energy simulation can require several hours for larger codes, whereas the application of our model can be run in seconds. The initial step of running energy simulations on individual components is not very time-consuming since individual components only comprise a handful of standard cells. These simulations must only be run once on a given technology to compare

a whole family of decoder configurations.

The energy model methodology closely resembles the method used by state-of-the-art circuit synthesis tools such as `Genus` to measure energy. This explains why we achieve such a close match between the model and simulation results. The precision of our model mainly results from the accuracy with which we can predict signal activities within the circuit.

CHAPTER 5 CONCLUSION

5.1 Summary of Works

We have presented a modified version of the Gallager-B LDPC decoding algorithm named Low-Activity Gallager-B (LA-GaB) that involves a change of variable that improves the energy efficiency of the decoder for many hardware architectures, including fully-parallel fully-unrolled architectures. The codeword syndrome is pre-computed once at the start of decoding, making messages reflect only corrections to the codeword, which have a much lower switching probability at low error rates. We have also demonstrated the resulting gains in dynamic energy on an ASIC implementation using post-synthesis activity-based energy simulations. We have shown that a reduction of more than 50% of dynamic energy is achievable while reducing surface area. We have adapted an analytical density evolution probabilistic method to predict the message activity of the modified decoder and validated it experimentally. Finally, we have developed a simple high-level energy model of the modified decoder that can be used with the density evolution activities to predict the dynamic energy consumption of any configuration of unrolled flooding-schedule Gallager-B decoder in a given implementation technology.

5.2 Limitations

Our validation is limited to fully parallel, fully unrolled decoder implementations that use flooded scheduling, which is not flexible regarding parity check matrix configurations or decoding iteration count past the implementation stage and does not benefit from the faster convergence speed of layered scheduling. This decoder configuration has the worst possible dynamic energy as successive clock cycles always process a different codeword. All message signals have a switching probability of $\frac{1}{2}$, hence why we can achieve such dramatic gains. This high activity is also found in architectures where physical circuits are mapped to different nodes on successive cycles, as is typical in QC-LDPC implementations. In the special case of a fully parallel but not unrolled architecture, where the full Tanner graph is mapped out to hardware but not decoding iterations, physical circuits are mapped to the same Tanner graph edge and the same codeword in successive clock cycles resulting in lower signal activity, thus reducing the benefit of using LA-GaB.

Our model only considers dynamic energy consumption and not leakage energy consumption. Dynamic energy is usually much larger than leakage energy, but this ratio differs for every

implementation technology node [13]. We also only used the TSMC 65nm LP technology, which is an older technology, but we would expect to see similar results in other technologies.

Finally, the DE method used to predict activity can be less accurate for very small codes, probably because the cycle-free assumption becomes inaccurate.

5.3 Future Research

Further lowering of message activities could be achieved by introducing intermediate decisions at specific decoding iterations, reducing the message activities for the following iterations. There would then be a trade-off to explore between the added surface area and energy from the added decision logic and the reduced activity.

The scope of this research is limited to flooded scheduling for simplicity. The same methodology could be applied with layered scheduling, as the number of decoding iterations necessary to reach a given error rate performance would be reduced. Clock gating could also be used as a form of early termination where message pipeline registers are deactivated for unnecessary decoding iterations to save energy [7].

Further exploration of the throughput, energy, and area trade-off could be studied, particularly with the control of pipelining stages and energy-focused clock period optimization. There is also a trade-off between error rate performance and surface area, as either more decoding iterations can be added or a larger code can be used to improve error rate performance at the expense of surface area.

The energy model presented in this thesis is well suited for the construction of irregular codes that minimize energy under fixed error correction and throughput constraints.

REFERENCES

[1] D. Hui *et al.*, "Channel coding in 5G new radio: A tutorial overview and performance comparison with 4G LTE," *IEEE Vehicular Technology Magazine*, vol. 13, no. 4, pp. 60–69, 2018.

[2] A. E. Cohen and K. K. Parhi, "A low-complexity hybrid LDPC code encoder for ieee 802.3an (10gbase-t) ethernet," *IEEE Transactions on Signal Processing*, vol. 57, no. 10, pp. 4085–4094, 2009.

[3] R. P. Fabris Hoefel, "IEEE 802.11ax: On performance of multi-antenna technologies with LDPC codes," in *2018 IEEE Seventh International Conference on Communications and Electronics (ICCE)*, 2018, pp. 159–164.

[4] S. Jeon and B. V. K. Vijaya Kumar, "Performance and complexity of 32 k-bit binary LDPC codes for magnetic recording channels," *IEEE Transactions on Magnetics*, vol. 46, no. 6, pp. 2244–2247, 2010.

[5] K.-C. Ho *et al.*, "A 3.46 Gb/s (9141,8224) LDPC-based ECC scheme and on-line channel estimation for solid-state drive applications," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 1450–1453.

[6] G. Dong, N. Xie, and T. Zhang, "On the use of soft-decision error-correction codes in nand flash memory," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 2, pp. 429–439, 2011.

