University of Alberta

Digit-Online LDPC Decoding

by

Philip Andrew Marshall

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer, Microelectronic Devices, Circuits and Systems

Department of Electrical and Computer Engineering

Abstract

Highly parallel VLSI implementations of low-density parity-check (LDPC) block code decoders have a large number of interconnections, which can result in designs with low logic density. Bit-serial architectures have been developed that reduce the number of wires needed. However, they do not fully realize the potential for deeply pipelined serial data processing.

Digit-online arithmetic allows operations to be performed in a serial, digit-by-digit manner, making it ideal for use in implementing a digit-serial LDPC decoder. Digit-online circuits for the primitive operations required for an offset min-sum LDPC decoder are simple, and allow deep pipelining at the digit level. A new hardware architecture for LDPC decoding is demonstrated, using redundant notation to allow for most-significant-digit-first processing of log-likelihood-ration (LLR) messages at all nodes.

We examine the effect of changing the precision of LLRs on the throughput, area and energy efficiency of bit-parallel and digit-online decoders for the irregular WiMAX rate 3/4A length-1056 code. Both single-frame and frame-interlaced decoding are considered. To examine post-layout and code size effects, 9-bit bit-parallel and digit-online decoders for the irregular WiMAX rate 3/4A and rate 5/6 codes are compared for code lengths varying from 576 to 2304.

Post-layout decoder results are presented for the (2048,1723) 10GBASE-T LDPC code in a general-purpose 65-nm CMOS technology. The decoder requires a core area of 10.89 mm^2 and operates at a clock frequency of 980 MHz. Decoding can be done with a message precision of 4 or 10 bits. With 4-bit precision, the decoder achieves a coded throughput of 82.8 Gbit/s, 73% higher than the state-of-the-art published decoder. We extend the message precision of previously published 10GBASE-T decoders from 4-5 bits to 10 bits. In this 10-bit mode we achieve a throughput of 41.8 Gbit/s, only 12% less than the state-of-the-art 4-bit decoder.

Acknowledgements

This thesis would not have been possible without the help and support of many people around me. There is no way to thank everyone by name, so I apologize for the inevitable omissions.

First and foremost I would like to thank my wife Erika for her constant love, support, and (especially) patience as I completed my graduate studies. There are no words to express how much I love and appreciate her.

I would not have been able to enjoy such a depth of education without considerable financial support from my grandfather (and fellow engineer) Jack Longworth. I have also always received unconditional support from my parents, my brother, and the rest of my family.

This thesis is the result of collaboration with my supervisors Duncan Elliott and Vincent Gaudet. Their ideas, knowledge, experience, and understanding have vastly enriched my education and research.

My work at the University of Alberta has only been possible by collaborating with my peers, and I would be remiss if I did not mention them. Leendert van den Berg and Tyler Brandon were instrumental in familiarizing me with graduate studies and the VLSI lab. I genuinely enjoyed working on various projects with Maziyar Khorasani, Mohammad Behnam, Russell Dodd, Brendan Crowley, Wesam Al-Haddad, and Benjamin Martin. John Koob and Andrew Hakman provided much-needed technical support. As well as being a true friend, Timothy Furtak knows more about algorithms and programming than I could ever hope to, and he has always been willing to provide help in those areas.

Finally, I would like to acknowledge the support of several research organizations. The Natural Sciences and Engineering Research Council of Canada (NSERC) provided financial support for this work, and CMC Microsystems provided software licensing and access to process technology kits. The synthesis and post-layout results present in this thesis were possible thanks to computing support from the High Performance Computing Cluster at the Integrated Nanosystems Research Facility (INRF).

Table of Contents

List of Tables

List of Figures

Nomenclature

List of Symbols

β	The offset constant for offset min-sum decoding.
δ	Initial delay. The processing delay in digits of a digit-online operation.
d^+	The positive bit of a signed-binary digit.
d^-	The negative bit of a signed-binary digit.
d_c	Check node degree.
d_v	Variable node degree.
E/b	Energy/bit. A measurement of the energy efficiency of a decoder.
$\boldsymbol{H}$	The parity-check matrix for an LDPC code.
K	The number of information bits in a code.
λ_i	An LLR message from variable node to check node.
L_i	An LLR message from check node to variable node.
M	The number of parity check nodes in a code.
N	Code length, or number of variable nodes in a code.

List of Terms

ASIC	Application-specific integrated circuit. A chip that is customized for a single purpose rather than being general-purpose.
Bit-serial decoding	A form of message-passing decoding in which messages are sent serially bit-by-bit between nodes.
Code rate	The fraction of data in a code that can transmit useful information.
Compare-select	An operation that takes two input operands, compares them, and selects the minimum or maximum of the two.
Digit-online	An operation that processes data serially digit-by-digit and has a small, fixed processing delay (in digits).
Forward error correction	A method of limiting communication errors which involves sending extra, redundant, data that allows for the repair of some errors.
Latency	In this thesis, latency generally refers to the processing delay (in seconds) to decode a codeword.
LDPC	Low-density parity-check codes are a class of forward error correcting codes.
LLR	Log-liklihood ratio. A measure of relative probability on a logarithmic scale.
LSB, LSD	Least-significant-bit/digit. The bit/digit of a value with the smallest weight.
Message-passing algorithm	An iterative decoding method.
Min-sum decoding	A message-passing algorithm used to decode LDPC codes based on an approximation of the sum-product algorithm.
MSB, MSD	Most-significant-bit/digit. The bit/digit of a value with the largest weight.
Redundant notation	A method of encoding numbers that does not have a unique representation for each value.
Signed-binary	A numeric representation which uses the digit set $\{-1, 0, 1\}$ and requires two bits per digit of storage.
Sum-product decoding	A message-passing algorithm used to decode LDPC codes.
TAR	Throughput/area ratio. An area-normalized measure of decoding throughput.
Throughput	The rate at which a decoder can process data (in bits/second).

Chapter 1

Introduction

1.1 Motivation

Communications systems must transmit information across noisy channels. As a result, there is a finite probability that the received data is not identical to the sent data. Since many applications require the transmission of exact information, error correcting codes are used to ensure reliable communications without retransmission.

Low-density parity-check (LDPC) codes are a popular class of error correcting codes because they approach the channel capacity of additive-white-Gaussian-noise (AWGN) channels while remaining relatively simple to decode. The ease in which they lend themselves to highly parallel implementations has resulted in many high-throughput ASIC designs in the literature.

However, practical considerations limit the amount of parallelism that can actually be achieved. The area efficiency of silicon designs has traditionally decreased with parallelism. This effect is worse at higher computational precisions where more interconnect wires are needed.

Using serial communication on chip has shown promise as a method to reduce the number of wires. It is an area of active research to develop such systems. Current bit-serial decoders do not achieve fully serial processing as they must buffer full-precision messages in parallel and reverse the order in which digits arrive (most-significant-bit-first or least-significant-bit-first) at different nodes.

1.2 Thesis Outline and Contributions

This thesis presents a new LDPC decoding architecture which performs all calcuations in a fully digit-serial manner. Using digit-online arithmetic, we can develop a system which processes all data in a most-significant-digit-first fashion. Digit-online operations are simple to implement and can be deeply pipelined. By exploiting the principles of online arithmetic, we achieve a decoder which does not require parallel storage of any messages during decoding.

An introduction to LDPC block codes is provided in Chapter 2. Iterative decoding with belief-propagation algorithms is explained, and existing decoder architectures are examined. In particular, we explain the shortcomings of these architectures. Hardware implementations from the literature are compared. Chapter 3 provides an introduction to online arithmetic, and outlines the operations that are used to implement the serial data processing used in the proposed decoder. We examine the issue of processing order and compare several redundant notations.

In Chapter 4 we use the principles of online arithmetic to develop a novel approach to belief propagation that allows serial processing of data using simple calculations. Chapter 5 provides further details of VLSI implementations of the new design using deeply pipelined logic.

Chapter 6 presents a synthesis study of how the performance of LDPC decoders vary with operand precision. We look at the throughput, area, and energy efficiency of both bit-serial and digit-online decoders for single-frame and frame-interlaced decoding. We show that throughput and area are much less dependent on precision in digit-online decoders than in bit-parallel decoders.

The effect of code length on post-layout performance is examined in Chapter 7. We also present post-layout results for a 10GBASE-T LDPC decoder ASIC with a throughputs of 82.8 Gbit/s at 4-bit message precision and 41.8 Gbit/s at 10-bit message precision. These results are compared to other published ASICS for the 10GBASE-T LDPC decoder code. We show that fully parallel digit-online decoders are practical for code lengths of at least 2048.

Chapter 8 concludes the thesis with a summary of new contributions and suggestions for future improvements to the architecture.

Chapter 2

Low-Density Parity-Check Codes

2.1 Introduction

Forward error correction involves sending redundant data in the transmission so that the receiver can detect and repair a limited number of errors. The ability to recognize and repair errors at the receiver is especially important because it avoids retransmission, which is often expensive or impractical.

$$C = B\log_2\left(1 + \frac{S}{N}\right) \tag{2.1}$$

Shannon's Capacity theorem for additive white Gaussian noise (AWGN) channels (2.1) gives the maximum theoretical capacity C of a noisy channel of bandwidth B, given the signal power S and noise power N [1]. The ratio S/N is frequently referred to as the signal-to-noise ratio (SNR), and it is used to make relative comparisons of channel qualities. It follows directly that, for a given noise, increasing the maximum capacity of a channel requires either more bandwidth or more transmission power. Practical communication systems must operate within bandwidth and power limitations. These limitations may either be imposed by practical considerations (such as the operating frequency of electronics and battery life), or by regulatory bodies (such as the CRTC and the FCC). Issues such as interference may also require transmission power to be limited. Within these limitations, it is desirable to transmit information at the maximum possible information rate while maintaining a low error rate. The quality of the coding used determines how closely the Shannon limit is approached.

Better codes generally yield a higher effective channel capacity, subject to the law of diminishing returns due to (2.1). However, better codes are usually more complex to process. For a code to be practical, it must be possible to process it at a high rate (throughput), with low processing delay (latency), and with reasonable processing power. This is especially vital in low-power applications where using more power for coding leaves less power for transmission (reducing the maximum attainable capacity). Thus, increasing the capacity of a communications system not only requires finding more powerful codes, but also finding ways to efficiently implement them.

Low-density parity-check (LDPC) codes are popular because they approach the Shannon channel capacity while remaining relatively simple to decode. This chapter provides a brief introduction to LDPC codes, and gives an overview of current decoder implementations. It also describes limitations of present systems that motivate continued work in the field.

2.2 LDPC Codes

LDPC codes are a class of forward error correcting codes proposed by Gallager [2]. They are popular due to their high performance and the simple, parallel structure of their decoders.

In this thesis we consider only block codes, which act on a series of discrete pieces of information, all of equal length. Codes with longer block lengths require more computational power to decode, but generally result in better error performance. Discussion here is limited to the case where the data symbols are single bits, but codes that act on larger symbols also exist.

LDPC codewords consist of K information bits, and $N-K$ check bits (where N is the code length). The check bits represent constraints on the message bits. If these constraints are not met, the codeword must contain one or more errors. By using the relationships between the message bits and check bits, it is possible to reconstruct the intended message if the number of errors is small enough.

Fig. 2.1 shows an example of the $\boldsymbol{H}$ matrix representation of a code. A code of length N with M check bits has a code matrix $\boldsymbol{H}_{N,M}$ such that $\boldsymbol{H}\boldsymbol{x}^{\boldsymbol{T}} = \mathbf{0}$ over $GF(2)$ for any valid codeword $\boldsymbol{x}^{\boldsymbol{T}}$. The problem of decoding can be stated as follows: given a received block of data, we wish to find the codeword that was most likely to have been transmitted. If maximum-likelihood decoding is used to attempt to simultaneously satisfy all constraints, the algorithm required to find the optimal solution is NP-hard [3].

Codes can also be represented as a Tanner graph, as in Fig. 2.2 [4]. A Tanner graph is a bipartite graph made up of two types of nodes: variable nodes and check nodes. Each

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

Figure 2.1: *Example $\boldsymbol{H}$ matrix representation of a (3,6) LDPC code with $N = 16$ and $M = 8$.*

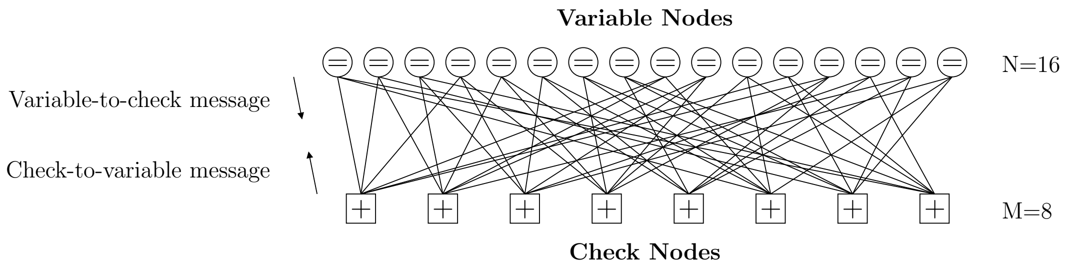


Figure 2.2: *Graphical representation of the (3,6) LDPC code from Fig. 2.1, using $N = 16$ variable nodes ("=") and $M = 8$ parity-check nodes ("+").*

type of node has a simple constraint, which allows us to split the complicated matrix-vector multiplication constraint into a set of much simpler constraints. Later we show how this allows us to use simpler decoding algorithms to find the approximate solution. Decoders using these approximate solutions can approach the error rate of the optimal decoder.

Each edge in the graph represents information about the codeword. For a variable node, the constraint is that all edges connected to a given node must be equal. They are represented by nodes with the equals sign. For a check node, the constraint is that the modulo 2 sum (or XOR) of all the edges must be zero. Put another way, the value of any edge must be equal to the XOR of every other edge connected to the same check node. Check nodes are represented in the graph by addition signs.

The Tanner graph and the $\boldsymbol{H}$ matrix are closely related. Given a set of N variable nodes and M check nodes, variable node i is connected to check node j if the entry $\boldsymbol{H}_{j,i}$ contains a one.

2.2.1 Classes of LDPC Codes

Variable nodes and check nodes are characterized in terms of their degree, or the number of connections each node has. For a regular (d_v, d_c) code, each variable node has degree d_v and each check node has degree d_c. The $\boldsymbol{H}$ matrix of a regular LDPC code contains d_v ones in each column and d_c ones in each row. Groups, or ensembles, of codes are formed from all codes having the same variable and check node degrees. The code in Figs. 2.1 and 2.2 is one possible $(3,6)$ code with $N = 16$.

Irregular codes have nodes of varying degrees [5–7]. Such a structure allows them to come arbitrarily close to channel capacity, with published codes coming as close as 0.0045 dB to the Shannon limit [7]. They can be characterized by their degree distribution, or the fraction of nodes that has a given degree [8]. Ensembles of irregular codes contain all codes having the same degree distribution.

Particular instances of codes are further characterized in terms of the number of variable nodes N, also known as code length or block size, and the number of check nodes M. Longer codes generally provide more powerful error correction, thereby more closely approaching the Shannon limit. However, longer codes require more powerful processing to encode and decode. As a result, there are practical limits on the code size that can be used.

If there are no linearly dependent check equations (i.e., $\boldsymbol{H}$ is of full rank), there are M constrained bits and $K = N - M$ bits which can transmit useful information. Any set of independent bits may be used to send information. If there are linearly dependent check equations, then $K > N - M$. The rate of a code is the fraction of the data stream which is intended to convey information, or K/M. With no linearly dependent check equations, the code rate is $(N-M)/N$. It follows that for regular codes with no redundant check equations, rate can also be expressed in terms of node degrees as $1 - d_v/d_c$. A common notations for instances of codes is (N,K). The code in Fig. 2.2 would therefore be referred to as a $(16,8)$ code.

2.3 Iterative LDPC Decoding Algorithms

Given the difficulty in implementing a maximum-likelihood algorithm, current decoders make use of iterative message-passing algorithms to approximately decode LDPC codewords. Each undirected edge in the Tanner graph can be thought of as the combination of two directed edges: one sending messages from the variable node to the check node, and one sending messages from the check node back to the variable node.

The variable nodes are initialized directly with incoming information from the channel. Through a series of iterations, the variable and check nodes exchange messages until a set number of iterations has been completed. Decoders may also stop once all parity constraints are met (known as early termination). Early termination results in lower energy per decoded bit and higher throughput since the decoder only runs as many iterations as are needed.

Gallager originally proposed two message-passing algorithms: one that used hard decisions, and one that used soft decisions [2]. Soft decisions are nearly exclusively used today due to their better error performance, but they are more easily understood if the hard decision case is first understood.

2.3.1 Hard Decoding

Using hard decisions, the receiver uses a simple threshold to decide whether each incoming bit is more likely to be a one or a zero. The result of each decision is put in the appropriate variable node. Each decoding iteration consists of two parts: one where the variable nodes are evaluating, and one where the check nodes are evaluating. Each node passes a message corresponding to the value it believes an edge should have. During the variable node update phase of each iteration, each variable node receives messages from the check nodes and calculates messages to send back. For each variable node edge, the outgoing message is what the majority of other edges believes the bit value to be. This is an attempt to satisfy the equality condition at each variable node by trying to make all edges equal. During the check node update, each outgoing edge is given the value of the XOR of all other incoming edges. This attempts to satisfy the parity check conditions.

2.3.2 Sum-Product Decoding

Though hard decisions are conceptually and computationally simple, they ignore the fact that the receiver often has exploitable information about how likely it is that each bit is correct. For example, consider a binary phase-shift keying (BPSK) modulator that encodes a zero as $+1$ and a one as -1. A hard decision would decide that any positive number received was intended to be a zero, and any negative number should be a one. However, a received value of 0.9 is much more likely to have been a zero than a value of 0.1. Soft decoding exploits this extra information for better coding gain. Rather than passing messages that contain only whether the bit is believed to be a one or a zero, the messages represent the probability that each bit is a one or a zero. The vector (p_0, p_1) represents the probability p_0 that the bit is a zero, and the probability p_1 that the bit is a one.

It is simple to extend the node constraints to deal with probabilities. For a degree-three variable node with edges A, B and C, the probability that an edge is one is equal to the probability that other two edges are both one. The probability that an edge is zero is equal to the probability that the other two edges are both zero. Hence, each edge should output the product of the incoming probabilities on the other edges. The product is normalized to retain $p_0 + p_1 = 1$. The variable node probability equations are given in (2.2) and (2.3).

$$p_{0A} = \frac{p_{0B}p_{0C}}{p_{0B}p_{0C} + p_{1B}p_{1C}} \tag{2.2}$$

$$p_{1A} = \frac{p_{1B}p_{1C}}{p_{0B}p_{0C} + p_{1B}p_{1C}} \tag{2.3}$$

For a degree-three check node, the 2-input XOR function can be used to derive the probability function. Edge A is a one if edge B is zero and edge C is one, or if edge B is one and edge C is zero. Edge A is zero if edge B and C are both zero, or if edge B and C are both one. This yields equations (2.4) and (2.5) for the check nodes.

$$p_{0A} = p_{0B}p_{0C} + p_{1B}p_{1C} \tag{2.4}$$

$$p_{1A} = p_{0B}p_{1C} + p_{1B}p_{0C} \tag{2.5}$$

The function for nodes of larger degrees can be derived by splitting them into several cascaded degree-three nodes. Since $p_0 + p_1 = 1$, there is only one degree of freedom and we can represent messages by a single value. Typically the Log-Likelihood Ratio (*LLR*) is used (equation (2.6)). It is simply a measure of relative probability on a logarithmic scale.

$$\lambda(p_0, p_1) = \ln(p_0/p_1) \tag{2.6}$$

From equations (2.2-2.6), the Sum-Product node update equations (2.7-2.9) can be derived [9–11]. The degree-d_v variable node update equation is shown in equation (2.7). λ_i represents the ith output of the variable node based on the inputs from the check nodes L_j (for $j \neq i$). $L_{channel}$ is the channel information. To begin decoding, all outputs of a variable node are initially set to the channel LLR value. When decoding is finished, the channel information at each variable node is added to all incoming LLRs to generate a final LLR,

λ_{final}, which is used to determine the decoded bit value. This is shown in equation (2.8). For BPSK modulation, the decoded bit is a 1 if $\lambda_{final} < 0$, and a 0 if $\lambda_{final} >= 0$.

$$\lambda_i = L_{channel} + \sum_{\substack{j=1 \\ j \neq i}}^{d_v} L_j \tag{2.7}$$

$$\lambda_{final} = L_{channel} + \sum_{j=1}^{d_v} L_j \tag{2.8}$$

Equation (2.9) shows the Sum-Product check node update equation. L_k represents the kth output of the check node, λ_l represents the lth input, and d_c represents the node degree.

$$L_k = 2\tanh^{-1}\left(\prod_{\substack{l=1 \\ l \neq k}}^{d_c} \tanh\left(\frac{\lambda_l}{2}\right)\right) \tag{2.9}$$

It is apparent that although the variable node message values can be easily calculated, the transcendental functions in the check node equation make it computationally expensive to evaluate the check node messages.

Sum-product decoding is optimal as long as each node is processing information from uncorrelated sources [4]. This is true in cycle-free graphs [3], or where the number of decoding iterations before convergence is less than half of the minimum cycle length of the graph [12]. In practice, sum-product decoding is near-optimal on cyclic graphs so long as the girth is kept large. The use of finite precision calculations introduces quantization noise which can degrade performance [13]. Some implementations use non-uniform quantization methods to limit the maximum quantization noise [13, 14].

2.3.3 Min-Sum Decoding

Since the sum-product method is computationally expensive, a simple approximation called the min-sum algorithm is often used. For a three-input checknode, equation (2.9) can be re-written as (2.10), where $sign(x)$ is as defined in (2.11) [15].

$$L_2 = sign(\lambda_0)sign(\lambda_1)\left(min(|\lambda_0|, |\lambda_1|) + ln(1+e^{-(|\lambda_0|+|\lambda_1|)}) - ln(1+e^{-||\lambda_0|+|\lambda_1||})\right) \tag{2.10}$$

$$sign(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases} \tag{2.11}$$

By neglecting the log terms, it is easy to generalize the equation into an approximation for many inputs (2.12). The minimum magnitude $|\lambda_{min,k}|$ is taken over all incoming edges with $l \neq k$. If messages are represented in sign-magnitude notation, the output sign bit is the XOR of the input signs for $l \neq k$. The minimum value of the magnitude bits can be taken to give $|\lambda_{min,k}|$.

$$L_k = |\lambda_{min,k}| \prod_{\substack{l=1 \\ l \neq k}}^{d_c} \operatorname{sign}(\lambda_l) \tag{2.12}$$

Although the min-sum approximation makes parity check nodes simpler to implement, it increases the error rate relative to sum-product decoding. Depending on the code rate, it results in a coding loss of 0.27-1.03 dB, with the higher losses occurring at lower rates [16]. Correction factors can be used to keep the simplicity of min-sum decoding while regaining most of the coding loss [8, 17]. The min-sum algorithm produces check node LLRs that have the same sign but a larger magnitude than sum-product [17]. Therefore, min-sum correction is done by reducing the check node magnitudes either by division or subtraction.

The normalized min-sum algorithm divides each check node LLR by a normalization factor $\alpha > 1$ as shown in equation (2.13).

$$L_k = \frac{|\lambda_{min,k}|}{\alpha} \prod_{\substack{l=1 \\ l \neq k}}^{d_c} \operatorname{sign}(\lambda_l) \tag{2.13}$$

Offset min-sum decoding subtracts a constant correction factor β from the minimum magnitude in the check node [17]. If the minimum magnitude is less than the offset, zero is output to prevent flipping the sign. Equation (2.14) summarizes the operation of the offset min-sum check node.

$$L_k = \max\left(|\lambda_{min,k}| - \beta, 0\right) \prod_{\substack{l=1 \\ l \neq k}}^{d_c} \operatorname{sign}(\lambda_l) \tag{2.14}$$

The one-step and two-step degree-matched check node equations are presented in [8]. The one-step degree-matched approximation, shown in equation (2.15), involves subtracting a constant value if the minimum magnitude of the check node inputs is greater than a certain threshold. The two-step degree-matched method is considerably more complex, requiring, among other things, the computation of more minimum values.

$$L_k = \begin{cases} \left(|\lambda_{min,k}| - \dfrac{\ln(d_c - 1)}{4} \right) \displaystyle\prod_{\substack{l=1 \\ l \neq k}}^{d_c} \operatorname{sign}(\lambda_l), & \text{if } |\lambda_{min,k}| \geq \dfrac{3\ln(d_c - 1)}{8} \\ |\lambda_{min,k}| \displaystyle\prod_{\substack{l=1 \\ l \neq k}}^{d_c} \operatorname{sign}(\lambda_l), & \text{otherwise} \end{cases} \tag{2.15}$$

The error results in [17] show that normalized min-sum and offset min-sum have similar error performance and, with the optimal correction factors, can perform within 0.1 dB of sum-product decoding. In [18] it is shown that offset min-sum outperforms sum-product decoding for the regular (2048,1723) 10GBASE-T LDPC code. A discussion in [8] shows that for regular codes there is little error performance difference between any of the four corrections mentioned. For irregular codes, there is advantage to the one-step and two-step degree-matched corrections since they apply different correction factors at check nodes of different degrees.

A further approximation to the min-sum algorithm involves replacing the extrinsic minimum function with a global minimum [19]. This simplification decreases the complexity of the check nodes by only requiring one minimum value to be found at each check node. An additional correction term may be applied to limit the performance loss [19].

2.4 LDPC Decoder Implementations

Fully parallel decoders are implemented by directly instantiating the Tanner graph in hardware. That is, each node in the graph corresponds directly to a physical component in hardware. Each edge in the graph corresponds to interconnect wiring between components. Such an approach is advantageous because it allows for largely parallel computation. Massively parallel computation is necessary to achieve the high throughput demanded by high performance applications [20]. Table 2.1 shows some current and proposed standards incorporating LDPC codes. Standards for 40 and 100 Gbit/s Ethernet have been approved using the same LDPC matrix as the 10GBASE-T code, continuing to raise the throughput targets for LDPC decoders. Likewise, the WirelessMAN standard has increased the throughput demands to over 1 Gbit/s from WiMAX's 100 Mbit/s.

Because of the large number of graph edges for larger codes, and the random nature of LDPC matrices, wiring overhead can consume a significant area of a realized system [12,18,21,22]. This is compounded in digital systems where each message is represented by

Standard	Code Length	Code Rate	Throughput Requirement
10GBASE-T	2048 bits	84%	6.67 Gbit/s
40 Gb/s Ethernet	2048 bits	84%	27 Gbit/s
100 Gb/s Ethernet	2048 bits	84%	67 Gbit/s
WiMAX	576-2304 bits	1/2, 2/3, 3/4, 5/6	100 Mbit/s
WirelessMAN	576-2304 bits	1/2, 2/3, 3/4, 5/6	1+ Gbit/s

Table 2.1: *Current and proposed standards incorporating LDPC codes*

several bits, so each graph connection may be realized with multiple wires to pass all bits of the message in parallel (known as bit-parallel message-passing). It is reported in both [18] and [22] that as the parallelism of a bit-parallel message-passing decoder increases, the wiring overhead increases. Thus the goal of a high decoding throughput is at odds with the goal of a high logic density. In [12] it was reported that the size of a fully parallel decoder was determined by wiring congestion and not logic area. Besides the obvious cost of the inefficient use of silicon area, wiring delays become a limiting factor in operating speeds [20].

These limitations have lead to the design of structured codes that are suitable for partially parallel decoding [14, 18, 22–24]. The variable nodes and check nodes are grouped in a way that a subset of them can be updated per cycle rather than all of them. This requires the $\boldsymbol{H}$ matrix to be made up of square sub-matrices that have only a single one in any given row or column [25]. In other words, each sub-matrix is a permutation of an identity matrix. In many cases this permutation is as simple as right-shifting all rows *mod* the width of the sub-matrix [26]. Partially parallel decoders are smaller than fully parallel decoders, but have correspondingly lower throughputs [25]. Since they have fewer computation elements than nodes in the code, the control logic is more complex than fully parallel decoders. For a given throughput, higher parallelism results in a better power efficiency because the more parallel design can operate at a lower supply voltage [27].

Other approaches to limit the number of wires, such as message broadcasting [21] and non-uniform quantization schemes [13,14], are applicable to both partially and fully parallel decoders. Using serial communication to pass messages in a bit-serial manner has also shown promise as a method to reduce the number of interconnect wires in LDPC decoders [19, 27, 28]. We discuss this in detail in Section 2.4.1.

There has also been work in creating new decoding algorithms that have reduced routing congestion at the cost of coding gain. Stochastic decoding algorithms require fewer wires than bit-parallel min-sum or sum-product decoding but do not get as large of a coding gain

[29,30]. A 4-bit offset min-sum decoder outperforms a stochastic decoder by approximately 0.4 dB for the (2048,1723) 10GBASE-T LDPC code [30]. Split-Row Threshold decoding is another alternate algorithm that improves routing at the cost of 0.2-0.3 dB of coding gain [25].

2.4.1 Bit-Serial Implementations

Recently, there has been an interest in digital systems that transfer each message serially across single wires [19, 27, 28]. While this approach has reduced wiring complexity, it creates new challenges to processing messages with low latency.

If the messages are binary encoded and sent most-significant-bit (MSB) first, the check node can evaluate the minimum function bit-by-bit as the messages are being received. However, serial addition is most efficiently done LSB-first. Each sum bit can be output as the messages come in, and any carry bits can be delayed and added in to the next set of input bits.

Taken alone, both operations are easy to implement with simple hardware while providing excellent performance. In a full system, continuous processing of messages is only possible if the variable and check nodes process the bits in the same order. As a result, it becomes necessary to add additional complexity to the system to resolve this conflict. Currently, this involves deserializing messages and performing parallel operations or reversing the order of bits. As a result, some hardware is only in use for only a small portion of the time.

Adding two q-bit two's complement numbers MSB-first or finding their minimum least-significant-bit (LSB) first requires at least q cycles [31]. Thus performing MSB-first addition or LSB-first minimum on two's complement operands requires no fewer cycles than buffering the operands.

In [19] and [28], bits arriving at the variable nodes were buffered and then added in parallel. While this allowed for a higher throughput with their design, parallel adders are more complex than serial adders. If the bit order were consistent, the serial adders could be pipelined to decrease the critical path. This would allow for increased operating frequencies while keeping the hardware efficiency.

While such bit-serial systems reduce wiring complexity, they suffer from relatively poor throughput because for q-bit messages, each set of nodes takes q cycles to produce a result, during which time the other set of nodes remains idle. Each decoding iteration then takes at

	Darabiha *et al.* [27]	Onizawa *et al.* [32]	Brack *et al.* [33]	Z. Zhang *et al.* [18]		Ueng *et al.* [22]
Technology (nm)	90	90	65	65		90
Code	(2048,1723)	(1024,512)	(9600,7200)	(2048,1723)		Multiple
Degrees	(6,32)	(3,6)		(6,32)		
Parallelism	Full	Full	Partial	Partial		Partial
LLR Precision (bits)	4	4[1]	6	4		5
Max. Iterations	8		10	8		8
Die Area (mm^2)		~23.1		6.67		
Core Area (mm^2)	9.8[2]	5.01	0.5[2]	5.35		4.41
Clock Frequency (MHz)	250	400	500	700	100	303
Info. Throughput (Mb/s)	13461	1600	1088	40130	6670	4080
Operating Voltage (V)				1.2	0.7	
Power (mW)				2800	144	855
E/b (pJ/bit)				59	22	176
TAR (Mb/$s{\cdot}mm^2$)	1374	319	2158	7501	1049	925
Scaled E/b (pJ/bit)				59		48
Scaled TAR (Mb/$s{\cdot}mm^2$)	3646	848	2158	7501		3528

[1] Non-uniform quantization
[2] Synthesis result

Table 2.2: *Comparison of sub-100nm LDPC decoder ASICs. Scaled E/b and TAR values are given for estimated performance in a 65-nm process.*

least $2q$ cycles to complete. It is, however, possible to process two codewords concurrently. While the variable nodes are processing the first codeword, the check nodes are processing the second. This doubles the throughput [19, 27, 28].

It is important to note that bit-serial systems allow the system precision to be increased without increasing interconnect requirements. However, increasing the number of bits in a message increases the number of cycles required to transmit a message, decreasing throughput and increasing latency.

2.5 LDPC Decoder ASIC Comparison

There are many possible metrics that can be used to compare LDPC decoder ASICs. Since the code size and degree of parallelism greatly affect throughput, it is often normalized with respect to area. This is sometimes called the throughput/area ratio (TAR). Some authors choose to normalize throughput with respect to the number of decoding iterations as well as area. Many decoders use early-termination, so unless the average number of iterations is known, this metric cannot be applied. The energy per decoded bit (E/b) is another important metric as it compares how efficiently decoders use power. Although both power and E/b are SNR-dependent, few authors report the SNR at which power was measured. Since early-termination makes throughput SNR-dependent, failing to report the SNR at

	Darabiha *et al.* [27]		Lin *et al.* [34]	Shih *et al.* [35]	Shih *et al.* [36]	Gunnam *et al.* [37]
Technology (nm)	130		130	130	130	130
Code	(660,480)		(1200,720)	(1944,972)	Multiple	(2082,1041)
Degrees	(4,15)					(3,6)
Parallelism	Full		Partial	Partial	Partial	Full
LLR Precision (bits)	4		6	5		
Max. Iterations	15		8	8	8	15
Die Area (mm^2)	9		13.5	7.39	8.29	5.29
Core Area (mm^2)	7.3		10.24	3.88	4.45	
Clock Frequency (MHz)	300	59		111	83.3	100
Info. Throughput (Mb/s)	2440	480	3552	125	111	2300
Operating Voltage (V)	1.2	0.6	1.2	1.2	1.2	
Power (mW)	1408	72	268	76	54	
E/b (pJ/bit)	577	150	75	608	486	
TAR (Mb/$s \cdot mm^2$)	334	66	347	17	24.9	434.8[1]
Scaled E/b (pJ/bit)	36		5	38	30	
Scaled TAR (Mb/$s \cdot mm^2$)	2674		2775	258	200	3478[1]

[1] Per total area

Table 2.3: *Comparison of 130-nm LDPC decoder ASICs. Scaled E/b and TAR values are given for estimated performance in a 65-nm process.*

which testing was done can make it difficult to perform a fairly compare throughput and TAR values.

Comparison of decoder implementation is further complicated when attempting to compare across fabrication generations. Here we first compare decoders of the same CMOS feature size, and then we use first-order scaling approximations to compare decoders across generations. Where core areas are available, TAR is reported per core area rather than die area.

To make a comparison of the best LDPC decoders across fabrication generations, we use simplified scaling models. Choosing 65 nm as an arbitrary technology to scale to, we define a scaling factor $s = \lambda/65$ for each other technology, where λ is the minimum feature size. Using the scaling rules in [39], we define scaling factors for clock frequency (and hence throughput), $s_f = s$, and area, $s_a = 1/s^2$. By assuming that power dissipation is dominated by the switching capacitance in wires, power scales with $s_p = 1/s^3$. Because the scaling rules contain assumptions about how the supply voltage scales, we do not scale separately with respect to supply voltage. Results that were obtained at a lowered supply voltage are also not scaled. From these scaling factors we can see that TAR scales as s^3 and E/b scales as $1/s^4$.

	Blanksby *et al.* [12]	Lin *et al.* [34]	Sha *et al.* [38]	Brandon *et al.* [28]	C. Zhang *et al.* [14]
Technology (nm)	160	180	180	180	180
Code	(1024,512)	(1200,720)	(8192,7168)	(256,128)	(8192,7168)
Degrees			(4,32)	(3,6)	(4,32)
Parallelism	Full	Partial	Partial	Full	Partial
LLR Precision (bits)	4	6	6	4	4[1]
Max. Iterations	64	8	15	32	16
Die Area (mm^2)	52.5	25	16.8	10.7	13.1
Core Area (mm^2)	∼38.4	21.23	11.3	6.96	
Clock Frequency (MHz)	64		317	250	290
Info. Throughput (Mb/s)	500	1998	4463	250	3150
Operating Voltage (V)	1.5	1.8		1.8	1.8
Power (mW)	690	644		1260	
E/b (pJ/bit)	1380	322		7560	
TAR (Mb/$s{\cdot}mm^2$)	13	94	395	36	241[2]
Scaled E/b (pJ/bit)	38	5		129	
Scaled TAR (Mb/$s{\cdot}mm^2$)	194	1999	8386	765	3478[2]

[1] Non-uniform quantization
[2] Per total area

Table 2.4: *Comparison of 160-180-nm LDPC decoder ASICs. Scaled E/b and TAR values are given for estimated performance in a 65-nm process.*

Tables 2.2-2.4 compare the unscaled performance of some recent decoders. The estimated performance in a 65-nm process is also given. The decoders in [28] and [27] are bit-serial implementations and all others are bit-parallel.

Table 2.5 shows the scaled performance metrics of the LDPC decoders with the best TAR and E/b performance from their original process. Since [34] fabricated the same decoder design in both 130 nm and 180 nm, we can look at which metrics scale most accurately. Area, power, and E/b scale quite well. Scaling seems to underestimate the throughput and therefore TAR of decoders.

Table 2.6 shows the scaled results for the bit-serial decoders. Even after scaling, the bit-serial implementation of the (2048,1723) in [27] cannot match the throughput of the bit-parallel decoder in [18]. The E/b of the (660,480) code in [27] is promising since it is lower than that of [18], and similar to many of the other scaled E/b ratios (not shown).

Overall, the decoders in [34] are by far the most energy efficient, while the decoders in [18] and [38] have the best TARs.

Fig. 2.3 shows a scatter plot of throughput/area vs. E/b performance for the LDPC decoders we have looked at.

	Z. Zhang *et al.* [18]	Lin *et al.* [34]	Lin *et al.* [34]	Gunnam *et al.* [37]	Sha *et al.* [38]
Original Tech. (nm)	65	130	180	130	180
Code	(2048,1723)	(1200,720)	(1200,720)	(2082,1041)	(8192,7168)
Degrees	(6,32)			(3,6)	(4,32)
Parallelism	Partial	Partial	Partial	Full	Partial
LLR Precision (bits)	4	6	6		6
Max. Iterations	8	8	8	15	15
Die Area (mm^2)	6.67	3.38	3.26	1.32	2.19
Core Area (mm^2)	5.35	2.56	2.77		1.47
Clock Frequency (MHz)	700			200	878
Info. Throughput (Mb/s)	40130	7104	5533	4600	12358
Power (mW)	2800	33.5	30.33		
Scaled E/b (pJ/bit)	58.7	4.72	5.48		
Scaled TAR (Mb/$s \cdot mm^2$)	7501	2775	1999	3478[1]	8386

[1] Per total area

Table 2.5: *Comparison of bit-parallel LDPC decoder ASICs scaled to 65 nm*

2.6 Summary

This chapter has introduced LDPC codes as a class of error control code that can closely approach the Shannon channel capacity. Various message-passing algorithms have been noted.

The drawbacks of parallel message-passing have been noted, and bit-serial decoders have been shown as one approach to reduce wiring density. They have the important property of having an interconnect complexity that is independent of the precision of the numbers used in calculations. However, existing bit-serial decoders have several important limitations and have not yet met the performance bit-parallel implementations.

We continue to a discussion of digit-online computation.

	Darabiha *et al.* [27]	Darabiha *et al.* [27]	Brandon *et al.* [28]
Original Tech. (nm)	90	130	180
Code	(2048,1723)	(660,480)	(256,128)
Degrees	(6,32)	(4,32)	(3,6)
Parallelism	Full	Full	Full
LLR Precision (bits)	4	4	4
Max. Iterations	8	15	32
Die Area (mm^2)		2.25	1.41
Core Area (mm^2)	5.11[1]	1.83	0.91
Clock Frequency (MHz)	346	600	692
Info. Throughput (Mb/s)	18638	4880	692
Power (mW)		176	59
Scaled E/b (pJ/bit)		36	128
Scaled TAR (Mb/$s \cdot mm^2$)	3646	2673	765

[1] Synthesis result

Table 2.6: *Comparison of bit-serial LDPC decoder ASICs scaled to 65 nm*

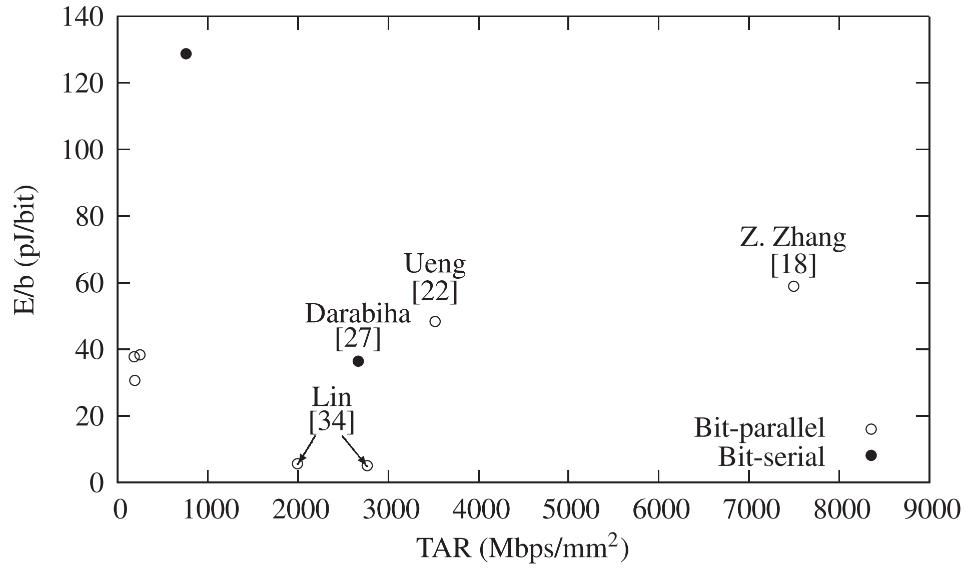


Figure 2.3: *Throughput/area vs. E/b of published LDPC block code decoders with notable results highlighted. E/b and TAR values are given for estimated performance in a 65-nm process.*

Chapter 3

MSD-First Digit-Online Arithmetic

3.1 Introduction

To build a fast LDPC decoder that uses serial message passing and serial data processing (which we refer to as a digit-online LDPC decoder), each output digit of any variable or check node should be calculated as independently as possible of the other output digits. Additionally, it is desirable that the output digits can be generated after receiving as few input digits as possible to minimize latency. These characteristics allow for a deeply pipelined system with minimum storage requirements and latency. Previous bit-serial LDPC decoders have not fully realized these pipeline goals and must buffer full-precision operands in parallel in order to reverse the bit order or perform bit-parallel arithmetic [19, 27, 28].

Our discussion of digit-online arithmetic is motivated by the operations required to implement a min-sum LDPC block code decoder. Variable nodes require the addition of inputs to implement equation (2.7). To implement equation (2.12), the check nodes must determine the sign and magnitude of inputs, select the minimum of two values (compare-select[1]), and multiply magnitudes by signs. It is important that our architecture supports correction factors to make up the error performance difference between min-sum and sum-product decoding. Therefore, division or conditional subtraction will also be required [8].

We show in Section 3.3 that LSB-first digit-serial processing is well-suited to addition, but not comparison. MSB-first bit-serial comparison can be done efficiently, but addition cannot because the carry values are not calculated until after they are needed. As we see in Sections 3.2 and 3.3, if we expand our representation from bits to generalized digits we are

[1]In this thesis we define compare-select as an operation that takes two input operands, compares them, and selects the minimum or maximum of the two.

Representation	MSD-first	LSD-first
Non-redundant (bits)	$>$	$+$
Redundant (digits)	$+,>$	$+$

Table 3.1: *Operations which are possible using MSD-first or LSD-first processing and redundant or non-redundant notation [40]. Addition and subtraction are represented by $+$, while compare-select is represented by $>$.*

able to do most-significant-digit (MSD) first addition while also being able to do MSD-first comparison. Table 3.1 summarizes these results.

An online algorithm is one that can process inputs piece-by-piece without having the full set of inputs available. In contrast, an offline algorithm has all inputs available to work with at once. Digit-online arithmetic allows operations to be performed in a serial, digit-by-digit manner [41] making it ideal for use in implementing a digit-serial LDPC decoder. By definition, a digit-online operation has a processing latency in digits that is small and fixed [31, 41].

Formally speaking, an operation is digit-online if after knowing the i most significant digits we can produce the $i-\delta$ most significant digits of the result. The initial delay δ represents the number of digits of processing latency, where δ is less than the p digits of precision of the system [31, 41]. We will refer to δ or initial delay rather than latency to avoid confusion with the other meanings of latency.

In fully pipelined digit-online computations, each operation takes in a new set of input digits per clock cycle and outputs a new set of output digits per clock cycle. Thus the latency (in time) of an operation is very closely related to the initial delay (in digits). For a single digit-online operation the total number of clock cycles required to produce a full result is $p+\delta$. This is because p cycles are required to introduce all the input digits, and because there is no output for the first $p-\delta$ cycles, so an additional δ cycles are required to finish computing output digits. Subsequent operations can be overlapped to reduce the total system delay. The total number of clock cycles required for two sequential digit-online operations having initial delays δ_A and δ_B is only $p+\delta_A+\delta_B$. This concept is illustrated in Fig. 3.1. Operations that are performed with the minimum possible initial delay may have combinational paths from input to output. If pipelining is added to split these paths, each register adds $\delta_r = 1$ to the total initial delay.

Online arithmetic usually implies that the digits are processed from most-significant-digit (MSD) to least-significant-digit (LSD). We will justify this choice by showing that

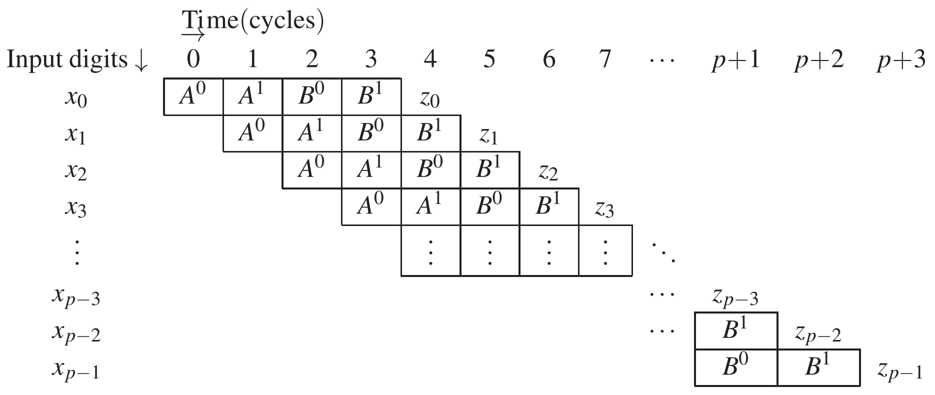


Figure 3.1: *Pipeline diagram of two successive digit-online operations A and B with initial delays $\delta_A = 2$ and $\delta_B = 2$. The pipeline stages are labeled A^0,A^1,B^0 and B^1. Input X has p digits. Output digits z_i are shown in the cycle they are produced. No output digits are produced for cycles 0 to $\delta - 1 = 3$; the p output digits z_0 to z_{p-1} are produced in cycles $\delta_A + \delta_B = 4$ to $\delta_A + \delta_B + p - 1 = p+3$.*

processing the digits from MSD to LSD is required to do compare-select in a digit-online manner. We assume that all numeric values are integers in this discussion, with the understanding that using fixed-point numbers does not require changes to any circuits for the operations we require.

As we have alluded to, some operations cannot be performed in an MSD-to-LSD manner if the operands are represented in a non-redundant number system such as two's complement binary. Section 3.2 introduces redundant number systems. Section 3.3 discusses which operations can be performed in a digit-online method. Circuits from the literature to perform these operations are examined, and two new circuits (figs. 3.7 and 3.9) are presented. Since any digit-online system will likely need to interface with systems that use conventional binary notation, we cover conversion between redundant and non-redundant forms in Section 3.4. We conclude in Section 3.5.

3.2 Redundant Notations

Positional number systems represent numbers as a string of digits. Each digit has a value and is weighted by its position. The sum of the weighted digits from the entire string is the value the number represents. For example, a p-digit unsigned base 10 number is usually represented with the digits $d_i \in \{0,1,2,3,4,5,6,7,8,9\}$ and each digit is weighted by a factor 10^i, where i is the position of the digit within the number starting with 0 at the least

$$\begin{array}{ccccccccc} & a_{p-1} & a_{p-2} & \dots & a_i & a_{i-1} & \dots & a_1 & a_0 \\ + & b_{p-1} & b_{p-2} & \dots & b_i & b_{i-1} & \dots & b_1 & b_0 \\ \hline & s_{p-1} & s_{p-2} & \dots & s_i & s_{i-1} & \dots & s_1 & s_0 \end{array}$$

Figure 3.2: *Addition of p-digit numbers A and B with result S. It is assumed that there is no carry-out from the addition so that S is also p digits.*

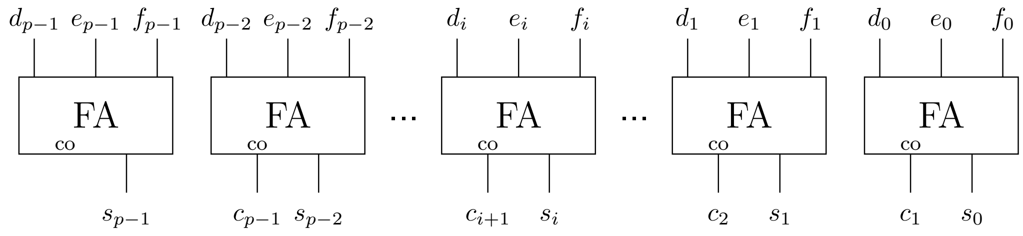


Figure 3.3: *Carry-save addition of three p-bit binary numbers D, E and F with resultant sum vector S and carry vector C.* $(D+E+F) = (S+C)$.

significant digit, and ending with $p-1$ at the most significant digit. Since no value can be represented by more than one string of digits, this is a non-redundant notation.

Using non-redundant number systems, MSD first online addition is impossible [40, 42]. Consider adding two p-digit numbers A and B (zero-padded if necessary) to get a sum S, as shown in Fig. 3.2 (for now we assume that there is no carry out from the addition so that the result is also p digits). Upper-case A represents the full precision vector, while lower-case a_i represents the i^{th} digit of the number A. At any digit position i, the value and the sum digit s_i and the carry digit to position $i+1$ depends not only on the digits a_i and b_i, but also on the carry in from the addition of $a_{i-1}+b_{i-1}$. That carry in depends on the carry out from the addition of $a_{i-2}+a_{i-2}$, and so on. The result is that the sum digit s_i cannot necessarily be determined until all digits a_i through a_0 and b_i through b_0 are known. This means that if we start by adding the MSDs a_{p-1} and b_{p-1}, we need to wait p cycles before we can produce the MSD of the output, which violates our assumption that our initial delay δ is much less than the precision p. However, using redundant number systems we can perform arithmetic where there is limited carry propagation.

Carry-save notation is a well-known redundant number system that allows the addition of binary numbers without carry propagation. Consider adding three p-bit binary integers D, E, and F in non-redundant unsigned-binary notation. We assume that overflow does not occur and that we can do not need to propagate the carry out from $d_{p-1}+e_{p-1}+f_{p-1}$. For each i, the bits d_i, e_i and f_i are input to a full adder to generate a sum bit and a carry bit as shown in Fig. 3.3. The sum bits are concatenated to form a p bit vector S, and (after ignoring the $(p-1)^{th}$ carry bit) the carry bits are concatenated to form a $(p-1)$-bit vector C'. C' is shifted left by one position to give C such that every c_i has the same weight as the corresponding s_i (c_0 is set to zero).

We now have two conventional binary numbers S and C such that $S+C$ is the result of our addition. If we wish to get a conventional binary result, we need only use a carry propagate adder to add S and C. Because we have not yet propagated the carries, the addition of any bit can be done independently of all other bits. Addition could be done in an MSD-first, LSD-first, or fully parallel manner. For parallel addition, the propagation delay is equal to the propagation delay of one full adder, regardless of the number of bits the input values have. Because of this, carry-save adders are popular to accumulate intermediate results in high-speed circuits, and only one expensive carry-propagate addition is needed to output the final result.

Since each bit x_i and y_i is in the set $\{0,1\}$, the sum at any position belongs to the set $D=\{0,1,2\}$. By selecting D as our set of digits, we can write any carry-save number as a string of single digits. The weight of any digit j is 2^j. Since the digit 2 represents a bit position with a carry-out, we now no longer have unique representations of numbers. For example, $10=02$, since $1\cdot 2^1 = 2\cdot 2^0$. Long carry chains are also possible, such as $100000=011112$. Because of this, we cannot convert from carry-save to binary notation in a digit-online fashion. Additionally, if we allow for two's complement numbers, the sign of a number in carry-save notation is not guaranteed to be known until all the digits are known.

Signed-digit representations were introduced in [42] as redundant number systems that also allow MSD-to-LSD addition of numbers. Suppose that instead of choosing a set of digits which were all positive, we allowed for negative digits. Consider the set of signed-binary digits $\{\bar{1},0,1\}$, where $\bar{1}$ represents the value -1. In the new system, the numbers $1\bar{1}=01$, since the digit $\bar{1}$ borrows from the position before it. The sign of the number is the sign of the most significant non-zero digit, since even if all other digits have the opposite sign their magnitude is not large enough to flip the sign [40,42]. As a result, there is still no upper bound on δ for sign detection since the sign remains unknown as long as the digits examined are zero. The absolute value of a number may still be calculated in a digit-online manner. If the first non-zero digit is negative, simply negate all digits. The ease of negating signed-binary numbers makes them preferable to two's complement carry-save notation, which would have a larger initial delay of $\delta=2$ for absolute value and negation operations.

Operation	LSD-First	MSD-First
Add/Subtract	Yes	Yes
Compare-select	No	Yes
Equality/Zero Detection	No	No
Multiplication	No	Yes
Division	No	Yes
Negation	Yes	Yes
Absolute Value	No	Yes
Sign Detection	No	No

Table 3.2: *Operations which can be performed on signed-digit operands in a digit-serial fashion with fixed initial delay less than the number of digits in the operands [40].*

This additional delay comes from the addition that is required to negate two's-complement numbers.

We show in Section 3.3.1 that the addition of two signed-binary numbers cannot be performed with an initial delay of less than $\delta = 2$. However, if we use the set of signed base-2^b numbers ($b > 1$) we can reduce δ to 1. Using a larger radix allows more information to be processed at once, but requires more complex logic and more communications wires. Signed-base-4 is common in the literature to allow for $\delta = 1$ addition with minimally complex hardware [40].

Signed-binary numbers require two bits-per-digit for storage. Signed-base-4 numbers require three bits-per-digit, giving them a lower relative storage overhead. However, the purpose of using serial processing in an LDPC decoder is to reduce the number of parallel wires between nodes. Signed-binary not only requires fewer wires, but less complicated logic for the arithmetic nodes. The focus of this chapter is on signed-binary systems, though signed-base-4 circuits will be discussed briefly for comparison.