[7] M. Herrmann, C. Kestel, and N. Wehn, "Energy efficient FEC decoders," in *2021 11th international symposium on topics in coding (ISTC)*, 2021, pp. 1–5.

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

[9] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Transactions on Communications*, vol. 50, no. 3, pp. 406–414, 2002.

[10] J. Chen *et al.*, "Reduced-complexity decoding of LDPC codes," *IEEE Transactions on Communications*, vol. 53, no. 8, pp. 1288–1299, 2005.

[11] M. Jiang *et al.*, "An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes," *IEEE Communications Letters*, vol. 9, no. 9, pp. 814–816, 2005.

[12] W. Yu *et al.*, "Complexity-optimized low-density parity-check codes for gallager decoding algorithm B," in *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, 2005, pp. 1488–1492.

[13] A. Golda and A. Kos, "Energy losses in digital CMOS integrated circuits: State-of-the-art and future trends," in *2008 Int. Conf. on Signals and Electronic Systems*, 2008, pp. 113–116.

[14] P. Schläfer *et al.*, "A new dimension of parallelism in ultra high throughput LDPC decoding," in *SiPS 2013 Proceedings*, 2013, pp. 153–158.

[15] S. Brown, J. Nadal, and F. Leduc-Primeau, "Low-activity gallager-b LDPC decoding," 2023, manuscript submitted for publication.

[16] W. E. Ryan and S. Lin, *Channel Codes*. Cambridge: Cambridge University Press, 2009.

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

[18] P. Kocanda and A. Kos, "Static and dynamic energy losses vs. temperature in different CMOS technologies," in *2015 22nd Int. Conf. Mixed Design of Integrated Circuits & Systems (MIXDES)*, 2015, pp. 446–449.

[19] J. Rabae, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits: A Design Perspective*. Massachusetts Institute of Technology, Cambridge: Pearson Education, 2002.

[20] K. Le *et al.*, "A probabilistic parallel bit-flipping decoder for low-density parity-check codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 1, pp. 403–416, 2019.

[21] Q. Zhang *et al.*, "A 3.01 mm2 65.38gb/s stochastic LDPC decoder for IEEE 802.3an in 65 nm," in *2019 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2019, pp. 271–274.

[22] O. A. Rasheed, P. Ivaniš, and B. Vasić, "Fault-tolerant probabilistic gradient-descent bit flipping decoder," *IEEE Communications Letters*, vol. 18, no. 9, pp. 1487–1490, 2014.

[23] Y.-L. Wang *et al.*, "A low-complexity LDPC decoder architecture for WiMAX applications," in *Proceedings of 2011 International Symposium on VLSI Design, Automation and Test*, 2011, pp. 1–4.

[24] Z. Zhang *et al.*, "An efficient 10GBASE-T ethernet LDPC decoder design with low error floors," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 4, pp. 843–855, 2010.

[25] I. Tsatsaragkos and V. Paliouras, "A flexible layered LDPC decoder," in *2011 8th International Symposium on Wireless Communication Systems*, 2011, pp. 36–40.

[26] A. Blanksby and C. Howland, "A 690-mw 1-gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, 2002.

[27] A. Balatsoukas-Stimming *et al.*, "A fully-unrolled LDPC decoder based on quantized message passing," in *2015 IEEE Workshop on Signal Processing Systems (SiPS)*, 2015, pp. 1–6.

[28] L. Lopacinski *et al.*, "Ultra high speed 802.11n LDPC decoder with seven-stage pipeline in 28 nm CMOS," in *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, 2022, pp. 1–5.

[29] M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 976–996, 2003.

[30] F. Kschischang and B. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, pp. 219–230, 1998.

[31] T. Brack *et al.*, "A synthesizable IP core for WIMAX 802.16E LDPC code decoding," in *2006 IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications*, 2006, pp. 1–5.

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

[33] W. Wang, G. Choi, and K. K. Gunnam, "Low-power VLSI design of LDPC decoder using DVFS for AWGN channels," in *2009 22nd International Conference on VLSI Design*, 2009, pp. 51–56.

[34] C. Zhang *et al.*, "High energy-efficient LDPC decoder with AVFS system for NAND flash memory," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–4.

[35] K. Cushon *et al.*, "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, 2010.

[36] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599–618, 2001.

[37] J. Nadal *et al.*, "Towards an accurate high-level energy model for LDPC decoders," in *2021 11th International Symposium on Topics in Coding (ISTC)*, 2021, pp. 1–5.

[38] A. Amaricai and O. Boncalo, "Cost effective FPGA implementation for hard decision LDPC decoders," in *2016 24th Telecommunications Forum (TELFOR)*, 2016, pp. 1–4.

[39] B. Ünal *et al.*, "Analysis and implementation of resource efficient probabilistic Gallager B LDPC decoder," in *2017 15th IEEE International New Circuits and Systems Conference (NEWCAS)*, 2017, pp. 333–336.