In order to develop logic-level circuits, we must specify a method of encoding signed-binary digits into binary bits. The encoding specified in [43] is used. For a signed-binary digit d, each digit is encoded as two bits: d^+ and d^-. The coding is summarized by equation (3.1).

$$d = d^+ - d^- \tag{3.1}$$

3.3 Digit-Online Operations

Assuming we are using a set of signed base-2^b digits, we can now examine which operations are possible to do in an online fashion. Table 3.2 shows an overview of common operations. Addition can be done LSD-first using circuits analogous to the well-known bit-serial adder, or MSD first using the circuits explained in Section 3.3.1. Finding the minimum or maximum of two numbers cannot be done LSD-first without an initial delay equal to the precision [40], but we show MSD first comparison in Section 3.3.2. Note that zero detection or equality testing cannot be done in either direction without an initial delay equal to the precision since these operations require that all bits be tested before generating a result [40]. Multiplication and division are covered in Section 3.3.3. Multiplication must be done MSD first if the output is to have the same precision as the input operands [40], and division must be done MSD-first [43]. Negation and multiplication by $\{+1, -1\}$ can be done simply in either direction. While the absolute value of a number can be computed in an online fashion, there is no guarantee at which point the sign of the number will be known. Determining the sign and magnitude of inputs is discussed in Section 3.3.4.

We show in Section 4.3 that a digit-online min-sum check node can still be designed even though sign detection is not strictly speaking a digit-online operation. All other operations to implement a min-sum LDPC decoder can be done in a digit-online manner if MSD-first processing is used. Since division and multiplication are also possible, normalized min-sum processing can be done by implementing equation (2.13) directly, or by precomputing $1/\alpha$ and using it as a multiplicative correction factor. Offset min-sum (equation (2.14)) can be done by subtracting the offset β and then using a compare-select module to replace it with zero if the result is negative. One-step and two-step degree-matched correction, however, are not possible to do in a digit-online manner because doing subtraction conditional on a threshold cannot be digit-online [40].

We now look at the individual operations we need to implement. In describing digit-online operations, we use the convention that the output z_i is the output digit generated at the ith clock cycle, and x_j are the input digits where $j \geq i$. Thus if a circuit takes (for example) an input x_{i+2}, the initial delay δ must be at least two. Any other intermediate values given the subscript i have the same digit weight as the output.

3.3.1 Addition

We want our MSD-first adder to produce only one output digit at a time, with no carry-out. To accomplish this, addition is split into two stages (Fig. 3.4) [40, 42]. First, two digits are added to generate an intermediate sum and a carry digit. This carry digit may be positive or negative. The final stage adds the carry digit to the previous sum digit to generate one

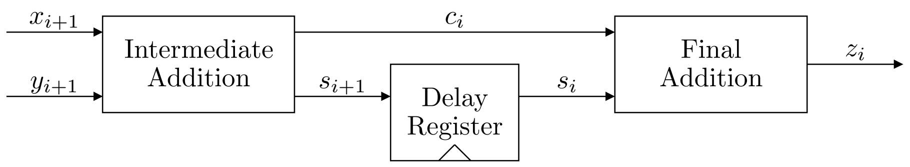


Figure 3.4: *Two-stage addition process of signed-base-4 digit-online addition $Z = X + Y$. The outputs of the intermediate addition stage are given in Table 3.3. The final addition is a simple addition of two numbers whose sum never requires a carry.*

$x_{i+1} + y_{i+1}$	c_{i+1}	s_{i+1}
-6	$\bar{1}$	$\bar{2}$
-5	$\bar{1}$	$\bar{1}$
-4	$\bar{1}$	0
-3	$\bar{1}$	1
-2	0	$\bar{2}$
-1	0	$\bar{1}$
0	0	0
1	0	1
2	0	2
3	1	$\bar{1}$
4	1	0
5	1	1
6	1	2

Table 3.3: *Outputs of the intermediate addition stage in a digit-online signed-base-4 adder from Fig. 3.4.*

output digit. By exploiting the redundant notation, we restrict the intermediate carry and sum values to ranges which guarantee that they can never cause a carry when added in the final stage.

In this way, signed-base-4 addition can be done with $\delta = 1$. The range of both intermediate values are restricted so that adding any two values in the second stage will be carry-free. If the intermediate carry is limited to the range [-1,1] and the intermediate sum is limited to the range [-2,2], it is still possible to represent all possible sums of two digits. One can easily see that there will be no second-stage carry-out since the result of any second-stage addition will be a number on [-3,3], exactly the range which can be represented in a single digit. One possible mapping (from [40]) of the intermediate sum and carry values is shown in Table 3.3.

The addition of two signed-binary numbers cannot be done with $\delta < 2$ because intermediate sum and carry digits can't be generated in a way that guarantees that the second

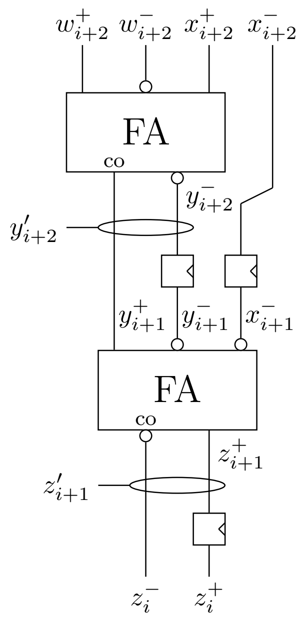


Figure 3.5: *Circuit used for digit-online addition of two signed-binary numbers with a signed-binary result [43].* $y'_{i+2} = 2{\cdot}y^+_{i+1} - y^-_{i+2}$ *and* $z'_{i+i} = z^+_{i+1} - 2{\cdot}z^-_i$ *are intermediate results. The factors of two account for the fact that the half-digits being added do not have the same weight.*

addition is carry-free [40, 42]. There are many implementations of signed-binary adders with initial delays of two digits or greater [42, 44, 45].

Fig. 3.5 shows the implementation we use for a $\delta = 2$ signed-binary adder using the desired digit encoding. Logic that breaks the carry chains between successive numbers is omitted for clarity. This logic is not complex, it simply prevents a carry out from the MSD and ensures that the carries into the LSDs are zero. A full explanation is given in Appendix A.

This addition circuit is more easily understood by considering each stage separately. Table 3.4 shows the truth table for each stage of the addition. Each stage reduces three half-digits down to one full digit. The additional flip-flop on the x^-_{i+2} input ensures that MSD of the input at the second stage arrives in the correct clock cycle. To look at the circuit in a stateless sense, on can compute the intermediate results $y'_{i+2} = 2{\cdot}y^+_{i+1} - y^-_{i+2}$ and $z'_{i+i} = z^+_{i+1} - 2{\cdot}z^-_i$. The factors of two account for the fact that the half-digits being added do not have the same weight.

By using each of the stages separately, we can introduce conventional unsigned operands into signed-binary operations. To add an unsigned-binary number in conventional notation to a signed-binary number, we use the circuit in Fig. 3.6(a). To subtract a conventional

(a)

w^+_{i+2}	w^-_{i+2}	x^+_{i+2}	w_{i+2}	x^+_{i+2}	y^+_{i+1}	y^-_{i+2}	y'_{i+2}
0	0	0	0	0	0	0	0
0	0	1	0	1	1	1	1
0	1	0	−1	0	0	1	−1
0	1	1	−1	1	0	0	0
1	0	0	1	0	1	1	1
1	0	1	1	1	1	0	2
1	1	0	0	0	0	0	0
1	1	1	0	1	1	1	0

(b)

y^+_{i+1}	y^-_{i+1}	x^-_{i+1}	y_{i+1}	x_{i+1}	z^-_i	z^+_{i+1}	z'_{i+1}
0	0	0	0	0	0	0	0
0	0	1	0	−1	1	1	−1
0	1	0	−1	0	1	1	−1
0	1	1	−1	−1	1	0	−2
1	0	0	1	0	0	1	1
1	0	1	1	−1	0	0	0
1	1	0	0	0	0	0	0
1	1	1	0	−1	1	1	−1

Table 3.4: *Truth tables for the (a) first and (b) second addition stage in the two-stage digit-online signed-binary adder shown in Fig. 3.5.* $y'_{i+2} = 2 \cdot y^+_{i+1} - y^-_{i+2}$ *and* $z'_{i+i} = z^+_{i+1} - 2 \cdot z^-_i$ *are intermediate results. The factors of two account for the fact that the half-digits being added do not have the same weight.*

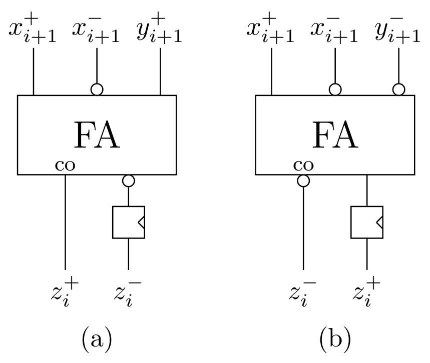


Figure 3.6: *Circuits used for (a) adding or (b) subtracting a conventional unsigned-binary operand Y to/from a signed-binary number X.*

unsigned number from a signed-binary number, we use the circuit in Fig. 3.6(b) [43]. Fig. 3.6(b) is used to implement the subtraction part of offset min-sum correction at the cost of a full-adder, a flip-flop and a few inverters. When the subtraction is done in this way, the offset does not need to be converted to or stored in signed-binary notation.

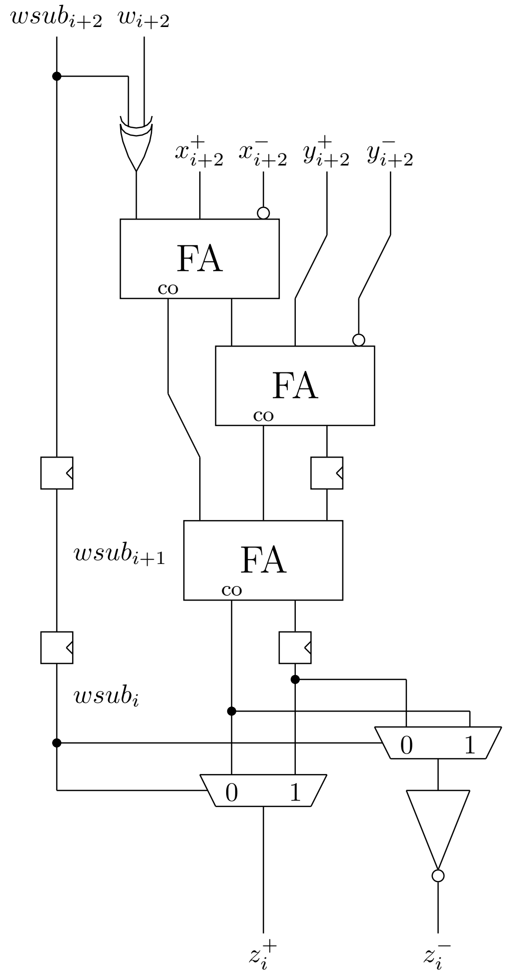


Figure 3.7: *Circuit used for digit-online addition of two signed-binary numbers X and Y and a conventional binary value W with a signed-binary result Z. Input wsub determines whether W is subtracted or added and must remain constant for the entire operation.*

The building blocks of Fig. 3.6 can be arranged to yield the circuit of Fig. 3.7 [46]. This circuit is used to add five signed-binary half digits where the sign of all is fixed at design time, or where the sign of *w* is known at its first digit. Input *wsub* is true when *w* is to be subtracted rather than added, but must remain constant over the entire operation. This higher fan-in $\delta = 2$ circuit can be used to build variable nodes with a lower initial delay. Since the signs of sign-magnitude numbers are immediately known, if the channel values are sent to the variable nodes in sign-magnitude notation, *w* can take the magnitude of the channel value and *wsub* can be set to a latched copy of the sign bit.

It is clear that addition requires much less logic if the operands are in signed-binary rather than signed-base-4. To add two signed-binary numbers requires only two full adders,

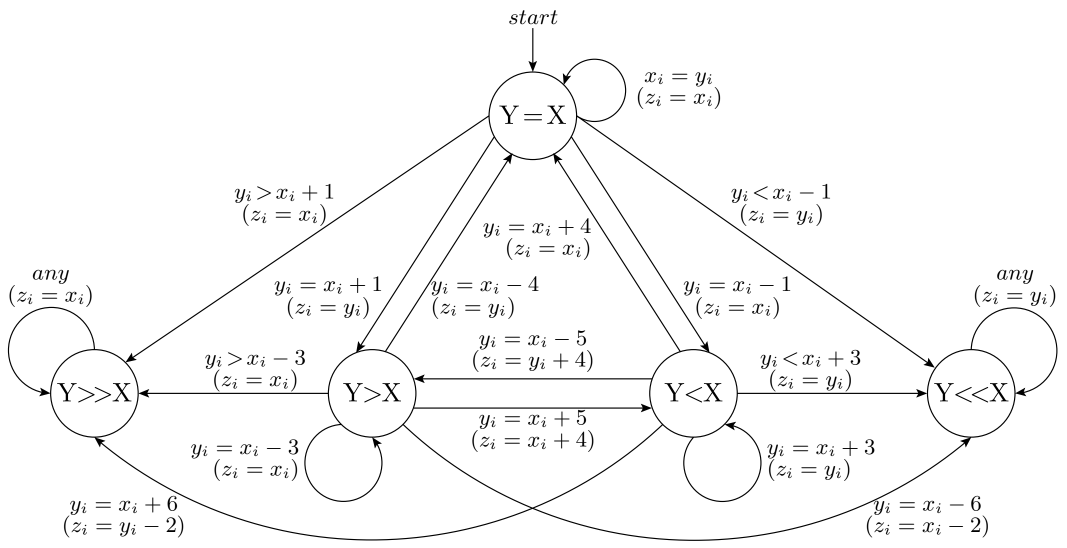

start
Y=X
$x_i = y_i$ ($z_i = x_i$)
$y_i > x_i + 1$ ($z_i = x_i$)
$y_i < x_i - 1$ ($z_i = y_i$)
$y_i = x_i + 4$ ($z_i = x_i$)
any ($z_i = x_i$)
any ($z_i = y_i$)
$y_i = x_i + 1$ ($z_i = y_i$)
$y_i = x_i - 4$ ($z_i = y_i$)
$y_i = x_i - 1$ ($z_i = x_i$)
$y_i = x_i - 5$ ($z_i = y_i + 4$)
$y_i > x_i - 3$ ($z_i = x_i$)
$y_i < x_i + 3$ ($z_i = y_i$)
Y>>X
Y>X
Y<X
Y<<X
$y_i = x_i + 5$ ($z_i = x_i + 4$)
$y_i = x_i - 3$ ($z_i = x_i$)
$y_i = x_i + 3$ ($z_i = y_i$)
$y_i = x_i + 6$ ($z_i = y_i - 2$)
$y_i = x_i - 6$ ($z_i = x_i - 2$)

Figure 3.8: $Z = \min(X, Y)$ *Mealy machine with* $\delta = 0$ *for a signed-base-4 compare-select element [40]. Inputs* x_i *and* y_i *are the input digits that arrive during cycle i, and output* z_i *is the output digit generated during cycle i. State* Y>X *means Y is larger than X up to the digits received. State* Y>>X *means Y is certainly larger than X regardless of remaining digits.*

but to add two signed-base-4 numbers requires two carry-propagate additions on three-bit inputs, for a total of six full adders. The first addition stage also requires look-up tables with 4-bit inputs.

3.3.2 Compare-Select

A basic compare-select module takes two inputs, and outputs either the minimum value, the maximum value, or both. In the case of LDPC decoding, we are largely concerned with the minimum of inputs. We wish to compare two digit-online values on the fly, and begin outputting the minimum before all the digits are known. Due to the nature of redundant notation, a number which initially appears larger (for example, has a larger MSD) is not necessarily the larger number. If a negative digit is encountered, the ordering of the numbers could swap. For example, in signed-binary, $1\bar{1}0 < 011$ even though when examined in an MSD-first fashion the first number initially appears larger.

A finite-state machine is presented in [40] which determines the minimum of two signed-base-4 numbers. In the context of a check node, we are interested in only the minimums of magnitudes (i.e., positive values). The state machine in [40] works also with

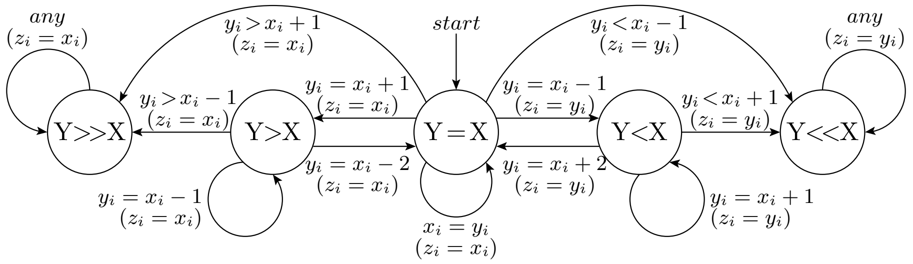


Figure 3.9: *$Z = \min(X,Y)$ Mealy machine with $\delta = 0$ used for a signed-binary compare-select element. Inputs x_i and y_i are the input digits that arrive during cycle i, and output z_i is the output digit generated during cycle i. State* Y>X *means Y is larger than X up to the digits received. State* Y>>X *means Y is certainly larger than X regardless of remaining digits.*

signed values. Since each output digit generated has the same weight as the incoming digits, $\delta = 0$.

The state diagram of this machine appears in Fig. 3.8. The difference between the two numbers is tracked for as long as the difference is small enough that either of the two numbers could still be the minimum. While this is still the case, the output of the state machine is such that the digits produced so far could be the first digits of an equivalent representation for either of the inputs. If the difference between the two numbers becomes too great, the state machine knows which is the smaller number and puts itself in a state where it outputs only the digits of that number.

The signed-base-4 compare-select module is easily adapted to signed-binary [46]. The resulting state machine is shown in Fig. 3.9. Although there are the same number of possible states, there are fewer possible transitions because of the more limited range of digit values. Additionally, the output digits are only ever selected from one of the input digits. The signed-base-4 state machine requires further arithmetic operations for some transitions (for example, from Y>X to Y<X).

The maximum calculation of offset min-sum decoding can be implemented using a much simpler state machine. Since one input is fixed at zero, the state machine can output zeros until the first non-zero digit is found. If the first non-zero digit is positive, the input is passed through unchanged. Otherwise the state machine continues to output only zeros.

3.3.3 Multiplication/Division

At a minimum, our system needs to be able to multiply by a scalar s in $\{+1, -1\}$ in order to multiply signs and magnitudes together at the output of check nodes. This can be done by negating all digits if $s = -1$ and by passing all digits unmodified if $s = +1$.

Division or more general multiplication is needed if normalized min-sum check equations are to be used. Division can be done in a digit-online manner with $\delta = 4$. By precomputing $1/\alpha$, we can use digit-online multiplication which has $\delta = 3$ for signed-binary and $\delta = 2$ for signed-base-4. However, both division and multiplication require relatively complex hardware [43]. As we show in Section 4.3, for a digit-online decoder LLR correction must be performed on every LLR individually. This requires that the correction used must be very simple to keep the hardware costs reasonable. We have already shown that offset min-sum has $\delta = 1$ and requires only a full adder, a flip-flop, and a simple state machine. Since normalized min-sum has much higher hardware cost, a larger initial delay, and a negligible error performance increase (see Section 2.3.3), offset min-sum is a much more appropriate correction method to use for digit-online decoding. Thus we can avoid implementing a general digit-online multiplier or divider.

3.3.4 Sign/Magnitude

A signed-digit number can be split into its sign and magnitude using the state machine shown in Fig. 3.10. At the beginning of each input, the machine begins in an unknown-sign state and remains there as long as incoming digits have zero magnitude. If the first non-zero digit is positive, the number must be positive and all digits from thereon can be passed through unchanged. If the first non-zero digit is negative, the number is negative and all digits must be negated to obtain the magnitude.

3.4 Conversion to and from Redundant Notations

LLR inputs are provided to the decoder in conventional binary format. Therefore, circuitry must be available to convert conventional binary numbers into the appropriate redundant format. If only the final hard-decision bits of the decoded block are required, the LLRs would not need to be converted from redundant form into two's complement format, only the sign would need to be determined.

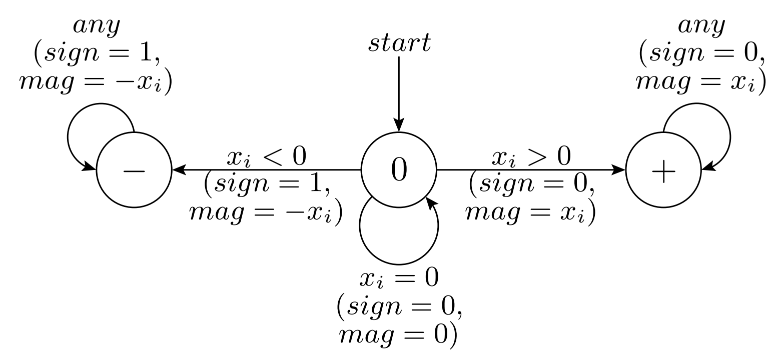


Figure 3.10: *Mealy machine used to determine the sign and magnitude (in signed-digit notation) of signed-digit numbers. The* $+$ *state means that the input is definitely positive,* $-$ *means the input is definitely negative, and* 0 *represents an as-yet-unknown sign. The magnitude is determined with* $\delta = 0$*. Sign-detection is not a digit-online operation, but* $\delta = 0$ *in the sense that the sign bit is determined in the same cycle as the first non-zero digit.*

To convert conventional binary to signed-binary, numbers can be broken down into their sign and magnitude. If a value is positive, each digit in its signed-binary representation is equal to the corresponding digit in its magnitude (though it is now stored using more bits-per-digit). For negative values, every digit in its magnitude is negated to produce the signed-binary form.

To perform a digit-online conversion of a p-bit two's complement number A into a p-digit signed-binary number B, the MSD of B is set to $\bar{1}$ if the MSB of A is one or zero otherwise. The rest of the digits of B are set to positive copies of the corresponding bits of A. This method is summarized by equation (3.2).

$$b_i = \begin{cases} -a_i, & \text{if } i = p-1 \\ a_i, & \text{otherwise} \end{cases} \tag{3.2}$$

For digit-online conversion of sign-magnitude numbers, when the sign bit is received it is stored and the MSD of the output is set to zero. For positive numbers the remaining digits are set to positive copies of their corresponding input bits. For negative numbers, each input bit is negated to arrive at the output digit. For an LDPC decoder, the channel values do not necessarily have to be converted to signed-binary. They can be stored in sign-magnitude form and introduced to the system in the variable node additions using one of the circuits in figs. 3.6 or 3.7.

Operation	δ
Add/Subtract	
signed-binary ± signed-binary	2
signed-binary ± conventional unsigned	1
Compare-select	0
Multiplication	3
Division	4
Negation	0
Absolute Value	0

Table 3.5: *Summary of initial delays for MSD-first digit-online operations on signed-binary operands*

Signed-binary numbers may be converted to two's complement values using either subtraction [42,45] or a state machine [47]. These methods are not digit-online since the carries must be fully propagated to arrive at the final result.

To use subtraction, the signed-binary number is broken into two binary numbers, corresponding to the positive and negative digits. The numbers are then subtracted in a two's complement subtraction circuit to produce a two's complement output. For example, to convert the number $001\bar{1}10\bar{1}0$ to two's complement, the subtraction of $00101000 - 00010010$ is performed. The first number corresponds to all the positive digits of our signed-binary number, the second to the negative digits.

The state machine presented in [47] takes signed-binary numbers in a serial fashion and produces a bit-parallel two's complement output. Two estimates of the two's complement value are kept: a high estimate and a low estimate. Both estimates are updated with each new incoming digit until the final result is known. For signed-base-4 numbers, [40] presents an adaptation of the method in [47] to allow conversion of numbers back to two's complement form. A method for input conversion is also presented.

Conversion from any of the redundant forms to two's complement is potentially expensive, especially at higher precisions. If possible, this conversion should be avoided by outputting only the final decoded bits off-chip. If the value of the output LLRs are needed for testing purposes, the LLRs should be output in redundant form.

3.5 Summary

Digit-online arithmetic in an MSD-first fashion is suitable for a digit-serial min-sum LDPC decoder. Signed-digit notations have been discussed. Due to the extra bit-per-digit of signed-base-4 (which would require an extra parallel wire between each arithmetic operation), and the additional complexity of the arithmetic operations, signed-binary notation is a more appropriate choice for a digit-online LDPC decoder. Table 3.5 summarizes the initial delays of the operations that have been discussed.

Given the constraints of a digit-online system, offset min-sum is the most appropriate correction method to use in the check nodes and should have a small overall hardware cost to implement. Using signed-binary, all of the required operations for offset min-sum processing can be performed in a digit-online manner with none of them needing $\delta > 2$. This is a promising result for developing a system with a low overall latency. Converting incoming conventional binary LLRs to redundant form is a relatively inexpensive operation. In fact, this operation can be avoided by using circuits that allow them to be directly added to signed-binary numbers. While converting back to conventional binary form is potentially expensive, such a conversion is not needed to determine the decoded codeword. Even for testing and debugging purposes, this conversion can be avoided.

In the next chapter we will demonstrate the construction of a digit-online LDPC decoder using the operations demonstrated in this section.

Chapter 4

Digit-Online LDPC Decoding

4.1 System-Level Design

In this chapter we assemble an architectural design for a digit-online LDPC decoder. Fig. 4.1 shows the overall architecture of our system. Messages flow from variable nodes through check nodes and back to variable nodes. The number of pipeline stages in one decoder iteration is equal to the length of the variable node pipeline (δ_v) plus the length of the check node pipeline (δ_c). Since one digit is stored in each pipeline stage, the maximum number of digits in a message $l = \delta_v + \delta_c$. However, some of these digits are used as guard digits to prevent overflow in the variable nodes (the exact number required is explained in Section 4.2). This allows LLRs of maximum magnitude to be added in the variable nodes without ill effect. The LLRs leaving variable nodes are allowed to exceed the maximum magnitude, but LLRs are saturated before reaching the variable nodes again.

Saturation may take place either immediately after the output of the variable nodes, or at any point within the check nodes. Saturation in the check node gives a clear area advantage because it requires only two magnitudes to be saturated (minimum and second minimum), rather than saturating every LLR individually.

It is desirable that the channel input LLRs be stored in conventional binary notation rather than signed-binary notation. This not only halves the channel storage requirements but also requires fewer total input bits into the addition trees. We showed in Section 3.3.1 that sign-magnitude numbers could easily be added to signed-binary numbers. Therefore, sign-magnitude channel LLRs are loaded in parallel and then sent serially to the variable nodes where they are added directly to LLRs from the check nodes. If channel LLRs are in two's complement, they can be converted upon input and stored in non-redundant sign-magnitude binary form on-chip.

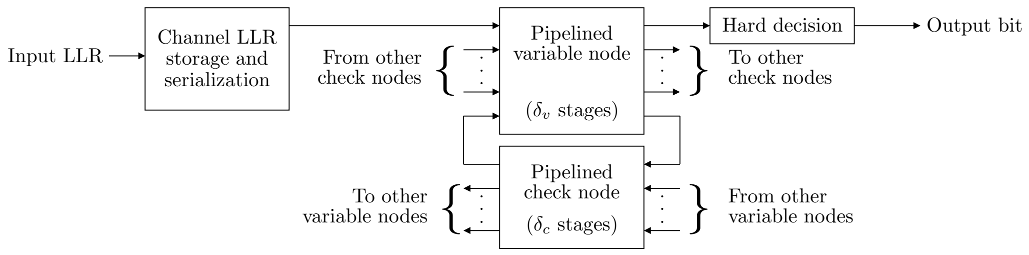


Figure 4.1: *System diagram of a digit-online LDPC decoder showing the path from one channel LLR to one output bit. Each pipeline stage stores one digit of a message.*

Each node must receive a control input that distinguishes the MSD of the LLR from other digits. We call the vector version of this signal R, and specify that for a corresponding value X, $r_i = 1$ if and only if x_i is the MSD. This allows carry chains to be broken in adders and state machines in the compare-select modules to be reset. This input is globally synchronized in the sense that each node of a given type receives its MSD at the same time.

There is flexibility in the way the R signal is distributed. It can be passed serially from node to node, or distributed as a global signal. Each node can be responsible for generating delaying versions of its R signal, or a many-phased R signal can be distributed globally.

If we expand R to be the length of the full decoding pipeline (and allow the corresponding data vectors to hold multiple values), we can decode multiple frames at a time by allowing more than one 1 in the R vector. The precision of the LLRs may be varied by changing the distance between 1s. Each frame need not be decoded with the same precision, so long as the precisions of all frames add up to the total pipeline length. While the input buffer and output detector must be aware of these parameters, the basic variable and check node circuits in no way depend on the precision or number of frames. The computation nodes work on a continuous stream of data, needing to know only where each value starts and ends. Existing bit-serial designs have an LLR length and number of frames is fixed at design time. It would not be as easy to implement variable word length in bit-serial designs because deserialization operations limit the maximum word length to half the pipeline length. The redundant notation used in digit-online decoding allows the entire pipeline to be used for one or more words.

In a decoder for an irregular code, differing node degrees may lead to different initial delays for the same type of node. Since each variable node must have the same processing delay and each check node must have the same processing delay, additional pipelining may be needed in some nodes.

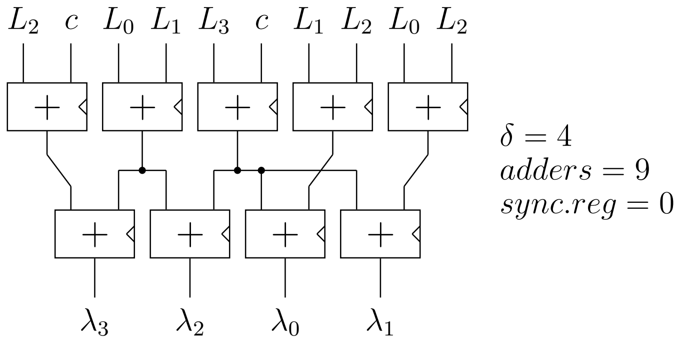

L_2 c L_0 L_1 L_3 c L_1 L_2 L_0 L_2
$\delta = 4$
$adders = 9$
$sync.reg = 0$
λ_3 λ_2 λ_0 λ_1

Figure 4.2: *Example degree-4 signed-binary digit-online variable node. The input c represents channel information.*

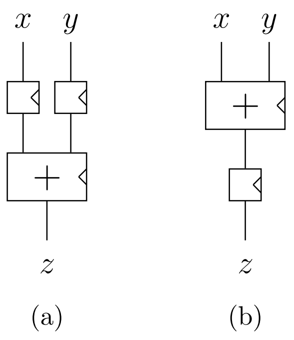


Figure 4.3: *Example of minimizing register use in addition trees. The structure in (b) is preferable to (a) because it requires one less register.*

4.2 Variable Node Design

Design of the digit-online variable node is relatively straightforward. The signed-binary adders presented earlier are used to implement equation (2.7). Each output is formed by the sum of the channel value and of all other inputs.

Care must be taken to synchronize all values moving through the node to make sure all the inputs to any adder arrive with a constant initial delay so that they have the same weight when added. Registers must be added to inputs that enter an unbalanced addition tree at later stages resulting in a higher hardware cost for these paths. Inputs should be taken from the output of the immediately previous stage when possible and, when synchronization flip-flops are unavoidable, as few paths as possible should contain synchronization flip-flops. For example, the structure in Fig. 4.3(b) is preferable to Fig. 4.3(a). If the channel LLR memory can be read with an offset, synchronizing flip-flops are not required for the channel input. Fig. 4.2 shows an example degree-4 variable node.

d_v	$\delta_{v,min}$	Guard digits	3:2 Adders	4:2 Adders	5:2 Adders	Sync. FF
2	1	1	2			0
3	2	2			3	0
4	4	2		2	4	16
5	4	3		6	5	0
6	4	3			14	0

Table 4.1: *Minimum initial delays (δ_v), number of guard digits required, and number of adders for variable node addition trees. Detailed equations are given in Appendix B.*

There are many ways of implementing the addition trees, and several possible optimization criteria. We choose to optimize for minimum δ_v and a minimum number of guard digits. This allows for the shortest critical path for a fixed pipeline length and for the least overhead due to calculating guard digits that would only be saturated out. The area of variable nodes is minimized by reusing addition sub-terms between the calculation of the d_v outputs (this is shown in fig 4.2).

The number of guard digits needed depends on the maximum variable node degree and the structure of the variable nodes. The optimal number of guard digits is $\lceil \log_2(d_{v,max}) \rceil$ to allow for the d_v values in each variable node to be added together. However, depending on the way the LLRs are added together, more guard digits could be required. For a given variable node design, the worst-case is given by the minimum δ_v, which corresponds to the most number of digits a carry can propagate forward. The actual carry propagation is dependent on input data and addition tree structure, and must be verified experimentally.

At the beginning of a new frame, the new incoming channel values can be passed directly through the variable node into the check node. Since all LLRs would be initialized to zero, doing additions is not necessary. This allows the addition tree to be used for the final sum (before hard decision) of the previous frame, resulting in more efficient hardware use. Calculating the final sum can be done with one extra addition by delaying L_0 and adding to λ_0, as shown in equation 4.1.

$$\lambda_{final} = (L_0 + \lambda_0) \tag{4.1}$$

The final LLRs are computed from the sum of $d_v + 1$ values. Since the exact value of the final LLRs is unimportant (only the sign is needed for hard decisions), this addition is treated differently to avoid requiring extra guard digits that would only be necessary for the final addition (this is covered in detail in Section 5.4).

Adder Type	Full-Adders	Flip-Flops	Delay
3:2	1	1	T_{FA}
4:2	2	3	$2T_{FA}$
5:2	3	2	$3T_{FA}$

Table 4.2: *Area and critical path delay for different types of signed-binary-output adders. Adders are classified by the number of input bits taken to produce a single-digit signed-binary (2-bit) output. The propagation delay of a full-adder is T_{FA}.*

	Bit-parallel		**Digit-online**		
d_v	Full Adders	Delay	Full Adders	Flip-flops	Delay
2	$3p$	$2p{\cdot}T_{FA}$	2	3	T_{FA}
3	$5p$	$3p{\cdot}T_{FA}$	9	8	$3T_{FA}$
4	$7p$	$3p{\cdot}T_{FA}$	16	32	$4T_{FA}$
5	$8p$	$4p{\cdot}T_{FA}$	27	30	$5T_{FA}$
6	$11p$	$4p{\cdot}T_{FA}$	42	30	$6T_{FA}$

Table 4.3: *Required resources for VLSI implementation of variable node addition trees for bit-parallel and digit-online processing on p-bit operands. The propagation delay of a full-adder is T_{FA}. Bit-parallel addition trees use the forward-backward architecture and are implemented with ripple-carry adders.*

Since we wish to use the addition tree for the final LLR of the previous frame, separate hardware must be provided to convert channel LLRs to signed-binary to generate variable node outputs for the first decoding iteration.

4.2.1 Initial Delay and Resource Requirements

Table 4.1 shows the minimum initial delays, number of guard digits required, and number of adders for the addition trees we have designed. A full description of the trees is given in Appendix B. Addition trees with the minimum possible δ_v have purely combinational paths from input to output. Further pipelining is needed to keep the clock period short.

Based on the values in table 4.1, we can compare the areas required to implement variable node addition trees using bit-parallel or digit-online processing. The digit-online requirements are computed from examining the addition circuits (see figs. 3.5-3.7) and noting that the channel sign delay registers can be shared between adders in the same register. Channel sign delay requires one additional flip-flop for $d_v = 2$, and two additional flip-flops for $3 \leq d_v \leq 6$. Table 4.2 shows the areas and critical path delays for each type of digit-online addition circuit, neglecting channel sign delay flip-flops. Adders are named according to the number of input bits taken to produce one signed-binary (2-bit) output digit. For example, the $4:2$ adder takes 4 bits of input and produces 2 bits of output.

Table 4.3 shows the areas and delays of addition trees for bit-parallel and digit-online processing. The bit-parallel resource requirements assume a forward-backward architecture where all d_v values are added together (requiring $d_v - 1$ adders) and then each input is subtracted out from that sum (requiring another d_v adders). This architecture requires $2d_v - 1$ total adders. The depth of the forward-backward addition tree is $\lceil \log_2(d_v) \rceil$ to initially add the d_v values together, and then one extra stage to subtract the original input for a total depth of $\lceil \log_2(d_v) \rceil + 1$ full adders.

It is clear that digit-online processing has a smaller combinational area and delay for even modest precisions. How the final areas compare will depend on the significance of the sequential area. Which architecture has a higher throughput will depend on how the shorter critical path of digit-online processing balances against the extra cycles it needs.

4.3 Check Node Design

The design of the offset min-sum check node is somewhat more involved. Fig. 4.4 shows the check node architecture based on equation (2.14). Input LLRs are split into signs and magnitudes using the state machine in Fig. 3.10, and offset correction is applied to each magnitude. To satisfy the extrinsic principle of min-sum decoding, we use a selection network[1] to calculate two minima: the first minimum, and the second minimum. In the case where the magnitude of two inputs are equal to the first minimum, the first minimum and second minimum are equal. The minimum and second minimum undergo saturation, and one of these corrected magnitudes is chosen for each output LLR depending on whether the corresponding LLR input was the first minimum. The second minimum is used as the magnitude for the output corresponding to the minimum input, while all other outputs are given the first minimum magnitude.

Offset correction is implemented by subtracting the offset from the magnitude unconditionally, then using a state machine to replace negative values with zero. The correction factor is stored in the decoding controller as a non-redundant binary number and broadcast to all check nodes (see Sections 5.1 and 5.5 for a more detailed explanation). Since a number in conventional binary notation is being subtracted from a signed-binary value, the adder is implemented as in Fig. 3.6(b) for a $\delta = 1$ subtraction.

[1] In this thesis we assume that selection networks produce ordered outputs, which is necessary for our LDPC decoder.

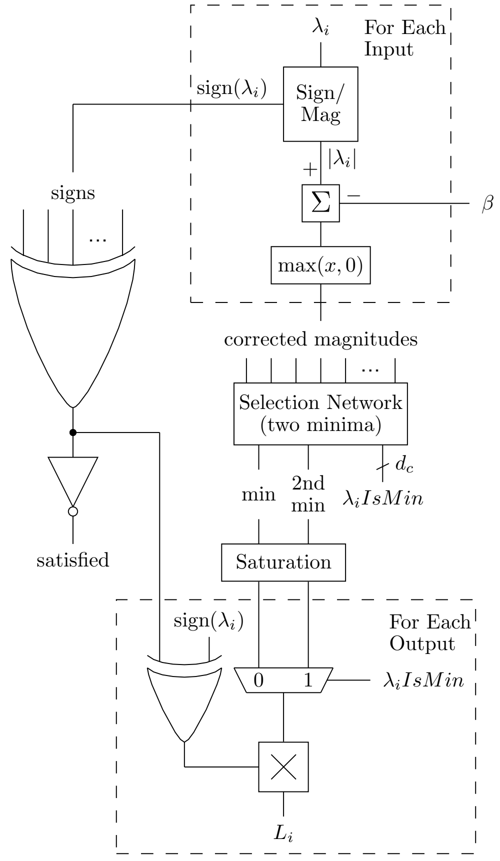
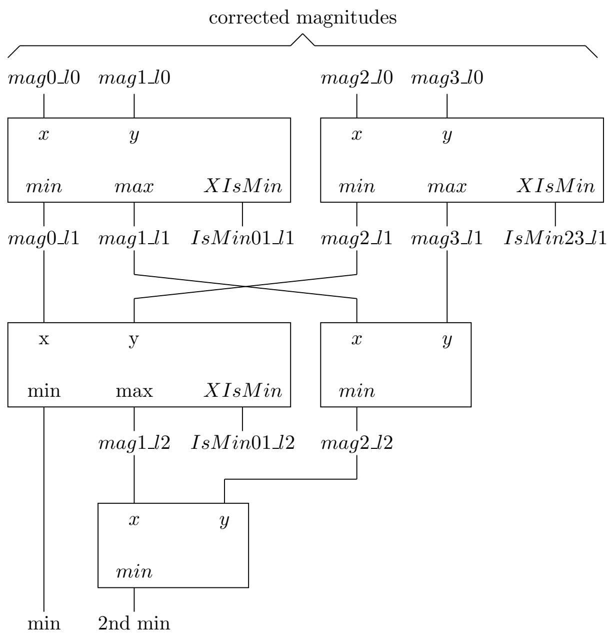


Figure 4.4: *Architecture of a signed-binary digit-online offset min-sum parity check node. The offset correction factor is represented by* β *as in equation (2.14). The sign/magnitude finite state machine is shown in Fig. 3.10. Each* $\lambda_i IsMin$ *signal is false unless* λ_i *is the input with the smallest magnitude. If two or more inputs are tied for the smallest magnitude, one is chosen arbitrarily. The selection network is detailed in Fig. 4.5.*

4.3.1 Selection Network

A selection network to find the two smallest values must be designed. Optimal (in number of comparisons or depth) sorting networks are well-known for up to 16 inputs, and heuristics are available to generate good sorting networks for larger numbers of inputs [48]. A (2,N) (two-output, N-input) selection network can be generated from an N-input sorting network by eliminating all comparators that do not lead to either of the two smallest outputs. We take advantage of the fact that there is no inherent initial delay in the compare-select operation to give us the most flexibility in pipelining the system. Compare-select modules with $\delta = 0$ are used, with the understanding that the check node can be pipelined to decrease the clock period. This flexibility allows us to best balance the pipeline stages.



$$\begin{aligned}\lambda_0 IsMin &= IsMin01\_l2 \cdot IsMin01\_l1\\ \lambda_1 IsMin &= IsMin01\_l2 \cdot \overline{IsMin01\_l1}\\ \lambda_2 IsMin &= \overline{IsMin01\_l2} \cdot IsMin23\_l1\\ \lambda_3 IsMin &= \overline{IsMin01\_l2} \cdot \overline{IsMin23\_l1}\end{aligned}$$

Figure 4.5: *Example of a selection network (built using compare-select modules) for $d_c = 4$, with explanation of how $\lambda_i IsMin$ signals are generated. The $XIsMin$ signal is true if $X <= Y$. Unused compare-select outputs are omitted.*

In order to track which input's magnitude was the first minimum, the compare-select modules are modified to mark which input had the lesser value, giving an output $XIsMin$ which is true if $X <= Y$. Setting $XIsMin$ true when $X = Y$ is arbitrary; the choice of input when $X = Y$ does not matter. The input magnitude which was the smallest can be determined by looking at the results of all the comparisons. Each magnitude $|\lambda_i|$ is marked with a signal $\lambda_i IsMin$ which is true if and only if that magnitude is the smallest. If two or more magnitudes are equal for smallest, any one can be chosen arbitrarily. Our selection network structure results in the input with the smaller index being chosen. Fig. 4.5 shows how $\lambda_i IsMin$ signals are generated for a $d_c = 4$ parity check node.

Recall from Section 3.3.2 that when comparing two signed-binary numbers, because redundant notation is used, a difference in the first digit does not imply that we know which input is smaller. Therefore, the input which first appears smallest may not be the minimum input and $XIsMin$ may change during a compare-select operation. However, to ensure correct results in the check nodes, each $\lambda_i IsMin$ signal must change only at certain times.

Specifically, the $\lambda_i IsMin$ signals may only change when the minimum and second minimum are equal up to the digits that have been received so far. This ensures that when the multiplexor switches from selecting one magnitude to another it does not produce invalid results. In the compare-select state machine, there are no transitions directly between states where $X<Y$ and states where $Y<X$. Therefore the only times the $XIsMin$ output can change is when moving between the $X=Y$ state and the states where $Y<X$ (since $XIsMin$ is true when $X=Y$). As long as the $XIsMin$ signals from all comparators remain synchronized to their corresponding minimum outputs, $\lambda_i IsMin$ transitions occur only at legal times.

This relationship between the $\lambda_i IsMin$ signals and the minimum magnitudes is the reason that offset correction must be applied to each input magnitude. Although it would require less hardware to apply offset correction at the output of the selection network (since only two magnitudes would need to be corrected), such a configuration would result in changes to the minimum magnitudes that break the relationship to the $\lambda_i IsMin$ signals and result in invalid outputs from the multiplexor. Such a design has been simulated and has been observed to generate incorrect results. It would also be possible to do offset correction between the multiplexor and the sign multiplication. However, as we will explain, we wish to perform offset correction before saturation.

The signs from the sign-magnitude splitters at the check node input are XORed together to form the output signs. Each output magnitude is multiplied by the corresponding output sign to form the final output LLR. The signs must be fully determined before the first non-zero digit is output to ensure a proper multiplication. Luckily, this is guaranteed to be the case, as illustrated in the next paragraph.

Consider determining L_2 in a degree-3 check node. The sign of L_2 need not be known at all because the output sign depends only on the sign of L_0 and L_1. The compare-select finite state machine outputs digits of the input that appears smallest. If the number with the most leading zeros is the minimum, then the sign of the maximum is known first, and both signs are known by the time non-zero digits are to be output. If the number with the fewest leading zeros is the minimum value, the state machine will still output zeros until the first non-zero digit of the maximum, at which point both signs are known.

Generalizing to larger degrees, by the time the first non-zero digit of the minimum magnitude is reaching the output multiplexer, the signs of all numbers are known. Any output LLR that does not correspond to the minimum input will output the minimum value and all signs are known by the first non-zero digit. For the output LLR that corresponds to the minimum input, the sign of the minimum does not need to be known. All other signs are known by the time the first non-zero digit of the second minimum is received.

The XOR tree for signs is implemented by XORing all inputs together, and then XORing each input sign in again to obtain the correct sign for the output. Compared to entirely independent XOR trees for each output sign, this structure reduces the number of gates and power consumption while adding only one gate delay. Since the selection network will need to be pipelined, the XOR tree may be pipelined as well to limit the effect of having a large fan-in tree. The modulo 2 sum of all input signs represents whether the check equation has been satisfied or not (sum is zero when the equation is satisfied). To allow for early termination (covered in Section 4.4), each check node outputs whether it has been satisfied or not.

4.3.2 Saturation

The ideal place to perform saturation in the check node is at the outputs of the selection network. This method requires that only 2 values be saturated rather than d_c. Another advantage of saturating in the check node is that only one comparison is necessary since all values are positive. Separately comparing for overflow and underflow is not necessary.

Numbers in signed-digit notation are difficult to saturate directly because in-range numbers can initially appear to be out-of-range [43]. Since we are not concerned with knowing whether our LLR is in range or out of range (only in clipping it), we can use a compare-select module. The LLR magnitude provides one input, and a state machine generating the maximum allowable magnitude provides the other. This ensures that all LLRs leaving the check node have magnitudes that are within the design range for the variable node inputs. The logic implementation of the saturation compare-select module can be simplified since one input is known to be positive and can be expressed in conventional binary notation. This method of saturation results in the minimum number of transitions within saturated values. If saturation is performed after offset correction, then the number of output transitions is minimized when check node values saturate.

4.3.3 Initial Delay and Resource Requirements

Since the subtraction of the offset constant is the only operation with $\delta > 0$, $\delta_{c,min} = 1$ for all check node degrees. The area of the check node is largely dependent on the number of comparisons in the selection network. For a (2,N) selection network (with ordered outputs),

N	Depth	Comparisons	N	Depth	Comparisons
2	1	1	17	7	31
3	3	3	18	7	33
4	3	5	19	7	35
5	4	7	20	7	37
6	4	9	21	7	39
7	5	11	22	7	41
8	5	13	23	7	43
9	6	15	24	7	45
10	6	17	25	8	47
11	6	19	26	8	49
12	6	21	27	8	51
13	6	23	28	8	53
14	6	25	29	8	55
15	6	27	30	8	57
16	6	29	31	8	59
			32	8	61

Table 4.4: *Depth and number of comparisons for (2,N) selection networks*

Architecture	Delay	Logic Cost
Bit-parallel ripple-carry	$O(\log pN)$	$O(pN)$
Bit-parallel tree adder	$O(\log(\log(p)N))$	$O(p\log(p)N)$
Digit-Online	$O(\log N)$	$O(N)$

Table 4.5: *Delay and Area for VLSI implementations of (2,N) selection networks on p-bit operands.*

the optimal number of comparisons is given by $2N-3$ [48]. For a $\delta_c = 1$ check node, the critical path is determined by the depth of the selection network. Table 4.4 shows the depth and number of comparisons required for (2,N) selection networks. The depth is bounded above by $2\lceil\log_2(N)\rceil$ since the smallest value can be selected in $\lceil\log_2(N)\rceil$ stages and the second smallest value can be selected in a further $\lceil\log_2(N-1)\rceil$ stages. By performing some of these comparisons in parallel we can achieve a smaller depth.

Table 4.5 compares the VLSI implementation costs for implementing (2,N) selection networks using bit-parallel or digit-online processing. The complexity of digit-online comparison circuits is independent of precision, so the delay is simply $O(\log N)$ and the logic cost is $O(N)$. For bit-parallel operations we first assume that the subtraction for the comparison is implemented as a ripple-carry adder, which is a reasonable assumption for small p. The delay and logic cost of a p-bit ripple-carry adder are both $O(p)$. Therefore, a bit-parallel selection network using ripple-carry adders has delay $O(\log pN)$ and logic cost $O(pN)$. If adders with $O(\log p)$ delay and $O(p\log p)$ area are used, the bit-parallel selection network will have delay $O(\log(\log(p)N))$ and logic cost $O(p\log(p)N)$.

4.4 Hard Decisions and Early Termination

The hard decision upon output is trivial. As the final sum is calculated, a simple state machine stores the sign of the first non-zero digit. This process can take place in the variable node to minimize wiring and activity between variable nodes and control logic. When all the signs are known, they form the bits of the decoded output and are sent off-chip.

Early termination is also easy to implement. Since each check node has an output indicating whether it has been satisfied or not, we can determine whether all check equations have been satisfied by ANDing the results from all the check nodes. If all check equations have been satisfied, the final output LLRs and the hard-decision outputs are calculated. If there are unsatisfied check equations, decoding continues as usual with the next iteration. Due to the initial delays of the final addition and the hard decisions (explained in Section 5.3), there is a cost of two extra iterations to terminate decoding early. The maximum number of iterations is never exceeded, however.

4.5 Summary

Compared to existing bit-serial implementations, the digit-online architecture has several advantages. First, the digit-online architecture does not need to store messages in parallel at any point. Bit-serial systems require parallel storage of messages in order to reverse the order of bits at each node, and in order to convert between two's complement and sign-magnitude forms [27]. Second, determining the magnitude of a signed-binary number is much simpler than determining the magnitude of a two's complement number. Current bit-serial systems rely on frame interleaving to keep all functional units busy [19, 27, 28]. Since digits always flow in the same direction in a digit-online system, all functional units are continuously in use even for systems designed to decode only a single frame.

With proper care, decoder design is possible for both regular and irregular codes. Channel information need not be converted to or stored in redundant format. If channel information is stored in sign-magnitude binary form it can be directly added to signed-binary data. The offset constant β can also be stored and distributed in conventional binary format. It is straightforward to implement early termination in order to increase throughput and decrease power consumption.

Having demonstrated a digit-online LDPC architecture, we move forward to show how to fully implement the logic.

Chapter 5

VLSI Implementation

5.1 Introduction

After looking at how digit-online LDPC decoding can be done in a general sense, we move forward to the VLSI design of a full decoding system. A top-level logic diagram of the system is shown in Fig. 5.1 and the top-level control signals are explained in table 5.1. Besides the variable nodes and check nodes, a decoding controller is needed. The controller synchronizes all nodes, loads and stores channel LLRs, outputs channel LLRs to variable nodes, tracks the number of iterations, and (after the final iteration) outputs the decoded codeword. To implement early termination, the controller also looks at whether or not the check nodes are all satisfied and terminates decoding early if they are.

Fig. 5.2 shows a timing diagram of the interface signals. The decoder is initialized by way of the $\overline{\text{reset}}$ signal. The frameInterlaceMode signal must be asserted first, as it is latched in on reset. This sets the decoder in either a single-frame or frame-interlaced mode. The decoder sets the ready signal high when it is able to begin decoding a new frame. The full-precision sign-magnitude LLRs (chanLLRsIn) and offset (offsetIn) must then be applied in parallel. The start signal is then held high until the decoder sets ready low to signal that decoding has started. Once the frame has been decoded, the decoder outputs the decoded bits (decodedOut) and asserts the valid signal to signal decoding has completed.

To allow for frame-interlaced decoding, and continuous decoding in single-frame mode, the decoder must be able to buffer two input frames. For reasons we explore in Section 5.4, we chose to distribute one of the frame buffers as local memory in the variable nodes. The other frame buffer is located in the controller and is used for parallel-to-serial conversion of channel LLRs.