[40] *TSMC N65LP Standard Cell Library Datasheet*, Taiwan Secmiconductor Manufacturing Company, 2011.

[41] M. Jeruchim, "Techniques for estimating the bit error rate in the simulation of digital communication systems," *IEEE Journal on Selected Areas in Communications*, vol. 2, no. 1, pp. 153–170, 1984.

[42] D. J. MacKay. Encyclopedia of sparse graph codes. [Online]. Available: https://www.inference.org.uk/mackay/codes/data.html

[43] ——. code5 random code generator. [Online]. Available: https://www.inference.org.uk/mackay/code/code.tar.gz

APPENDIX A NUMERICAL EXAMPLE OF THE DECODING PROCESS

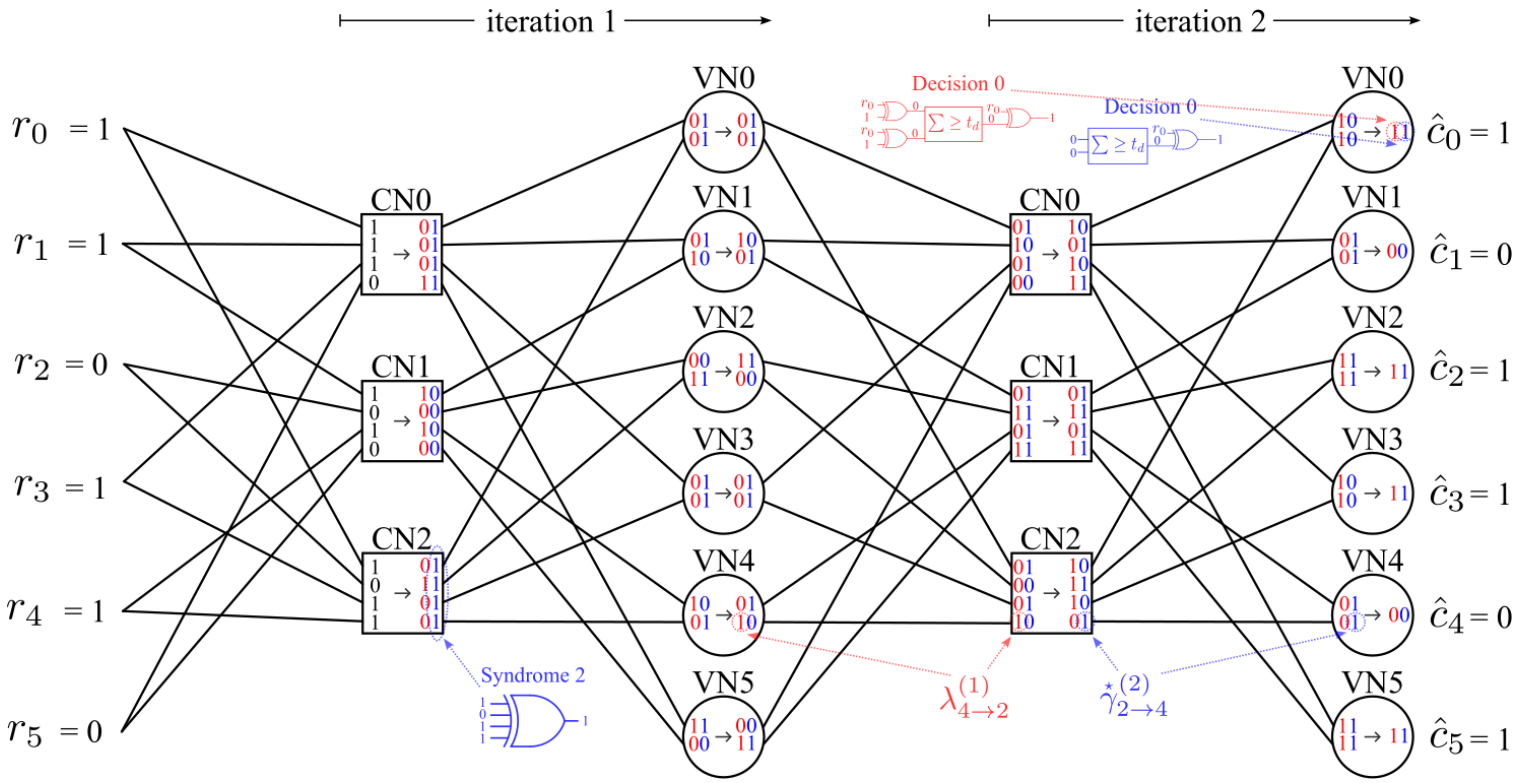

Figure A.1 Numerical example comparing O-GaB decoding in red to LA-GaB in blue. The majority vote thresholds are $t_h = t_d = 1$. For systematic codes, the a posteriori estimated information vector $\hat{b}$ is a sub-vector of length K of the a posteriori estimated codeword vector $\hat{c}$.