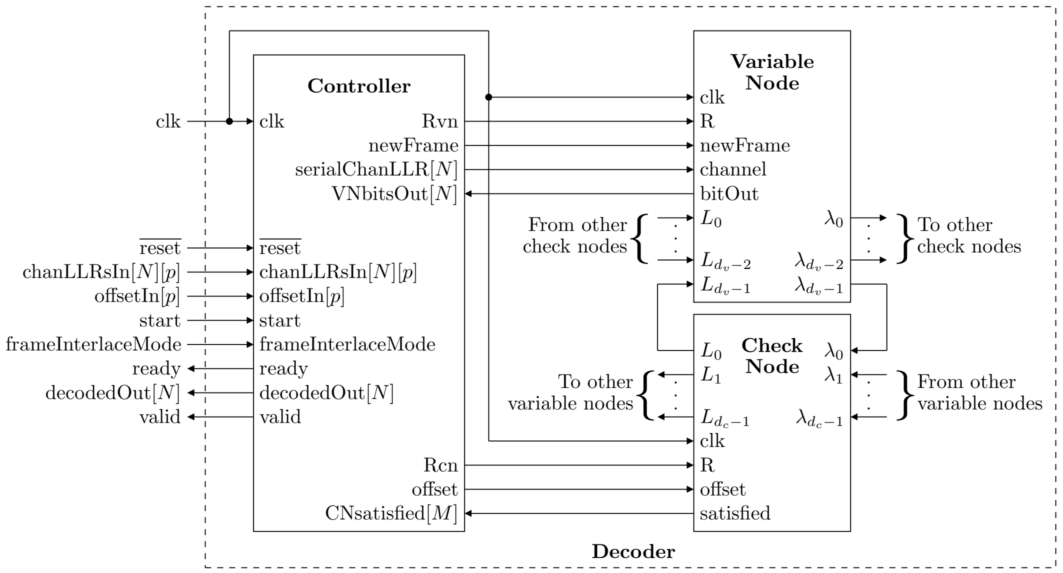


Figure 5.1: *Top-level block diagram of a digit-online LDPC decoder for a code of length N with M parity checks. Channel LLRs are quantized to p bits. For simplicity, only one of the N variable nodes and one of the M check nodes are shown. Variable-to-check node connections λ_i and check-to-variable node connections L_i are 2-bits wide to allow signed-binary digits to be passed between nodes. Controller signals are detailed in table 5.1.*

Interface with	**Signal**	**Description**
Off-chip	clk	System clock
	$\overline{\textit{reset}}$	Global reset
	chanLLRsIn[N][p]	Bit-parallel channel LLRs (sign-magnitude)
	offsetIn[p]	Bit-parallel decoding offset constant (β)
	start	Instruct the decoder to load a frame and begin decoding it
	frameInterlaceMode	Specifies single-frame (low) or frame-interlaced (high) decoding. Latched on reset.
	ready	Set high when the decoder is ready to load a new frame
	decodedOut[N]	Decoded codeword output
	valid	Set high when decodedOut is updated
Variable Nodes	Rvn	Variable node synchronization signal (breaks carry-chains, etc.)
	newFrame	Instructs the variable nodes to store new channel values and perform hard decisions for the previous frame
	serialChanLLR[N]	Serial sign-magnitude LLRs
	VNbitsOut[N]	Hard-decision bits from variable nodes
Check Nodes	Rcn	Check node synchronization signal (resets sign-magnitude state machines, etc.)
	offset	Bit-serial offset constant (β)
	CNsatisfied[M]	CNsatisfied[i] set high when check node i's check equation is satisfied

Table 5.1: *Top-level controller signals*

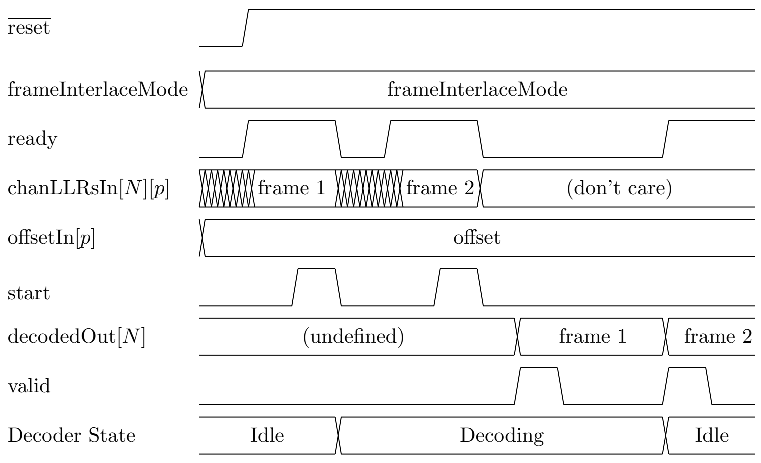


Figure 5.2: *Top-level timing diagram of a length=N digit-online LDPC decoder with p-bit channel quantization showing main control signals (time not to scale). The frameInterlace-Mode must be stable at reset at which point it is stored. When ready is high, chanLLRsIn and offset are set and start is asserted to begin decoding. The signals chanLLRsIn and offset must remain stable until ready goes low signalling that they have been read in. When decoding of a frame is complete, the decoded bits are set on decodedOut and valid is set high.*

File	**Contains**	**Specific To**		
		d_c, d_v	Code	Pipeline
Pipeline Specification	Channel quantization p, initial delays $\delta_v, \delta_{v,final}, \delta_c, \delta_{sat}$, maximum iterations	no	no	yes
Interleaver Specification	Code length N and number of parity checks M, node degrees d_c, d_v for all nodes, $\boldsymbol{H}$ matrix connections, data types	yes	yes	no
Check Node	Check node logic implementation	yes	no	yes
Variable Node	Variable node logic implementation	yes	no	yes
Controller File	Decoding control logic	no	no	no
Decoder Top-level	Top-level connections	yes	no	no

Table 5.2: *HDL Files used in digit-online LDPC decoder design*

The remainder of this chapter provides logic implementation details for the major system components. System-wide considerations are covered in 5.2. The functions of the decoding controller are explained in Section 5.3. The full variable node design is demonstrated in Section 5.4, and the remaining design of the check nodes are presented in Section 5.5.

5.2 High-Level Design

The VHDL for the decoder is designed with readability and re-usability in mind. In order to require as few new files as possible to be created for a given decoder design, each file in the design makes as few assumptions as possible about the structure of the decoder. The design attempts to abstract out as many parameters as possible into configuration files.

Table 5.2 gives a summary of the design files that are needed for a full decoder design. The interleaver specification stores the graph connections as integer arrays. In the case of an irregular code, the interleaver specification also contains arrays specifying the degree of each node in the decoder. Variable nodes and check nodes are designed for given degrees and initial delays. The variable nodes and check nodes are instantiated in the top-level file by generate statements that index the connection array. This avoids explicitly defining connections for each node in the decoder file, while still allowing the matrix to be arbitrary. In irregular codes, these generate statements also reference the node degree arrays to instantiate each node with the correct degree. Both the decoder VHDL file and the interleaver package file can be generated automatically after reading an ALIST file specifying the parity matrix. In the case of structured codes (such as the WiMAX family of codes), connections can also be generated from a submatrix specification and expansion factor.

The decoder top-level file is specific to an ensemble of codes, but different sized codes can be quickly synthesized by loading different interleaver specifications. Since the initial delays are stored in a separate file, it is easy to quickly synthesize designs for the same code while varying the number of pipeline stages.

5.3 Decoding Controller Implementation

Throughout this chapter we use the convention that all variable nodes have the same initial delay δ_v and all the check nodes have initial delay δ_c. As stated previously, in the case of irregular codes all nodes of a type must be pipelined to the same depth. This results in a total pipeline length $l = \delta_v + \delta_c$. The exact amount of pipelining is ignored at this point. Extra registers can always be added in nodes to achieve the desired loop latency (and therefore precision), and register retiming can be enabled in synthesis to balance the stages automatically. Since pipelining is relatively inexpensive in the control logic compared to the computation nodes, the control logic is pipelined deeply enough so that the critical paths are in the computation nodes.

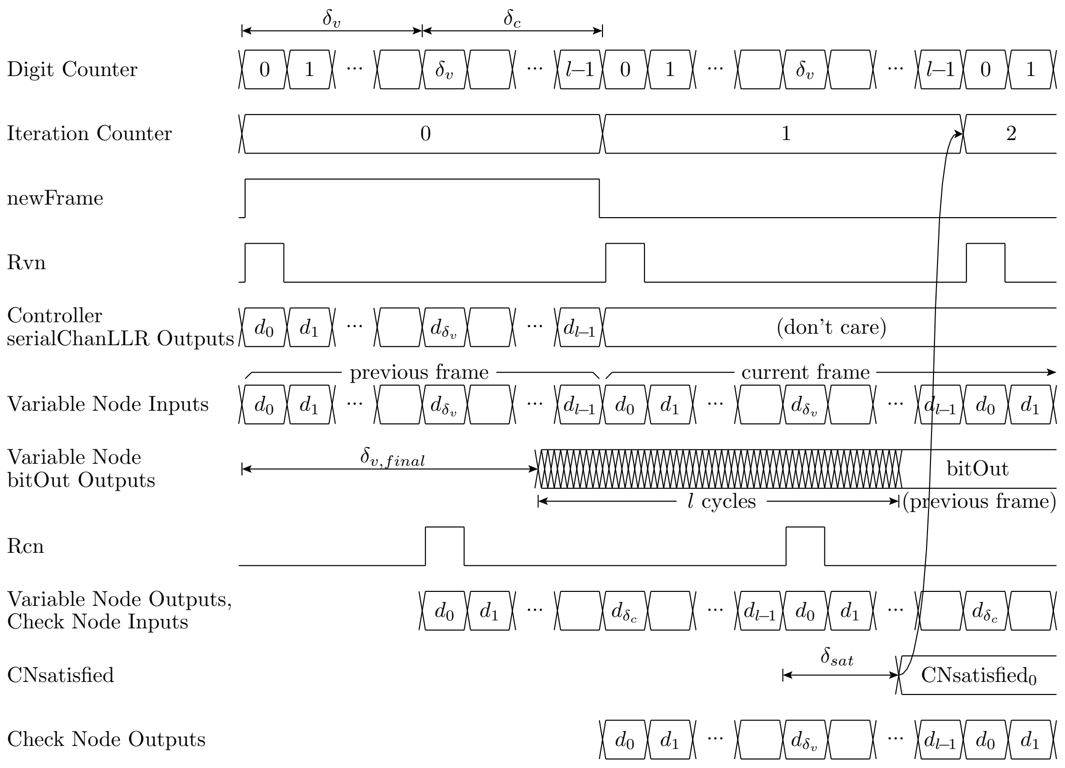


Figure 5.3: *System timing diagram for a single-frame digit-online LDPC decoder showing the location of each digit for one iteration. The pipeline length $l = \delta_v + \delta_c$, where δ_v and δ_c are the pipeline lengths of the variable nodes and check nodes, respectively. The value D with digits d_i $(0 \leq i < l)$ represents a general message in the decoding pipeline. The newFrame signal remains high for the entire first iteration to signal the variable nodes to store the new channel values and produce the decoded bits for the previous frame (if applicable). The CNsatisfied signal is updated once per iteration, δ_{sat} cycles after the checknodes receive the last input digit of the LLR. Since the delay from the start of decoding to CNsatisfied updating is $\delta_v + l + \delta_{sat}$, the decoder must perform two decoding iterations before it knows whether the input codeword was valid. This is the source of the two iteration penalty for early termination.*

Figure 5.3 shows the timing diagram for the control signals for single-frame decoding. There is a counter to track the number of iterations that have been run. The controller keeps the variable nodes and check nodes are synchronized by way of their *R* signals. When *R* is high, the current digit is the MSD of a new value. This information is used to break carry chains and reset state machines. The *R* signals received at the variable nodes and check nodes are denoted *Rvn* and *Rcn* respectively. A digit counter keeps track of which digit position is currently being output to the variable nodes and outputs $Rvn = 1$ when the variable nodes are receiving the MSDs of their messages. *Rcn* is set high after waiting a number of cycles equal to the initial delay δ_v of the variable node. Delayed versions of *Rvn* and *Rcn* needed for synchronizing internal signals in the nodes are generated locally at each node. Other than the *R* signals, reset signals to the variable and check nodes are not needed. Any unknown values that are present at power up are ignored by the decoder.

The controller loads the channel LLRs in bit-parallel fashion in non-redundant sign-magnitude binary format. They are then sent in serial to the variable node channel inputs via serialChanLLR. The newFrame signal is kept high for the entire first iteration to allow the variable nodes to load the new channel values. While newFrame is low, variable nodes must use their local storage for their channel values (this is explained in detail in Section 5.4). During the time newFrame is high, the variable nodes are also computing their final sums and performing their hard decisions for the previous frame. There is a $\delta_{v,final}$ stage delay to perform the final addition, and then the hard decisions can take up to a further l cycles depending on how many leading zeros there are in each node's final sum. Since the sign of the final sum is known for certain as soon as a non-zero digit is received, there is at most one transition during the evaluation of each variable nodes' bitOut signal.

Each check node generates a satisfied signal to denote whether or not its check equation is satisfied. This signal changes only once per iteration, $l+\delta_{sat}$ cycles after Rcn. This is to allow for the l bits of the LLRs to be known so that the signs of all inputs are known for certain, and to allow for the δ_{sat} cycle delay in the XOR tree needed to perform the parity check. Since the total delay relative to the beginning of the iteration is $\delta_v + l + \delta_{sat}$ and is greater than $\delta_v + \delta_c$, it takes two extra iterations to determine whether all checks were satisfied. If $(\delta_v + l + \delta_{sat}) < 2 \cdot (\delta_v + \delta_c)$, the extra cycles can be used to pipeline the AND tree used by the controller to determine if all checks are satisfied simultaneously.

When decoding is finished (after all check equations are satisfied, or after a specified number of iterations), the controller takes in the hard decision bits from the variable nodes and outputs them as the decoded codeword. The controller accounts for the delay the variable nodes require to make their final hard decisions, and waits until the variable node hard decision modules have received the LSDs of their final sums, at which point all decoded bits are guaranteed to be known.

After examining the timing of the decoder in detail, we can derive equation (5.1) for the throughput of the decoder for single-frame decoding, where n_{iter} is the number of decoding iterations. If early termination is enabled, the average number of iterations per frame should be used to calculate throughput, taking into account the two iteration penalty to terminate early. Equation (5.2) gives the maximum latency of the decoder.

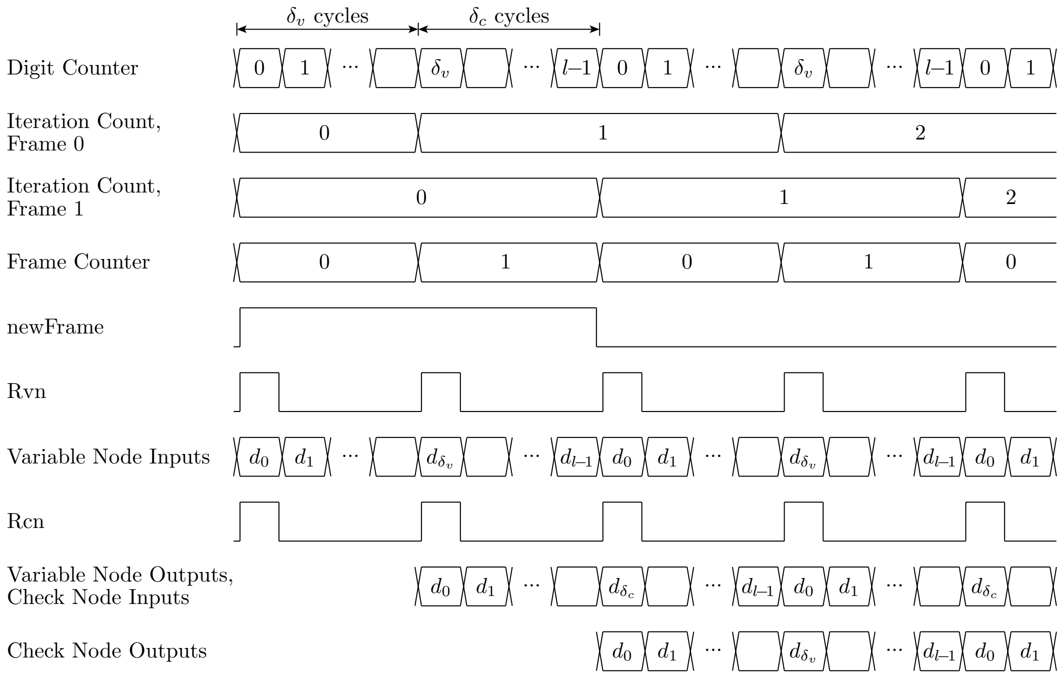


Figure 5.4: *System timing diagram for a frame-interlaced digit-online LDPC decoder showing the location of each digit for one iteration. The pipeline length $l = \delta_v + \delta_c$, where δ_v and δ_c are the pipeline lengths of the variable nodes and check nodes, respectively. The value D with digits d_i $(0 \le i < l)$ represents a general message in the decoding pipeline. The first half of the digits of D belong to frame 0, and the other half belong to frame 1. The newFrame signal remains high for the entire first iteration of each frame to signal the variable nodes to store the new channel values and produce the decoded bits for the previous frame (if applicable). Note that in general it is not necessary that $\delta_v = \delta_c$, nor is it necessary that both frames be on the same decoding iteration.*

$$Throughput_{1f} = \frac{N \cdot f_{clk}}{(cycles/iteration) \cdot n_{iter}} = \frac{N \cdot f_{clk}}{(\delta_v + \delta_c) \cdot n_{iter}} \tag{5.1}$$

$$Latency_{max} = \frac{(n_{iter,max} + 1) \cdot (\delta_v + \delta_c) + \delta_{v,final}}{f_{clk}} \tag{5.2}$$

Fig. 5.4 shows a simplified timing diagram of frame-interlaced decoding to highlight its important differences to single-frame decoding (Fig. 5.3). The controller must use a frame counter to keep track of which frames messages are currently being processed. The controller must also now keep two iteration counters, one for each frame being simultaneously decoded. The control signals *Rvn* and *Rcn* must now be set high twice during each iteration to mark the MSD of each of the two sets of message MSDs to the check and variable nodes. Since the variable node and check nodes require no knowledge of the precision

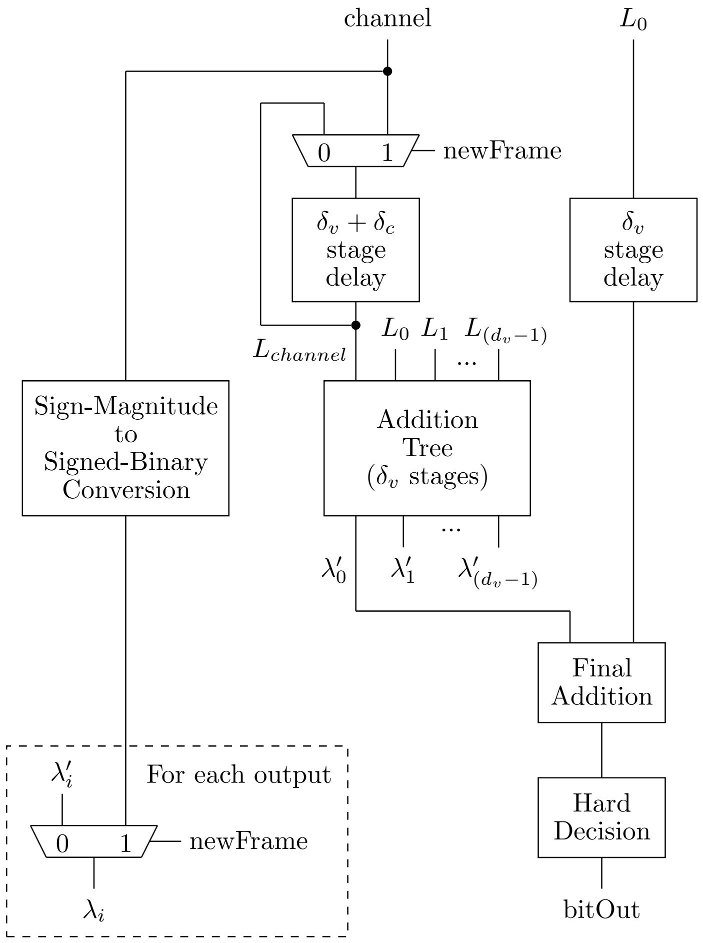

channel
L_0
0 1
newFrame
$\delta_v + \delta_c$ stage delay
δ_v stage delay
$L_{channel}$
L_0 L_1 ... $L_{(d_v-1)}$
Sign-Magnitude to Signed-Binary Conversion
Addition Tree (δ_v stages)
λ'_0 λ'_1 ... $\lambda'_{(d_v-1)}$
Final Addition
Hard Decision
bitOut
λ'_i
For each output
0 1
newFrame
λ_i

Figure 5.5: *Block diagram of a digit-online variable node. The sign-magnitude to signed-binary conversion is explained in Fig. 5.6. The final addition stage is shown in Fig. 5.8*

of their inputs (only when new values begin), they require no changes to accommodate frame-interlaced decoding.

Since the decoder can now store two messages in *l* stages, the throughput of the system doubles, as shown in equation (5.3). The maximum latency remains the same as single-frame decoding (equation (5.2)).

$$Throughput_{2f} = \frac{2 \cdot N \cdot f_{clk}}{(\delta_v + \delta_c) \cdot n_{iter}} \tag{5.3}$$

5.4 Variable Node Implementation

Fig. 5.5 shows a block diagram of the variable node architecture used. To minimize routing between control logic and variable nodes while allowing efficient use of the addition tree,

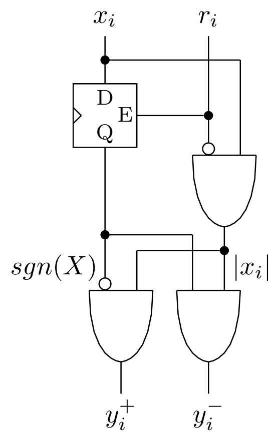


Figure 5.6: *Circuit for converting a sign-magnitude input X to a signed-binary output Y with $\delta = 0$. When r_i is high signalling that x_i is the MSD, the sign of X is stored in the flip-flop to be used throughout the conversion. The magnitude of y_i is x_i except for the MSD which has magnitude zero.*

there is local channel memory in each variable node rather than having two input buffers in the control logic. This storage is implemented as a $(\delta_v + \delta_c)$-stage delay line.

While newFrame is low, decoding is continuing from one iteration to the next of the same frame, and the outputs λ_i are set to the outputs of the addition tree λ'_i. During this time no channel values arrive from the decoding controller and the local channel memory is used instead.

The newFrame signal remains high during the entire first decoding iteration to allow the variable node to transition between successive frames. While newFrame is high, a new channel value is loaded into the local channel memory as the old channel value is shifted out to complete the final addition before hard decision. Simultaneously, the new channel value is converted to signed-binary form and output as the initial LLR values for the new frame. Performing an explicit conversion allows the addition tree to be used to compute the final sum before hard decision of the previous frame. To save power, the final sum and hard decision are computed only during the final iteration (whether because of early termination or hitting the iteration limit).

The conversion from sign-magnitude to signed-binary is shown in Fig. 5.6. When the MSB of the sign-magnitude value is received, it is stored to use as the sign throughout the conversion. The MSD of the signed-binary value is set to zero. For the remaining digits, the input bit is used as the magnitude, and the stored sign is used.

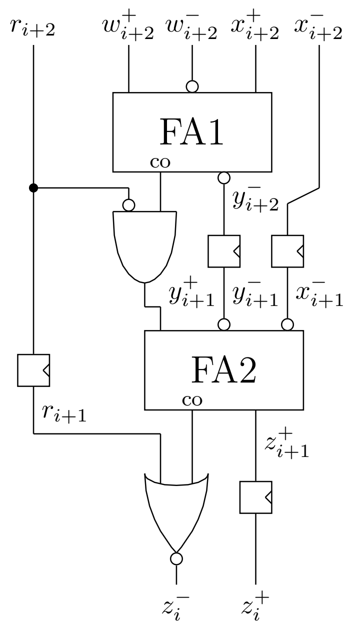


Figure 5.7: *Circuit for digit-online addition of two signed-binary numbers W and X with a signed-binary result Z.*

The addition tree is simply made up of the adders presented in Section 3.3.1. The exact structure of the addition trees are given in Appendix B. We will now discuss the logic to break carry chains between successive addition operations.

Fig. 5.7 is used to add two signed-binary values, and has the most straightforward logic for breaking carry chains. When r_{i+2} is high (i.e., the MSDs of W and X are present in the first stage), we wish to avoid a carry-in to FA2. This is easily accomplished by ANDing y_{i+1} with $\overline{r_{i+2}}$. When r_{i+1} is high, the carry-out from FA2 must be blocked. Since this carry-out must be inverted to give the correct z^-_i, r_{i+1} can be NORed with the carry out to yield z^-_i. The additional control logic for the rest of the addition circuits in Section 3.3.1 is covered in Appendix A.

5.4.1 Hard Decisions

The final LLR value is calculated using $\lambda_{final} = (L_0 + \lambda'_0)$. This means that L_0 must be delayed by δ_v stages before the addition to match the delay of the addition tree which calculates λ'_0. The result of the addition is fed into a sign detector to form the hard decision output of the variable node. The sign of the summation represents the value of the decoded bit. For example, in a BPSK system a positive value would correspond to a 0 and a negative value to a 1.

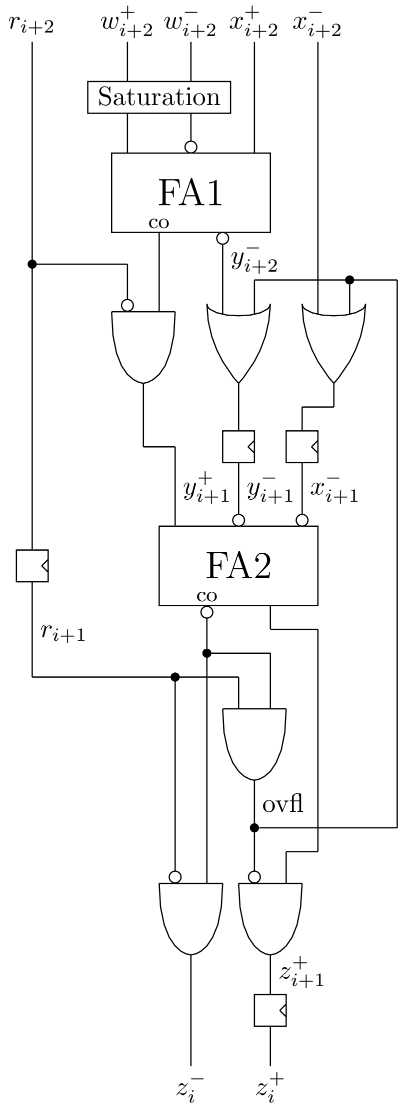


Figure 5.8: *Final addition stage before hard decision. This circuit always produces a sum with the correct sign, but the sum does not always have the correct magnitude without the additional guard digits (and latency) that would prevent overflow.*

The hard decision is performed using a simplified version of the sign-detecting state machine shown in Fig. 3.10. The initial delay to the bit output $\delta_{v,final}$ is not the same as δ_v, but this is accounted for by the control logic. Since this extra initial delay is outside of the processing loop, it does not limit the throughput of the decoder. The extra delay is also small compared to the total number of decoding cycles, so it does not increase decoding latency much.

If the incoming LLR values are large, there may not be enough guard digits to protect against overflow in the final addition. Since only the sign of the sum is required for the hard decision, the final addition can be done in a way that generates an incorrect magnitude on overflow so long as the sign is preserved. This is done using the circuit in Fig. 5.8.

The additional overflow correction performed is just an expansion of the carry-chain-breaking logic. Negative overflow occurs in the second stage of addition when there is a carry-out from FA2. When a carry-out from FA2 is detected, the overflow signal (ovfl) is asserted to perform a correction. In this case, the MSD (the next digit output) should be $\bar{1}$ to give the entire value a negative sign. This means that z^+_{i+1} should be zero and the next z^-_i should be one. When ovfl is high, z^+_{i+1} is immediately zeroed. To ensure that the next z^i_i is one, y^-_{i+2} and x^-_{i+2} are set to ones to ensure that no carry-out from the second full-adder occurs in the next cycle.

Positive overflow occurs in the first stage of the addition when there is a carry-out from FA1. To prevent this, one input is saturated to a maximum positive value to ensure that the sum is not large enough to cause a positive overflow. The saturation value is chosen that it is still large enough to result in a positive sum even if the other input is as negative as is possible.

5.5 Check Node Implementation

HDL for the check nodes is entirely computer generated from a text file containing a selection network. The input file is simply a list of all compare-select operations required to select the two smallest outputs. The compare-select operations are grouped in the input files according to which can be performed in parallel, so the depth of the selection network is known. The HDL generator is parameterizable in terms of the number of pipeline stages and whether offset correction is inserted. The generator traces each input through the selection network and generates the appropriate logic to determine which input had the minimum magnitude from the results of the comparisons.

The selection networks were created by taking sorting networks from [49] and removing unnecessary comparisons[1]. The remaining comparisons were then arranged into the minimum possible depth. For convenience, the sorting networks were exported in the text format used by [49] and all necessary utilities were programmed using this format.

[1]For $N \leq 16$ the best known sorting networks were used. For larger networks, Hibbard's algorithm was used. This algorithm was chosen because it tended to determine the first and second-smallest value the fewest number of stages into sorting, therefore resulting in the minimum depth $(2,N)$ selection network.

A utility was written to test that selection networks are valid. An N-input sorting network can be tested by making sure that it properly sorts all 2^N combinations of zeros and ones at the inputs [48]. Since we are interested in $(2,N)$ selection networks where only the two smallest values are important, we need only test all combinations where all inputs are one except for zero, one, or two inputs that are zero. This ensures that our sorting network always finds the two smallest values, and always correctly orders them. This results in only $(N(N-1))/2+N+1=(N^2+N)/2+1$ test vectors and makes it viable to test the $(2,32)$ selection network required for the degree-32 check nodes of the 10GBASE-T LDPC code. The selection network tester also ensures that the number of comparisons is optimal for the size of the network to ensure minimum area in our check nodes.

5.6 Summary

Building on the general digit-online LDPC decoding architecture presented in the last chapter, we have demonstrated the principles for developing a VLSI implementation of such a decoder. The controller to generate the control signals for the decoder and provide an interface was presented, and the remaining details for the variable and check nodes were filled in. The connections between components were explained, and equations for evaluating the throughput and latency of the decoders were presented.

The next chapter will examine the effects of varying message precision on decoder performance.

Chapter 6

Precision Scaling Analysis

6.1 Introduction

We showed in chapter 4 that the area and delay of circuits are less dependent on precision for a digit-online architecture than for a bit-parallel architecture. For LDPC decoders, the most important performance characteristics are throughput, the throughput/area ratio (TAR), and energy efficiency. The remainder of this chapter looks at the effect of message precision on all of these metrics for synthesized digit-online and bit-parallel decoders.

To increase the word length in a bit-parallel system, the nodes become necessarily larger. Adders and comparators must be widened to accommodate the extra digits, and more wires are required in the interleaver. To increase the message precision in a digit-online decoder, only the number of pipeline stages needs to increase to store the extra digits of precision. Addition and comparison can be done with the same circuits regardless of precision, and the number of wires in the interleaver is fixed with the code chosen.

The size and delay of a variable node is determined by the addition circuits. If we assume a bit-parallel implementation using ripple-carry adders, the area and delay increase linearly with the number of bits. For a tree-adder implementation, the area and delay scale as $O(q \log q)$ and $O(\log q)$, respectively, with the number of bits (q) per message.

The delay and size of a comparison circuit determine the performance of the check nodes. Since bit-parallel comparison is implemented by subtraction, the scaling for bit-parallel check nodes will be the same as bit-parallel variable nodes.

In a digit-online system, the number of clock cycles needed to process each message is proportional to precision. A more precise digit-online decoder is inherently more deeply pipelined, so the clock cycle length could potentially be decreased by rebalancing pipeline stages. However, we first make the conservative assumption that the length of the clock

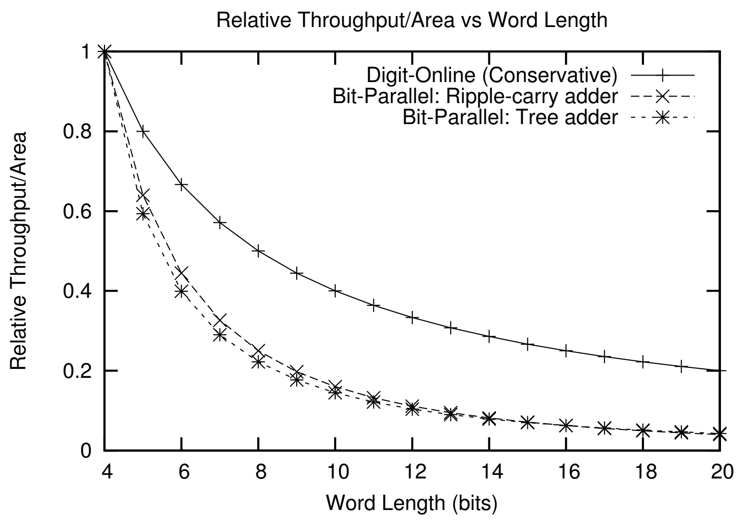


Figure 6.1: *Relative effect of increasing message precision q on throughput/area for various LDPC block decoder implementations. For the digit-online system it is assumed that the clock period is constant as precision increases.*

cycle remains constant. In this case, throughput is inversely proportional to precision. For a p-digit message, the precision q in bits is equal to $p - p_g + 1$, where p_g is the number of guard digits required for the system. Since the area remains constant, the throughput/area should scale as $O(q^{-1})$. A more aggressive assumption is to assume that the length of the clock cycle scales as $O(1/q)$. This results in a throughput that is $O(1)$.

From the scaling orders determined above, the digit-online system has a throughput/area that scales as $O(q^{-1})$ for a constant clock period, or a throughput/area that scales as $O(1)$ if the clock period is $O(1/q)$. A bit-parallel system with ripple-carry adders has a throughput/area of order $O(q^{-2})$. Using tree adders, bit-parallel systems have a throughput/area that scales as $O(1/q(\log q)^2)$.

Fig. 6.1 shows the throughput/area as a function of message precision relative to a 4-bit implementation with the same arithmetic scheme. The bit-parallel implementations scale similarly. However, even with conservative clock period assumptions, the digit-online implementation scales better than any of them. For example, an increase from 4 bits to 8 bits decreases throughput/area to half of the original value in digit-online systems, and to one quarter or less in digit-parallel. To go from 4 bits to 16 bits reduces throughput/area to one quarter of the original value in digit-online, and to one sixteenth in digit-parallel.

6.2 Decoding Parameters

To demonstrate the effects of message precision on decoder performance, many decoders for the irregular WiMAX rate-3/4 base matrix A (which we refer to as rate-3/4A for short) length 1056 code were synthesized in a TSMC 65 nm general purpose process[1]. Both digit-online and bit-parallel decoders were synthesized to allow for a direct comparison of their performance. All results are reported for nominal process corners and the conservative wiring model. Automatic register re-timing was enabled to balance the pipeline stages without designer intervention. Each decoder was synthesized over a large range of clock frequencies. The design with the highest throughput/area for each precision was chosen to present in the thesis. More information about our methodology is presented in Appendix C. Table 6.1 shows the decoder parameters common to all decoders synthesized.

Published WiMAX decoders generally use a channel quantization between 5 and 8 bits, and use up to 9 bits to represent internal messages [26, 50–54]. There is little improvement to error performance beyond 8 bits [54]. Decoders with 12 or more pipeline stages are included to obtain frame-interlaced results at higher precision. It is unlikely that 11 or more bits of precision would be used in practice.

Likewise, since error performance drops off sharply below 5 bits of precision, frame-interlaced decoders for less than 5 bits are included for comparison only and will not provide acceptable error performance.

Throughout this chapter we look at the scaling of decoder parameters relative to a 6-bit decoder of the same architecture with the same number of simultaneous frames. Since inverse relationships are expected between precision and throughput, we show plots of 1/throughput and area/throughput to make the scaling more apparent.

WiMAX 3/4A codes have $d_{v,max} = 4$, so the maximum useful precision in the digit-online architecture is $p_{1f} = l-2$ for single frame decoding, and $p_{2f} = l/2-2$ for frame-interlaced decoding. A p-digit signed-binary number can store a p-bit signed number without requiring an additional sign bit, so it is related to the channel quantization q by the equation $q = p+1$. Thus our pipeline can decode one frame at $(q_{1f} = l-1)$-bit precision or two frames at a $(q_{2f} = l/2-1)$-bit precision.

The chosen code requires variable nodes of degrees 2, 3 and 4 and check nodes of degree 14 and 15. The selection networks of the degree-14 and degree-15 check nodes are

[1]Standard cells used were version 140b of the tcbn65gplus cells. Logic synthesis was performed in Synopsys Design Compiler D-2010.03-SP5.

LDPC Code	WiMAX 802.16e
Codeword size	1056
Code Rate	3/4, base matrix A
Parallelism	Full
Variable Node Degrees	2, 3, 4
Check Node Degrees	14, 15
Decoding Algorithm	Offset Min-Sum
Max. Iterations	10
Early Termination	Yes[1]

[1] Decoders support early termination but results given are worst-case without early termination.

Table 6.1: *Parameters of synthesized WiMAX rate-3/4 base matrix A decoders.*

					Single Frame				Frame-Interlaced			
Pipeline Length	δ_v	δ_c	Clk. Freq. (MHz)	Area (mm^2)	Precision (digits)	Precision (bits)	Tput (Gbit/s)	TAR (Gbit/$s{\cdot}mm^2$)	Precision (digits)	Precision (bits)	Tput (Gbit/s)	TAR (Gbit/$s{\cdot}mm^2$)
6	4	2	575	2.68	4	5	10.1	3.78	1	2	20.2	7.56
7	4	3	990	2.61	5	6	14.9	5.72				
8	5	3	1205	2.52	6	7	15.9	6.31	2	3	31.8	12.62
9	5	4	1449	2.60	7	8	17.0	6.54				
10	6	4	1538	2.70	8	9	16.2	6.01	3	4	32.5	12.02
12	7	5	1923	2.97	10	11	16.9	5.70	4	5	33.8	11.40
14	7	7	1961	3.10	12	13	14.8	4.77	5	6	29.6	9.53
16	7	9	2273	3.12	14	15	15.0	4.82	6	7	30.0	9.63
18	7	11	2381	3.35	16	17	14.0	4.18	7	8	27.9	8.35
20	7	13	2500	3.58	18	19	13.2	3.69	8	9	26.4	7.37

Table 6.2: *Synthesis results of digit-online decoders for the WiMAX rate-3/4A length 1056 code at various message precisions. δ_v is the length of the variable node pipeline and δ_c is the length of the check node pipeline. Decoders with an even number of pipeline stages can be operated in a single-frame high precision mode or a frame-interlaced low precision mode. The precision in digits is the number of signed-binary digits used to store each message. Two additional digits are required for each frame to prevent overflow. Since signed-binary does not require an additional sign bit, the number of bits that can be stored is one more than the number of digits. TAR is the throughput/area ratio. Throughput and TAR are worst-case assuming 10 iterations are run.*

both 6 levels deep. The degree-14 check node requires 25 comparisons while the degree-15 check node requires 27 comparisons. These are the optimal numbers of comparisons for (2,14) and (2,15) selection networks [48].

Pipeline details for the digit-online decoders are shown in table 6.2. Because of result forwarding between pipeline stages, there are many paths around the processing loop with fewer than the maximum number of registers. At the minimum pipeline depth of 4 for a degree-4 variable node there are combinational paths between input and output LLRs. To ensure synchronization of values, the degree-2 and degree-3 variable nodes are pipelined to the same length as the degree-4 variable nodes and each will always have at least one

	Single Frame				Frame-Interlaced			
Precision (bits)	Clk. Freq. (MHz)	Area (mm^2)	Tput (Gbit/s)	TAR (Gbit/$s{\cdot}mm^2$)	Clk. Freq. (MHz)	Area (mm^2)	Tput (Gbit/s)	TAR (Gbit/$s{\cdot}mm^2$)
5	250	1.57	24.0	15.3	526	2.02	50.5	25.0
6	222	1.84	21.3	11.6	446	2.30	42.9	17.6
7	222	2.31	21.3	9.3	286	2.34	27.4	11.7
8	222	2.84	21.3	7.5	386	3.09	37.1	12.0
9	200	3.00	19.2	6.4	245	3.14	23.5	7.5
11	167	3.55	16.0	4.5	222	4.06	21.3	5.3
13	167	4.36	16.0	3.7	254	5.01	24.4	4.9
15	143	4.51	13.7	3.0	200	5.41	19.2	3.6
17	167	5.97	16.0	2.7	233	6.42	22.4	3.5
19	167	6.87	16.0	2.7	143	6.09	13.7	2.3

Table 6.3: *Synthesis results of bit-parallel decoders for the WiMAX rate-3/4A length 1056 code at various message precisions. TAR is the throughput/area ratio. Throughput and TAR are worst-case assuming 10 iterations are run.*

register between input and output. The check node requires forwarding one stage ahead due to the offset subtraction.

Given these minimum forwarding values, the minimum number of registers in the processing loop can be found by subtracting 5 from the pipeline length. Thus a minimum pipeline length of 6 was chosen to ensure that each path contained at least one register. Additionally, 5 bits of precision are required for the optimal offset constant β to be greater than zero at efficient channel scaling.

Increasing the pipeline length of digit-online decoders past 20 increased the synthesis times greatly and made it infeasible to generate enough synthesis results to draw meaningful conclusions from. Even at the pipeline lengths synthesized we are approaching the limits of clock frequency for an edge-triggered design. At higher number of pipeline stages the register clock$\rightarrow$q and setup times become an increasingly large proportion of the clock cycle. About 100 ps of the critical path is consumed by flip-flop setup and clock$\rightarrow$q times, which accounts for more than 20% of the delay at high frequencies.

For the bit-parallel decoders, a fully-parallel design was used. Different decoders were needed for single-frame and frame-interlaced decoding. For the single-frame decoders, the variable and check nodes evaluate during the same clock cycle, giving one clock cycle per iteration. During a given clock cycle for the frame-interlaced decoders, one frame's variable nodes were evaluating while the other's check nodes were evaluating. This results in two clock cycles per iteration. The bit-parallel architecture requires one extra iteration to calculate the hard decisions. Table 6.3 shows the synthesis results for bit-parallel decoders.

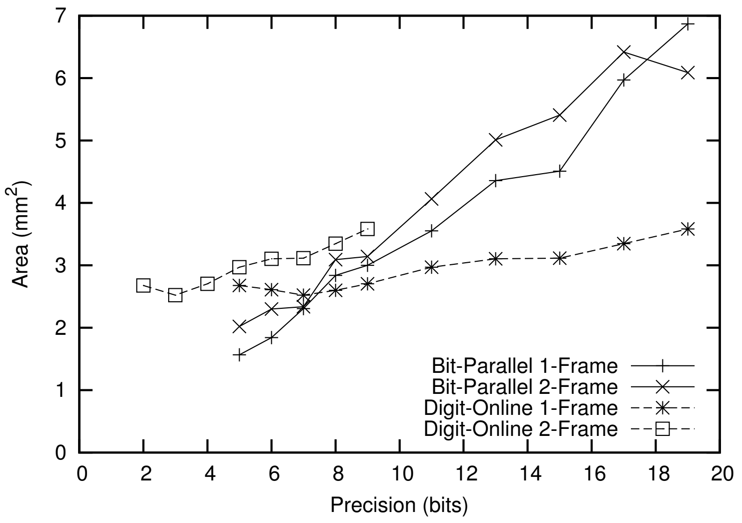


Figure 6.2: *Effect of changing message precision on decoder area for single-frame and frame-interlaced decoding.*

6.3 Area

Fig. 6.2 shows the effect of changing message precision on decoder area. For digit-online decoding, there is a weakly linear relationship between message precision and decoder area for single-frame decoding. This trend is magnified for frame-interlaced decoding. The 6-stage and 7-stage digit-online decoders actually requires more area than the 8-stage digit-online decoder due to the low number of registers in the worst-case loops. For both digit-online and bit-parallel architectures, frame-interlaced decoders are larger than single-frame decoders for a given precision which is part of the cost for their increased throughput. At low precisions the bit-parallel decoders are smaller, but as precision increases they become larger than digit-online decoders.

The breakdown of the area for single-frame digit-online decoding is shown in Fig. 6.3. The size of the control logic is dominated by the size of the input buffer. While the area of the control logic increases with precision, the greatest area increase is accounted for by the greater number of pipeline stages in the computation nodes. For larger numbers of pipeline stages the size of the input buffers could be decreased by allowing only frame-interlaced decoding. However, the area savings would not be significant. As expected, the sequential area of the processing nodes is strictly increasing as the degree of pipelining increases. The combinational area has a downward trend as more of the drive is provided by flip-flops and the cycle times decrease.

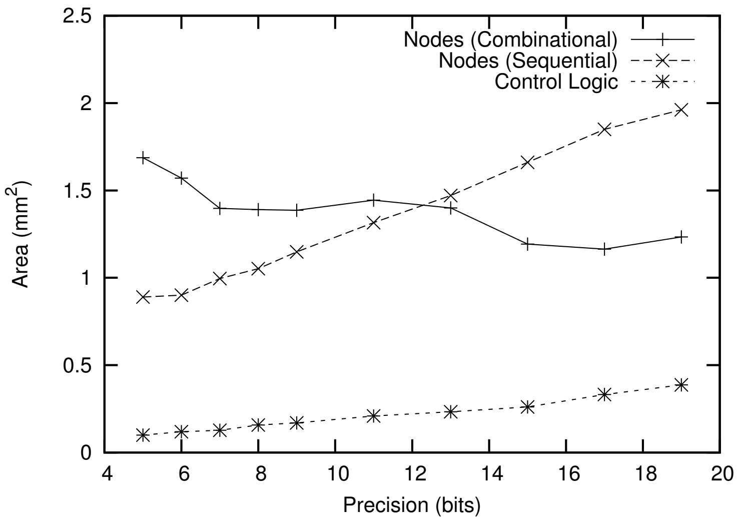


Figure 6.3: *Breakdown of single-frame digit-online decoder area into computation node area control logic area.*

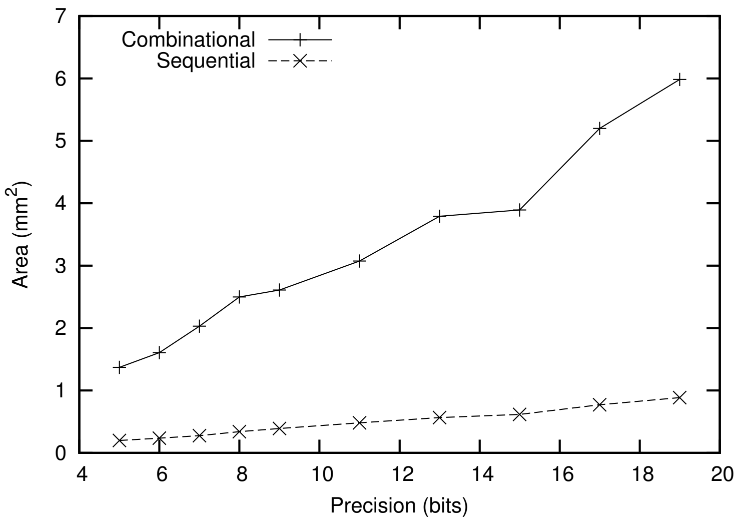


Figure 6.4: *Breakdown of single-frame bit-parallel decoder area into computation node area control logic area.*

Fig. 6.4 shows the area breakdown for single-frame bit-parallel decoders. The sequential area is much smaller than for digit-online decoding, which is expected due to the smaller degree of pipelining.

The relative decoder area for digit-online decoders is shown in Fig. 6.5. We have seen that the constant area assumption holds approximately for the combinational area, but that the sequential area increases significantly with precision. Thus the slopes of the relative

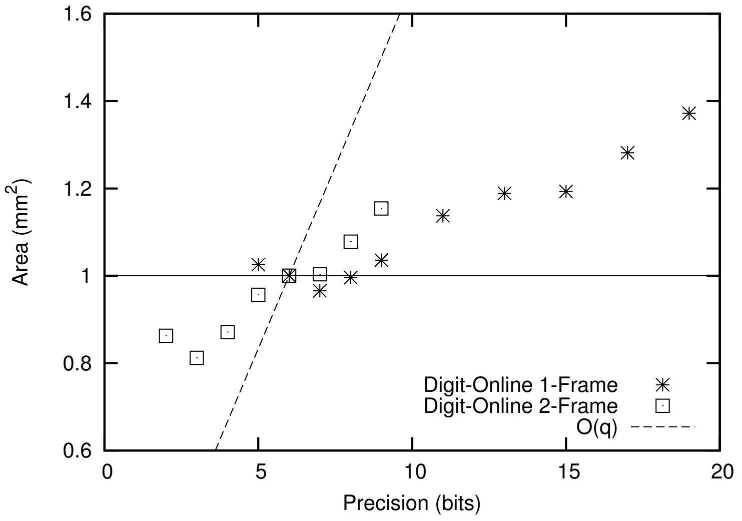


Figure 6.5: *Effect of changing message precision on relative decoder area for single-frame and frame-interlaced digit-online decoding.*

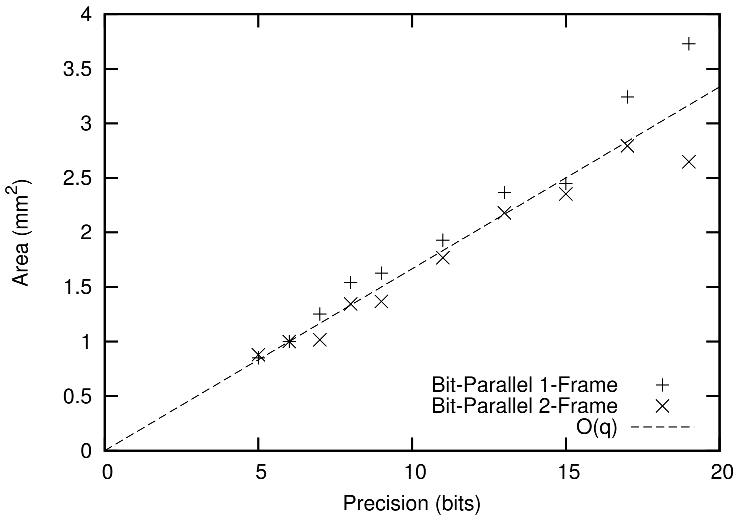


Figure 6.6: *Effect of changing message precision on relative decoder area for single-frame and frame-interlaced bit-parallel decoding.*

areas are greater than zero, but the area is still much less than if area were proportional to precision.

The relative area of bit-parallel decoders is shown in Fig. 6.6. Both single-frame and frame-interlaced decoders have areas that are proportional to precision.

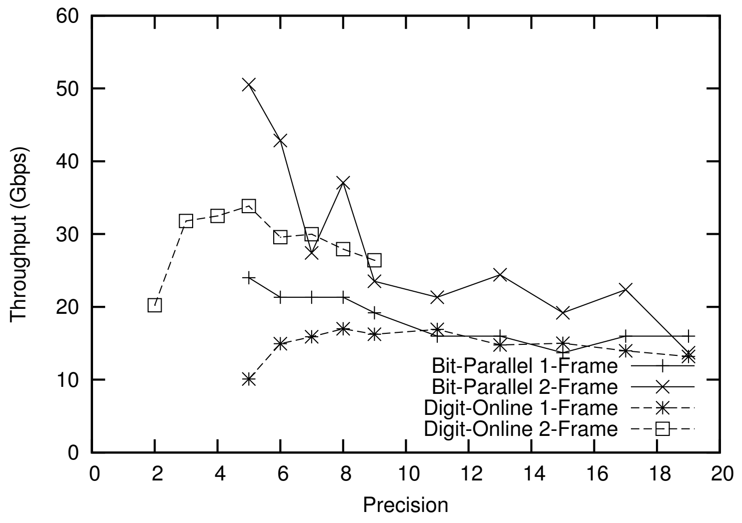


Figure 6.7: *Effect of changing message precision on decoder throughput for single-frame and frame-interlaced decoding. Throughput is worst-case with no early termination.*

6.4 Throughput

The throughput for each of the synthesized decoders is shown in Fig. 6.7.

For digit-online decoders, as the message precision increases, the number of clock cycles per iteration also increases. However, the depth of the pipeline increases as well, allowing the clock cycles to shorten. Thus, for single-frame digit-online decoding with message precisions from 6 to 15 bits the throughput is relatively constant. The effect on throughput is magnified for frame-interlaced decoding, but throughput is relatively flat for 3 to 7 bits of precision. Using frame-interlaced decoding results in much higher decoding throughput for a given precision.

It is much more clear that throughput decreases with precision for bit-parallel decoders. While bit-parallel decoders have higher throughput at low precisions, the throughput of digit-online becomes comparable at higher precisions.

The relative 1/throughput for digit-online decoders is shown in Fig. 6.8. For a digit-online decoder this is equivalent to the clock period times the number of stages. The conservative expectation was for a $1/q$ relationship, but the synthesis results are better than that. Single-frame decoding has a very small slope, and even though frame-interleaved decoding has a steeper slope it is still better than proportional.

Fig. 6.9 shows the relative 1/throughput for bit-parallel decoders. For bit-parallel decoders, this is simply proportional to the clock period and has no direct dependence on pre-

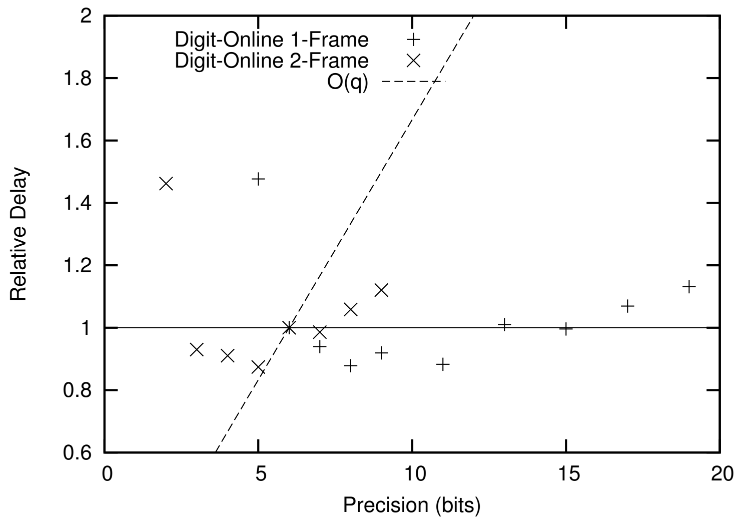


Figure 6.8: *Effect of changing message precision on relative decoder 1/throughput for single-frame and frame-interlaced digit-online decoding. Throughput is worst-case with no early termination.*

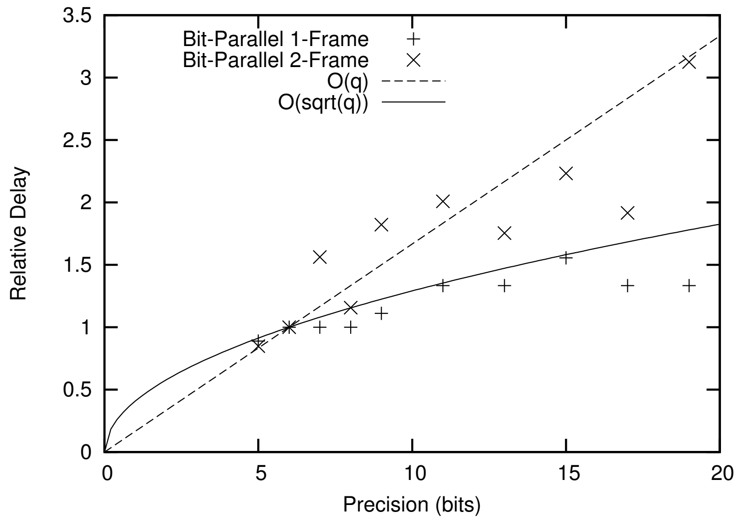


Figure 6.9: *Effect of changing message precision on relative decoder 1/throughput for single-frame and frame-interlaced bit-parallel decoding. Throughput is worst-case with no early termination.*

cision. The frame-interlaced decoder has approximately linear scaling, which (along with its linear area relationship) suggests that the synthesis tool has used ripple-carry adders. The single-frame decoder has a linear area relationship, but its 1/throughput is sub-linear. One possible adder structure is the carry-select adder which has $O(q)$ area and $O(\sqrt{q})$ delay. This assumption seems to fit the delay scaling of the single-frame decoder.

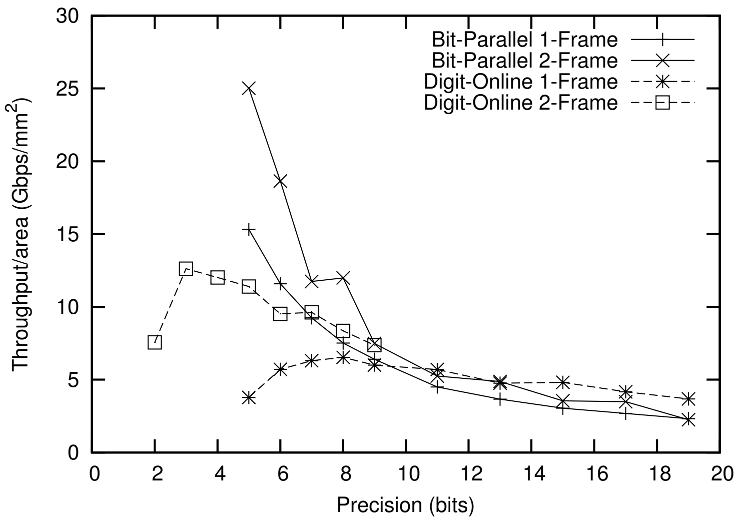


Figure 6.10: *Effect of changing message precision on digit-online decoder throughput/area. Worst-case throughput with early termination disabled is used.*

6.5 Throughput/Area

Since area can often be reduced at the cost of throughput by decreasing parallelism, the area-normalized throughput or throughput/area ratio (TAR) is an important metric of decoder performance. Fig. 6.10 shows the throughput/area ratio for each of the decoders. Despite the fact that frame-interlaced decoders are larger for a given precision, they still have a superior throughput/area ratio.

In contrast to digit-online decoders, bit parallel decoders show a clear inverse relationship between precision and throughput/area. Digit-online decoders have lower throughput/area than bit-parallel decoders at low precision, but they become competitive at higher precision.

By examining the scaling of area/throughput, we can get a clearer idea of the relationship between precision and throughput/area. Fig. 6.11 shows the relative area/delay for digit-online decoders. For 5-9 bits, the frame-interlaced decoder comes close to the conservative estimate of proportional scaling. The single-frame scales much closer to the constant area/throughput expected with $1/q$ clock cycle scaling.

Fig. 6.12 shows relative area/throughput for bit-parallel decoders. Single-frame bit-parallel decoders were estimated to have $O(q)$ area and $O(1/q)$ throughput. This suggests an $O(q^2)$ relationship between precision and area/throughput. While this seems to hold for precisions up to 11 bits, decoders at higher precisions perform slightly better. Frame-

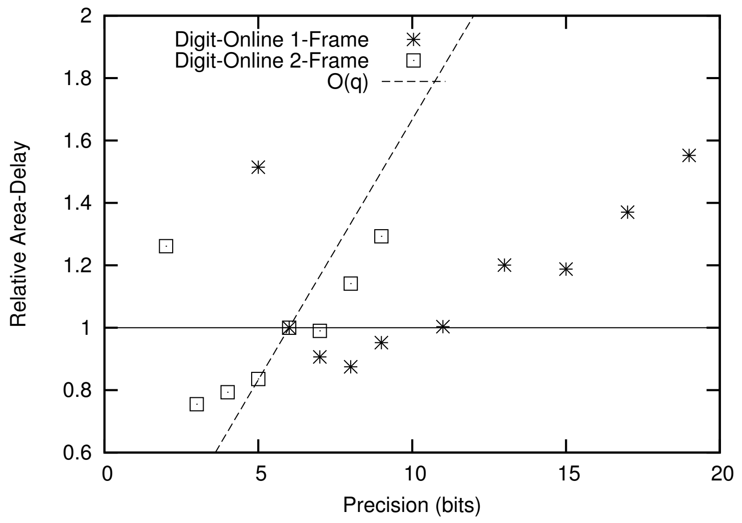


Figure 6.11: *Effect of changing message precision on relative area/throughput for digit-online decoders. Worst-case throughput with early termination disabled is used.*

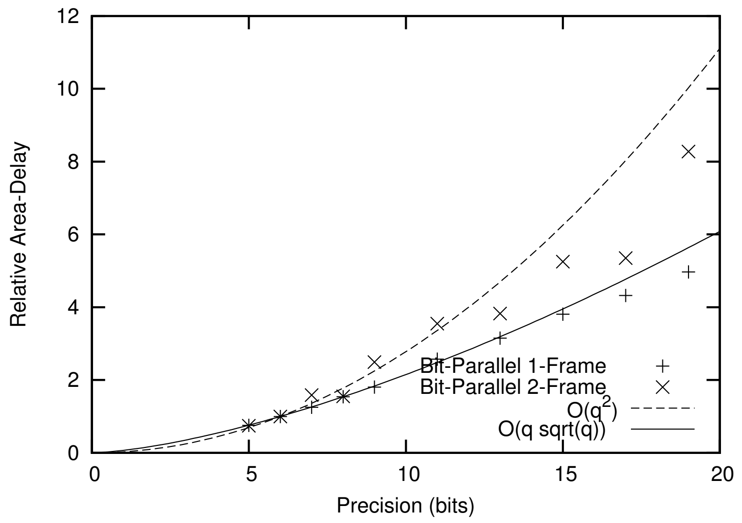


Figure 6.12: *Effect of changing message precision on relative area/throughput for bit-parallel decoders. Worst-case throughput with early termination disabled is used.*

interlaced decoders were estimated to have $O(q)$ area and $O(1/\sqrt{q})$ throughput, resulting in $O(q \cdot \sqrt{q})$ area/throughput. The synthesized results are quite close to this estimate.

6.6 Energy/Bit

The energy per decoded bit results in this section are based on Synopsys power estimates. It is assumed that the true energy/bit may vary by a constant factor from the estimates but the

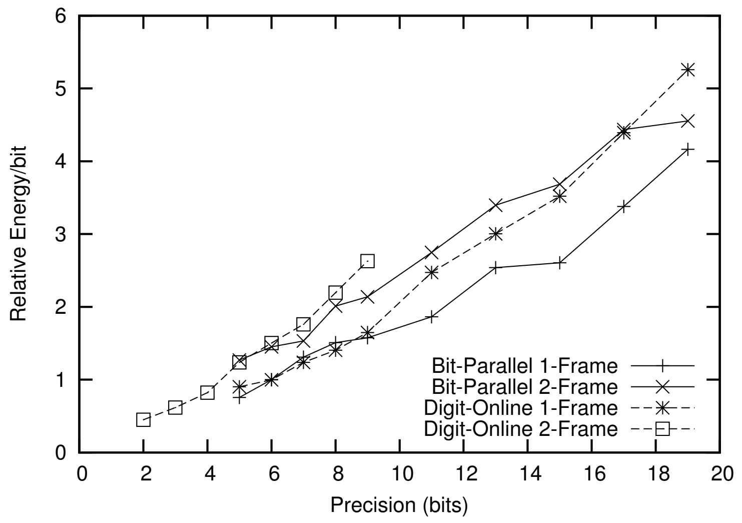


Figure 6.13: *Effect of changing message precision on energy per decoded bit. Each plot is scaled relative to a single-frame 6-bit implementation in the same architecture.*

relative energy values are informative. In particular, the absolute values of energy/bit are not directly comparable between digit-online and bit-parallel decoders, so we look only at relative values. Each energy/bit value is reported relative to the energy/bit of a single-frame 6-bit decoder of the appropriate architecture (either digit-online or bit-parallel).

Fig. 6.13 shows the energy efficiency of the decoders in both single-frame and frame-interlaced decoding modes. While frame-interlaced decoding results in better throughput and throughput/area performance at a given precision, a higher energy/bit is required to decode two frames at once. The energy/bit has a larger dependence on precision for the digit-online decoders.

Fig. 6.14 shows the energy additional cost of frame-interlaced decoding. For digit-online decoding, the energy/bit increases by 40-60% in the range of precisions where both 1-frame and 2-frame results are available. Bit-parallel generally has a lower energy increase for frame-interlaced decoding, with most points falling between 30% and 50%.

The static power of all digit-online decoders is very small. Static power accounts for about 1.5% of total power for the 6-stage decoder and less than 1% for all others. The bit-serial decoders in [28] and [27] both reported much less than 1% static power for their designs which were fabricated in 180 nm and 130 nm processes, respectively. The static power consumption of bit-parallel decoders is 5-8% of total power. The static power is more significant in bit-parallel due to the smaller fraction of sequential cells.

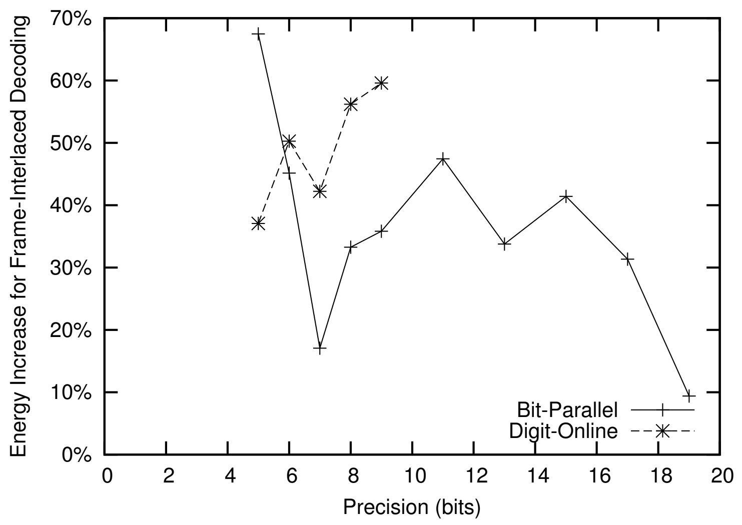


Figure 6.14: *Increase in energy/bit for frame-interlaced decoding.*

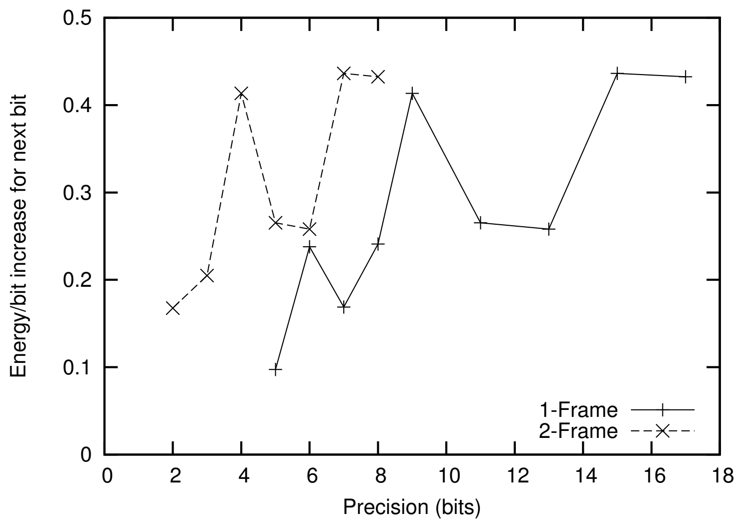


Figure 6.15: *Marginal energy/bit cost to increase precision by one bit in digit-online decoders. In regions where precision increases two bits at a time it is assumed that the increase from each bit is equal.*

The next two plots show the increase to energy/bit to increase the precision by one bit at each precision. For regions where the decoders vary by two bits of precision it is assumed that the increase from each bit is equal.

Fig. 6.15 shows the marginal energy/bit cost for digit-online decoders. The marginal cost generally trends upward, suggesting that each additional bit becomes more expensive from an energy/bit standpoint. At a given precision, frame-interleaved decoding has a higher marginal energy/bit cost than single-frame in addition to the higher absolute ener-

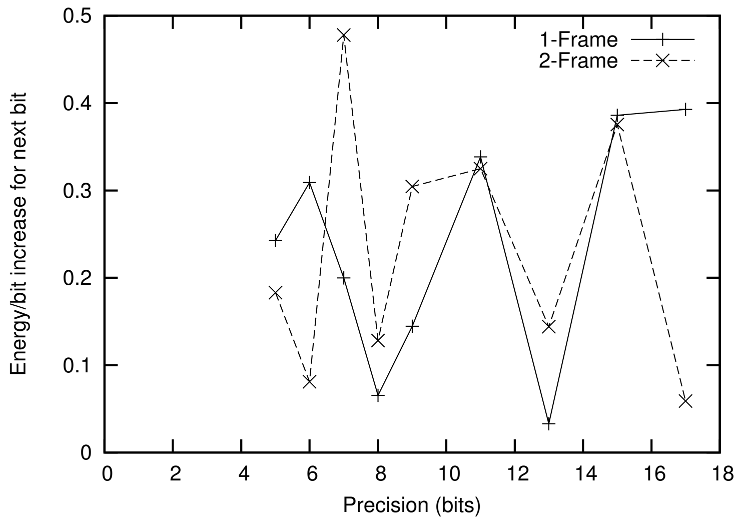


Figure 6.16: *Marginal energy/bit cost to increase precision by one bit in bit-parallel decoders. In regions where precision increases two bits at a time it is assumed that the increase from each bit is equal.*

gy/bit. This can be seen in Fig. 6.13. The slope of the plot increases with precision, and the energy/bit of the frame-interlaced decoders increases more rapidly with precision than single-frame.

The marginal energy/bit cost for bit-parallel decoders is shown in Fig. 6.16. The marginal cost does not seem to grow with precision, and is similar for single-frame and frame-interlaced decoding. Referring to Fig. 6.13, both single-frame and frame-interlaced decoding have similar slopes which do not change much with precision.

6.7 Summary

We have examined the effects of varying message precision in LDPC decoders for the irregular WiMAX rate-3/4A length-1056 code. Both digit-online and bit-parallel architectures were examined.

We have shown that for digit-online decoders there is often no throughput cost to increasing the message precision, and decoder area grows slowly with precision (e.g., tripling the precision, from 5 to 15 bits, results in a 1.16x increase in area). However, the cost in decoding energy efficiency is significant and must be considered when designing a digit-online decoder.

We have demonstrated that in both architectures, frame-interlaced decoding is superior to single-frame decoding when it comes to throughput and throughput/area. However, frame-interlaced decoding comes at a higher energy/bit cost, so designers must consider whether the increased power consumption is acceptable.

Digit-online decoders have areas and throughputs that are less sensitive to precision than bit-parallel decoders. Bit-parallel decoders have an energy efficiency that grows more slowly with precision, and a lower energy cost to frame-interlaced decoding.

The next chapter goes beyond logic synthesis and compares the post-layout performance of digit-online and bit-parallel decoders.

Chapter 7

Post-Layout Analysis

7.1 Introduction

While the previous chapter focused on synthesis results, the effect of wiring congestion on the performance of LDPC decoders cannot be ignored. For this reason, the post-layout performance of digit-online decoders must be explored. Again, a direct comparison to bit-parallel decoders will be provided. Since wiring congestion affects different sized designs differently, we will look at post-layout results (in a TSMC general-purpose 65-nm process) across a range of code sizes[1]. Our synthesis and place and route methodologies are presented in Appendix C.

We begin by again using the WiMAX family of codes since they provide a large range of code sizes. Given that placement and routing have much higher computing requirements than synthesis, the design space must be constrained in order to make analysis feasible. We use 9-bit quantization, which is reasonable for the requirements of WiMAX but towards the high end [26, 50–54]. Using a high quantization will result in an error performance that is at least as good as other decoder implementations.

We will look at the difference between bit-parallel and digit-serial implementations over various code sizes. WiMAX allows code lengths from 576 to 2304. In addition to the length-1056 code used in our detailed precision analysis, decoders of length 576, 1728, and 2304 will be synthesized to span the entire range of WiMAX code sizes.

[1]Version 140b of the tcbn65gplus standard cell library was used for all results in this chapter. Logic synthesis was performed in Synopsys Design Compiler D-2010.03-SP5.

Both [18] and [22] report that achievable placement density[2] decreases as parallelism increases in bit-parallel implementations. Based on those results, we expect density in the bit-parallel implementation to decrease with code size. If digit-online has less routing congestion, its achievable placement density should be less dependent on code size. Additionally, total wire length can be used as a measure of routing congestion [25]. We also expect digit-online decoders to have a lower total wire length than bit-parallel because of the much smaller number of interleaver wires required and because of their narrower data paths.

Section 7.2 compares decoder architectures for the WiMAX rate 3/4A code, which is the code used in the previous chapter for our precision analysis. Frame-interlaced architectures are used in order to target maximum throughput, with the expectation that energy efficiency can be improved by voltage scaling. We find that under these conditions the throughputs of digit-online decoders are limited by the extremely high clock frequencies that are required. These high clock frequencies result in a large throughput drop from synthesis to post-layout.

To see how the architectures compare for a higher code rate, we look at decoders for the WiMAX rate-5/6 code in section 7.3. This section uses single-frame decoding to decrease the required clock frequencies to a more acceptable level. Under these conditions, digit-online compares more favourably to bit-parallel, which suffers because of the higher congestion associated with higher check node degrees. We show that local check node wiring is the largest contributor to wiring congestion in high-rate codes.

Finally, we present a digit-online decoder ASIC for the 10GBASE-T code in section 7.4 in order to compare our architecture to the literature. We show that our architecture results in a fully parallel decoder with superior throughput and throughput/area to other published designs.

7.2 WiMAX Rate-3/4A Code Size Study

7.2.1 Area and Throughput

Fig. 7.1 shows the post-layout area vs. code size for 9-bit frame-interlaced LDPC decoders. There is very little difference in area between the digit-online and bit-parallel decoders for any code size. However, even though the architectures have similar areas, the digit-

[2]The initial density of a design refers to the ratio of total standard cell area to core area before placement. The final density refers to the ratio of total standard cell area to core area after placement, optimizations, and clock tree insertion. Unless otherwise specified, placement density refers to the initial density.

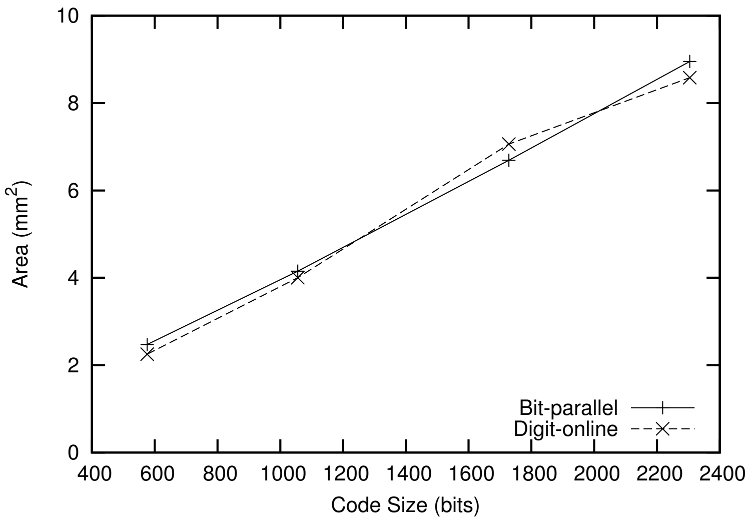


Figure 7.1: *Post-layout area for 9-bit frame-interlaced WiMAX rate-3/4A decoders for various code lengths.*

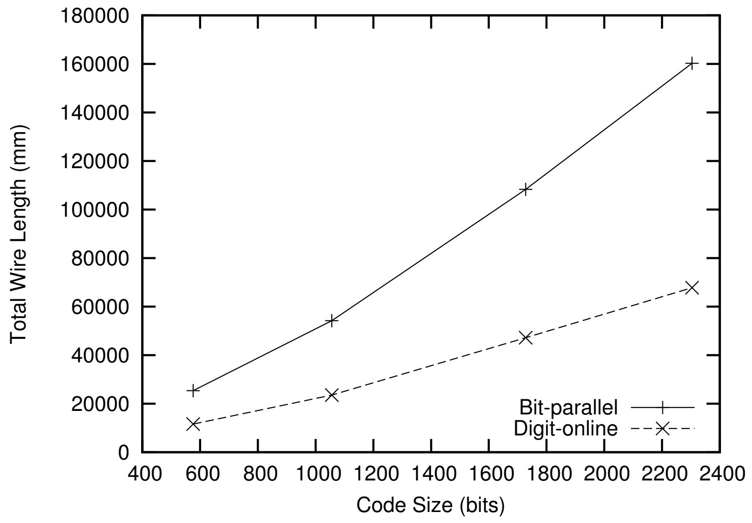


Figure 7.2: *Post-layout wiring length for 9-bit frame-interlaced WiMAX rate-3/4A decoders for various code lengths.*

online architecture requires less than half as much total wiring as a bit-parallel architecture (Fig. 7.2). This is not surprising, since the digit-online architecture requires only 2/9ths as many wires for the interleaver, and also has narrower data paths. There was evidence of ill effects from wiring congestion in the bit-parallel designs for longer codes. The bit-parallel decoders for the length 1728 and 2304 codes could not be routed when placed with an initial density of 90%, even though the digit-online decoders were still routable at such a dense placement. The length 2304 bit-parallel decoder went from an initial density of 85% to a

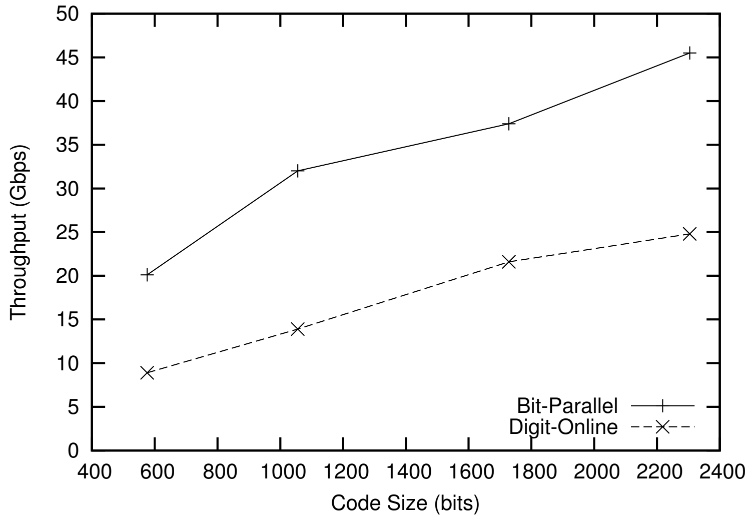


Figure 7.3: *Post-layout throughput for 9-bit frame-interlaced WiMAX rate-3/4A decoders for various code lengths.*

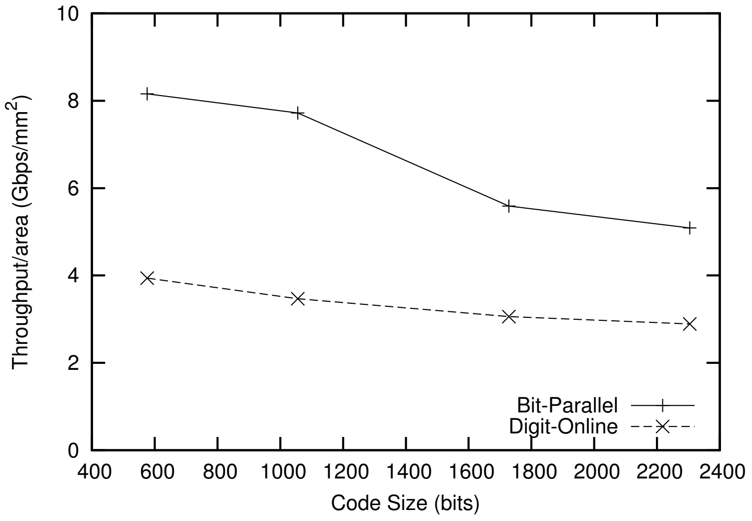


Figure 7.4: *Post-layout throughput/area for 9-bit frame-interlaced WiMAX rate-3/4A decoders for various code lengths.*

final density of 97.2%. This large increase in density suggests that wire congestion was resulting in large wire delays which needed extra buffering to meet timing. In comparison, the length-2304 digit-online decoder grew from a 90% initial density to a final density of 96.6%. When increasing the code length from 576 to 2304, the clock frequency dropped by 44% for bit-parallel decoders and by only 30% for digit-online decoders.

Fig. 7.3 shows the throughput of the decoders. There is a clear advantage for bit-parallel. The digit-online decoders suffer because they require very high clock frequencies which are difficult to maintain through placement and routing. The digit-online decoders would require a clock frequency 10x as high as the bit-parallel decoders to have equal throughput. Using frame-interlaced decoding compounds this problem because it doubles the number of pipeline stages resulting in even higher clock frequencies.

The throughput/area of the decoders is shown in Fig. 7.4. Since the bit-parallel decoders had a higher throughput than and a similar area to the digit-online decoders, they have a superior throughput/area. The TAR of digit-online decoders decreases less when increasing code length from 576 to 2304 (27%) than bit-parallel (39%), but the bit-parallel system still has a much higher TAR at the largest code size.

7.3 WiMAX Rate-5/6 Code Size Study

High check node degrees exacerbate hardware and wiring complexities in bit-parallel decoders, especially in highly parallel designs [22]. Since bit-parallel decoding seemed to be more appropriate for the WiMAX rate-3/4A codes, we now look at the WiMAX rate-5/6 codes which have higher check node degrees. WiMAX rate-5/6 codes use $d_c = 20$ compared to $d_c = \{14, 15\}$ for rate 3/4A. We also use single-frame decoding. By using a higher check node degree and performing single-frame decoding, the clock frequency of the digit-online decoders can be greatly reduced. This should result in better post-layout performance.

7.3.1 Area and Throughput

For single-frame rate-5/6 decoders, the digit-online architecture results in areas 10-29% smaller than equivalent bit-parallel designs, as shown in Fig. 7.5. The difference in wire length has grown, with bit-parallel designs requiring more than 2.6x as much wiring as equivalent digit-online designs (Fig. 7.6).

The throughput and throughput/area of the single-frame rate-5/6 decoders is shown in Fig. 7.7 and Fig. 7.8, respectively. The bit-parallel decoders still achieve a better throughput, but the throughput/area gap has shrunk significantly. While the frame-interlaced rate-3/4A digit-online decoders achieved about half of the throughput/area as bit-parallel, for single-frame rate-5/6 decoding they now have a throughput/area of no more than 15% less. For the length-576 and length-2304 codes the throughput/area performance is almost identical between the two architectures.

At length-2304, the rate-5/6 bit-parallel design suffered even more from congestion than did the rate-3/4A design. Again the decoder was unroutable at 90% placement density, but

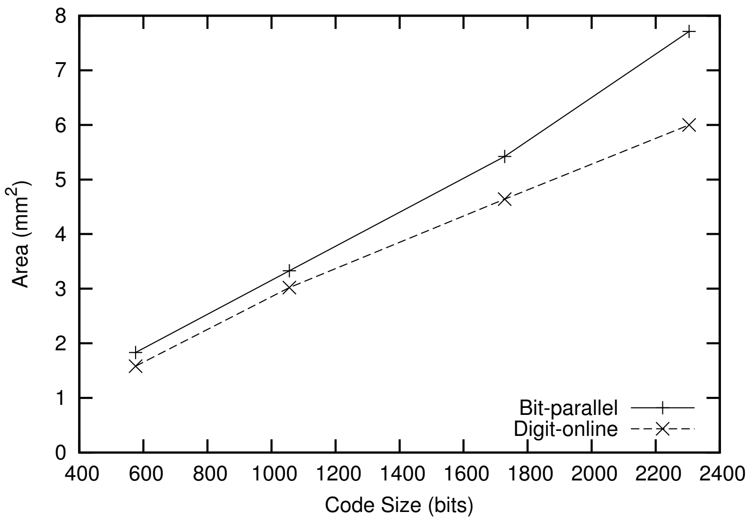


Figure 7.5: *Post-layout area for 9-bit frame-interlaced WiMAX rate-5/6 decoders for various code lengths.*

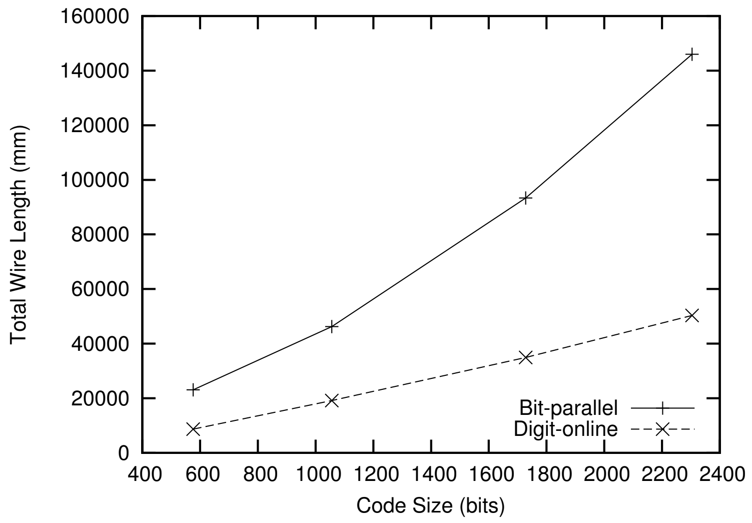


Figure 7.6: *Post-layout wiring length for 9-bit frame-interlaced WiMAX rate-5/6 decoders for various code lengths.*

even at 85% the timing goal had to be greatly reduced to allow the design to meet timing. Digit-online decoders were able to achieve a consistent throughput/area from length 1056-2304.

7.3.2 Wiring and Congestion

To demonstrate the difference in wiring congestion, we look at the percentage of available routing that was used. This utilization is broken down by metal layer, but metal 1 is omitted

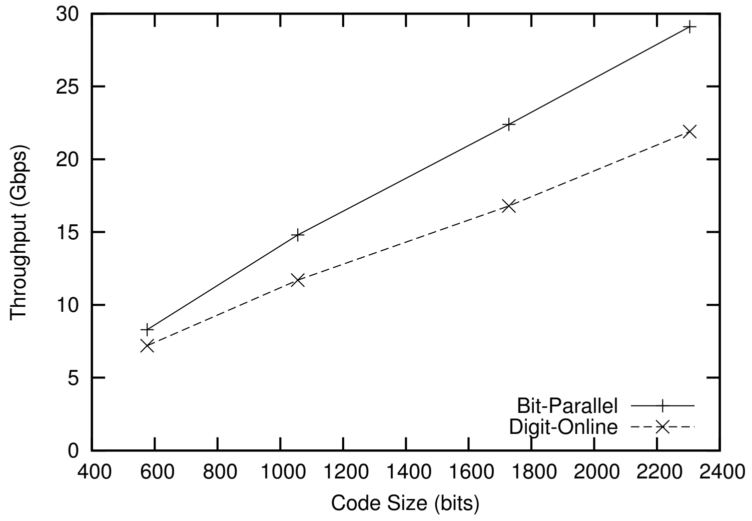


Figure 7.7: *Post-layout throughput for 9-bit frame-interlaced WiMAX rate-5/6 decoders for various code lengths.*

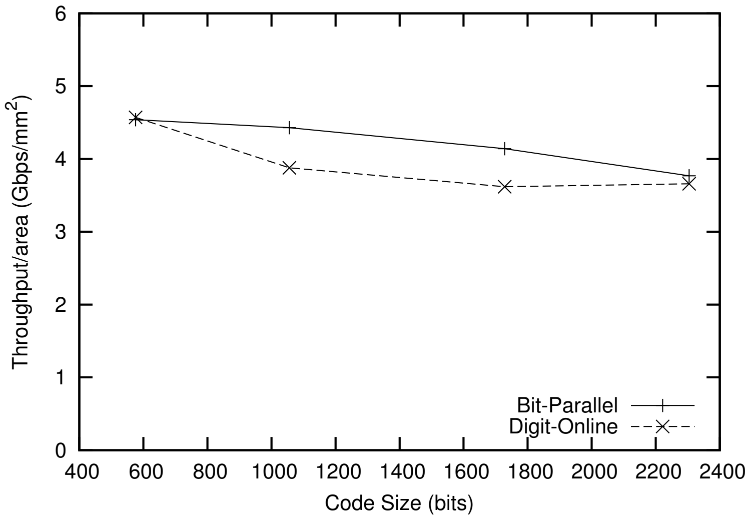


Figure 7.8: *Post-layout throughput/area for 9-bit frame-interlaced WiMAX rate-5/6 decoders for various code lengths.*

because it is largely consumed inside standard cells and very little is used for routing (less than 0.3% of what would be available if none was used inside standard cells). Metal 2 is also present in standard cells, but we report only the portion of metal 2 used for routing between cells.

Fig. 7.9 shows the routing utilization for bit-parallel decoders, and Fig. 7.10 shows the routing utilization for digit-online decoders. In both cases the utilization increases with the code length. However, the digit-online systems show a much lower overall utilization,

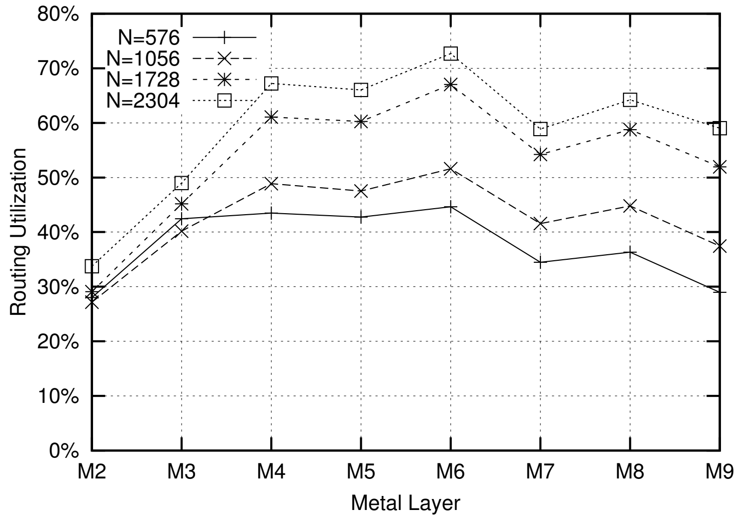


Figure 7.9: *Routing utilization by metal layer for bit-parallel single-frame WiMAX rate-5/6 decoders of various code lengths*

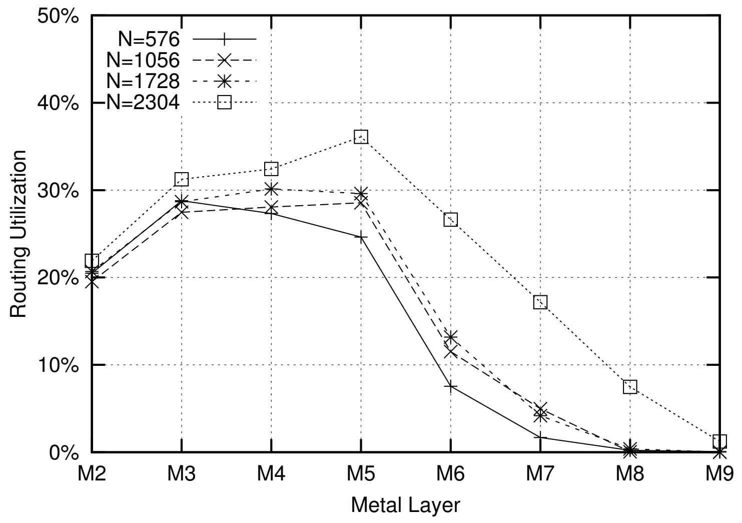


Figure 7.10: *Routing utilization by metal layer for digit-online single-frame WiMAX rate-5/6 decoders of various code lengths*

particularly on the higher metal layers. This suggests that the digit-online systems are much less congested than the bit-parallel systems.

To examine the cause of congestion, we look at a breakdown of total net length by function for the length-2304 decoders. This total net length is normalized to core area to give a measure of congestion. The wire lengths/core area are shown in Figs. 7.11 and 7.12 for bit-parallel and digit-online decoders, respectively. Here we see that for both architectures, the largest fraction of wiring is consumed locally in the check nodes. As expected, each

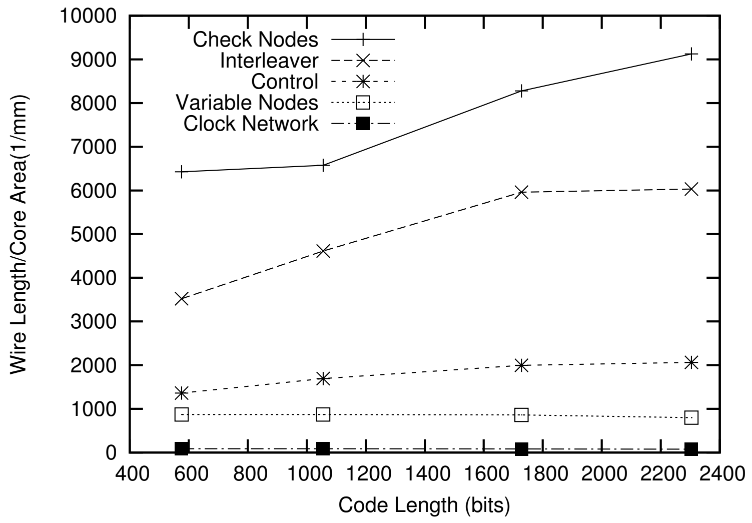


Figure 7.11: *Wire length/core area by function for bit-parallel length-2304 single-frame WiMAX rate-5/6 decoders*

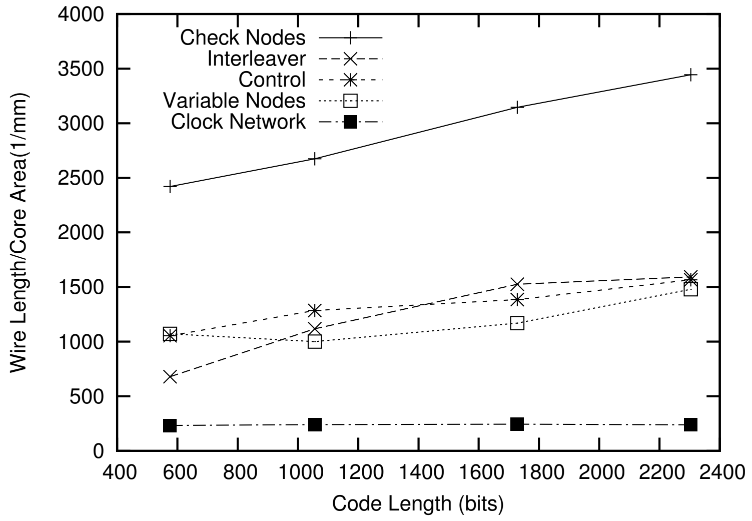


Figure 7.12: *Wire length/core area by function for digit-online length-2304 single-frame WiMAX rate-5/6 decoders*

of the functions consumes less wiring in the digit-online systems than the corresponding bit-parallel systems.

The LDPC code for the 10GBASE-T standard uses an even higher check node degree of 32. Since the relative performance of digit-online decoder seems to improve at higher d_c, we expect that a digit-online ASIC will compare well to other published implementations.

7.4 10GBASE-T ASIC

Using the digit-online architecture, a decoder for the (2048,1723) 10GBASE-T LDPC code was implemented. The decoder requires a core area of 10.89 mm^2 in a TSMC general-purpose 65-nm process. The post-layout clock frequency is 980 MHz under nominal conditions and using the conservative wiring model. The design was placed at an initial core density of 80%. We were able to place and route our design at 90% initial density, but the achievable clock frequency dropped from 980 MHz to 890 MHz.

The degree-32 check node was based on a selection network with 61 comparisons in 8 levels. It was pipelined to give $\delta_c = 6$. Using the high fan-in addition circuit of Fig. 3.7, a two-stage variable node was designed and pipelined to give $\delta_v = 6$. The 12 stages of the total system pipeline allows for 12-digit signed-binary messages to be stored and processed. Three of the digits are required to prevent overflow, leaving 9 that can store useful data. As previously explained, LLRs are saturated to 9 digits once per iteration. Since signed-binary notation does not require a separate sign digit, this is equivalent to using 10-bit two's complement messages.

Since 10-bits of precision is much higher than what is found in other published 10GBASE-T decoders [18, 22, 27], the 12 stages of the pipeline can also be used to store two 4-bit (3 signed-binary digit) frames and decode them in a frame-interleaved fashion. This requires changes to only the control logic, so it is possible for the same decoder to decode either one 10-bit frame or two 4-bit frames at once.

The error performance of the system has been verified with bit-accurate C-simulations. Fig. 7.13 shows the error performance of the decoding system at 4-bit and 10-bit precisions. The higher precision provides only a slight 0.1 dB coding gain improvement but is included because of the low area cost (about 1%) of supporting this mode as well.

At low SNR, given 8 iterations per codeword and 12 clock cycles per iteration, a 980 MHz operating frequency corresponds to a 20.9-Gbit/s coded throughput at 10-bits. At 5.5 dB we are able to realize a 2.0x speedup from early termination, bringing the coded throughput to 41.9 Gbit/s. Using the decoder to decode two interleaved 4-bit frames doubles the worst-case throughput to 41.8 Gbit/s and brings the throughput at 5.5 dB to 82.8 Gbit/s.

The power consumption of the decoder is 8.9 W. This corresponds to an energy/bit of 128 pJ/bit for 4-bit messages and 254 pJ/bit for 8-bit messages. Sequential elements consume 45.9% of the power while combinational elements consume 45.4%. The remaining

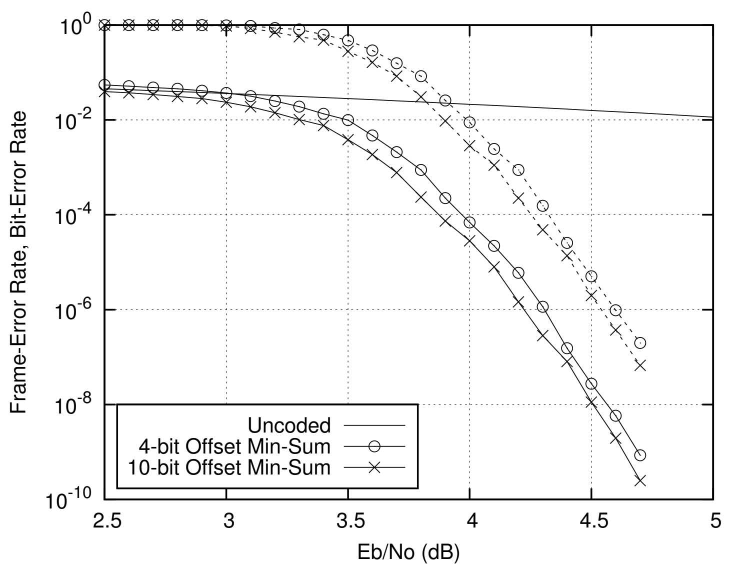


Figure 7.13: *Frame-error rate (dashed lines) and bit-error rate (solid lines) performance of the (2048,1723) 10GBASE-T LDPC code from bit-accurate C-simulation using BPSK modulation.*

8.7% is consumed by the clock network. Leakage power is only 56 mW, which is 0.6% of the total power. This fraction is consistent with power measurements reported for bit-serial systems [27, 28].

A comparison of results is shown in Table 7.1. Throughput is normalized to area to give the throughput/area ratio (TAR). To achieve a more fair comparison, we scale the 90-nm results of [22] and [27] to 65 nm using the approximate scaling factors of [39].

At a 4-bit precision, the digit-online architecture is an improvement over bit-serial ([27]) in terms of absolute throughput while achieving a similar area-normalized throughput. It should be noted that the check nodes in [27] calculate only a single minimum and not the extrinsic minimums. This decreases their areas, but results in an SNR shift of about 0.5 dB compared to extrinsic minimum algorithms. Additionally, their results are pre-layout. A placed design would have a larger core area and possibly a lower clock frequency.

The bit-parallel decoder in [18] achieved a coded throughput of 47.7 Gbit/s through the use of early termination. They reported that routing congestion at high levels of parallelism limited their placement densities and timing. As a result they chose to implement a partially parallel design to limit congestion. They were able to achieve 80% placement density at their desired timing. The decoder in [22] required a lower 70% initial placement to meet their goals even though it is a much smaller design. Our design was fully parallel and yet

	Darabiha *et al.* [27]	Z. Zhang *et al.* [18]	Ueng *et al.* [22]	This work [46]	
Technology (nm)[a]	90	65	90	65	
Architecture	Bit-serial	Bit-parallel	Bit-parallel	Digit-online	
Max. Iterations	8	8	8	8	
Early Termination	yes[b]	yes	no	yes	
Core Area (mm^2)	5.11[c]	4.52	2.30	10.89	
Initial Density		80.0%	70%	80.0%	
Final Density		84.5%	76%	87.3%	
Clock Freq. (MHz)	346	700	420	980	
LLR Precision (bits)	4	4	5	4	10
SNR (dB)[d]	4.8	4.3	4.2	4.3	4.2
Block interleaving	yes	no	no	yes	no
E/b (pJ/bit)		59	48	128	254
E/b @ SNR (dB)		5.5		5.5	
Throughput (Gb/s)[e]	22.1	13.3	6.7	41.8	20.9
TAR (Gb/s/mm^2)[e]	4.33	2.48	2.92	3.84	1.92

[a] Results for 90 nm decoders are scaled to 65 nm.
[b] Early termination to lower power only, does not increase throughput.
[c] Synthesis result.
[d] To achieve a bit-error rate of 10^{-6}.
[e] For 8 iterations.

Table 7.1: *Comparison of sub-100-nm LDPC decoder ASICs for the 10GBASE-T (2048,1723) LDPC code*

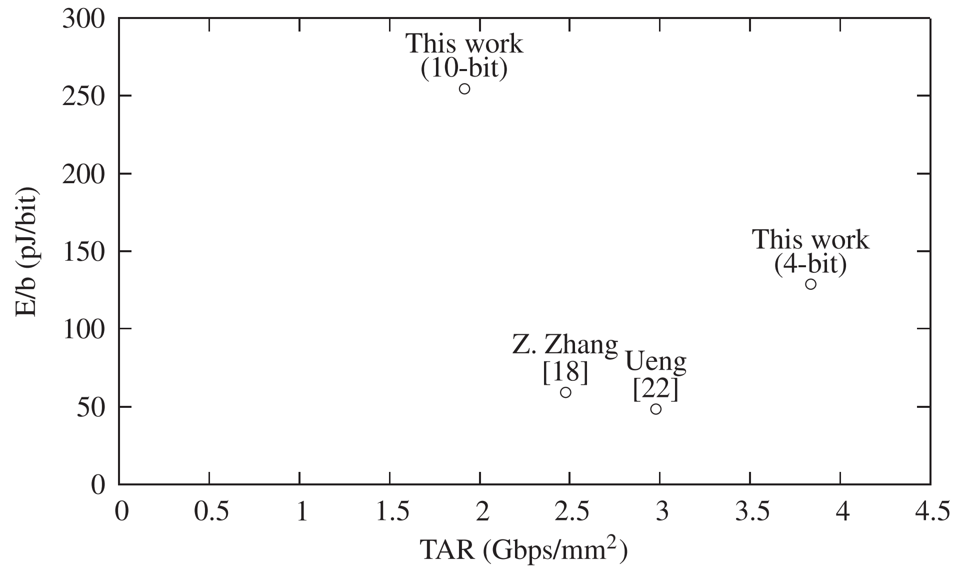


Figure 7.14: *Worst-case (8 iteration) throughput/area vs. E/b @ 5.5dB of published 10GBASE-T decoders. E/b and TAR values are given for estimated performance in a 65-nm process.*

we observed no routing congestion or timing difficulties at 80% placement density. This presents a strong argument that digit-online computations can mitigate routing issues in highly parallel decoder designs.

At 4-bit precision we achieve a higher throughput and throughput/area than [18] and [22]. Our bit-error rate performance is identical to [18]. The decoder in [22] performs slightly better due to the higher message precision of 5-bits. At a 10-bit precision we still achieve a higher throughput than [18] and [22], but at a lower throughput/area.

The energy/bit of the digit-online decoder is larger than the bit-parallel decoders. At 4 bits the digit-online decoder has an energy/bit of 2.2-2.7x higher than the bit-parallel decoders. However, not much work has been done to optimize the power of digit-online decoders yet. Additionally, [18] was implemented in a low-power process whereas our decoder was implemented in a general-purpose process. Voltage scaling could also improve the E/b of the digit-online design. The 10-bit E/b of the digit-online decoder is about double the E/b at 4-bit. Given the much lower throughput and the much higher E/b, it is probably not practical to use 10 bits of precision to achieve a 0.1 dB higher coding gain.

Fig. 7.14 shows a scatter plot of the throughput/area and E/b values of the ASICs we have discussed.

7.5 Summary

High-precision digit-online decoders tend to have a smaller area than equivalent bit-parallel decoders. However, digit-online decoders do not always have superior throughput or throughput/area. Because digit-online decoders require much higher clock frequencies than bit-parallel decoders, their post-layout performance suffers compared to synthesis results if the pipeline length becomes too long (as observed for frame-interlaced decoding at high precision). Digit-online decoders also compare more favourably to bit-parallel decoders where the check node degree is high. This decreases the clock frequency digit-online decoders need to match the throughput of bit-parallel decoders (since the achievable clock frequency of decoders decreases with increasing check node degree). Additionally, high check node degrees increase wiring congestion in bit-parallel decoders more than in digit-online decoders due to the wider data-paths in bit-parallel decoders. Congestion did not limit throughput/area for bit-parallel decoders of lengths 576 and 1056, but became an issue at lengths 1728 and 2304.

The digit-online decoder for the 10GBASE-T LDPC code compares favourably to other published ASICs. This is likely due to having a higher check node degree than the WiMAX codes. It achieves a higher throughput and throughput/area than other published ASICs for the same code. It also demonstrates that digit-online decoders are able to achieve good throughputs at very high precision. However, the energy efficiency is not as good as other published results. This gap could be narrowed by using a low-power rather than general-purpose process, or voltage scaling. Future work should also generate a fully parallel bit-parallel decoder for the 10GBASE-T code for a direct comparison to digit-online.

Chapter 8

Conclusion

8.1 Summary of Contributions

Fully parallel LDPC block code decoders are potentially limited by wiring congestion. Using serial communications on-chip is one approach to mitigating this problem, but existing bit-serial message-passing LDPC block code decoders have several shortcomings. By using signed-binary redundant notation, we have expanded bit-serial message-passing to digit-serial message-passing. This allows us to use digit-online techniques to develop deeply pipelined LDPC decoders. Redundant notation removes the need of previous bit-serial decoders to reverse the bit order at each node or de-serialize and perform parallel operations. Unlike existing bit-serial decoders, frame-interleaving is not necessary for efficient use of hardware.

Digit-online decoders do not require any special considerations for code design, do not require specialized decoding algorithms, and are suitable for fully parallel decoding. They achieve throughput comparable to state-of-the-art implementations and make efficient use of silicon area. Placement density of at least 80% is achievable without routing congestion, even for a fully parallel design.

Digit-online LDPC decoders have a lower relative cost to increasing word length than bit-parallel systems. They show promise to be used in implementations where high computing precision is needed. Digit-online decoders can dynamically switch between a higher-precision lower-throughput mode and a frame-interleaved lower-precision higher-throughput mode.

In this thesis we have presented a complete digit-online LDPC decoding architecture along with several ASIC implementations. We have demonstrated how the area, throughput, and energy efficiency of digit-online LDPC decoders vary with precision. The effect of increasing code length on post-layout performance was also presented. We have provided a comparison to bit-parallel systems and shown the conditions under which each of the architectures should be preferred. A digit-online LDPC decoder was presented for the length-2048 rate-0.84 10GBASE-T LDPC code which compares favourably to other published ASICs. At 4-bit precision our 82.8-Gbit/s throughput is 73% higher than the published state-of-the-art. The 10-bit decoding mode extends the precision of previously published decoders from 4-5 bits. Our throughput of 41.8 Gbit/s with 10-bit resolution is only 12% less than the state-of-the-art 4-bit decoder.

8.2 Future Directions

The digit-online decoding architecture was developed with no assumption about the structure of LDPC codes and is applicable to completely random $\boldsymbol{H}$ matrices. In reality, many LDPC codes are highly structured and have semi-regular connections (including the WiMAX family of LDPC codes and 10GBASE-T LDPC code). In the future, this structure should be considered when designing digit-online decoders. Structured codes provide a natural grouping of nodes into localized groups which can be exploited to reuse logic. Node groups could share logic for operations such as generating delayed versions of R signals and generating saturation values.

The digit-online designs considered so far have been fully register-based, but energy efficiency could be improved by replacing shift registers with RAMs. The input buffer for channel values in the controller would be simple to replace with RAM. In structured codes, variable node groups could each have a block RAM for channel values and share logic to generate the appropriate addresses. Since channel LLRs are required in a bit-serial manner, each word in memory would hold one bit from each of several channel LLRs with the number of words corresponding to the decoding pipeline length. Fig. 8.1 shows the local variable node memory in both the architecture covered in this thesis (Fig. 8.1(a)) and the proposed RAM-based system (Fig. 8.1(b)).

The results in [18] show that post-processing of LLRs is a good way of removing the error floor for decoders which is necessary for codes which operate at extremely low bit error rates, such as the 10GBASE-T code. Post-processing is performed on codewords that do not converge within the iteration limit. At high SNRs this is usually due to absorbing sets, or patterns of bits that cannot be successfully decoded with message-passing decoding [18]. This is because the variable nodes in the set tend to reinforce incorrect values

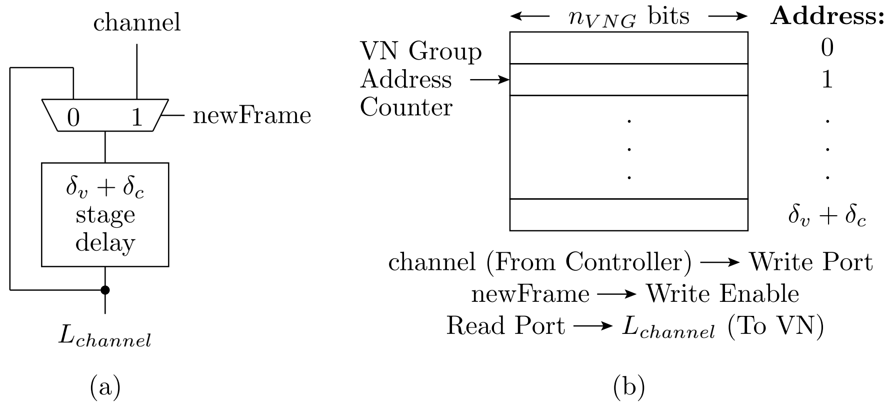


Figure 8.1: *(a) Existing architecture and (b) proposed RAM-based architecture for local variable node memory assuming n_{VNG} variable nodes are grouped together. The full digit-online variable node appears in Fig. 5.5.*

through cycles in the graph (messages from unsatisfied check nodes from outside the set are overpowered by messages from satisfied check nodes inside the set that reinforce the error). In an attempt to correct this, the LLRs are decreased in magnitude at the satisfied check nodes (or increased in magnitude at unsatisfied checks). Then, extra decoding iterations are performed to see if decoding converges after that point. Since the proportion of codewords that fail to converge at high SNR is very small, there is almost no effect on average throughput, but [18] shows that there is a significant effect on error rate.

Post-processing could be implemented in our digit-online architecture by changing the offset sent to the check nodes. Check nodes would be modified to have another control input that bypasses the offset subtraction. If decoding fails to converge before the iteration limit is reached, the decoding controller would send out an alternate offset that would be applied only at satisfied check nodes. Decoding could then continue as usual for another set number of iterations.

This thesis has only considered fully-parallel implementations, but most published ASIC implementations are partially parallel. Further work is needed to adapt the digit-online architecture to partially node-parallel decoding. Developing a partially parallel digit-online architecture would also work towards multi-mode digit-online decoders. When considering partially parallel and multi-mode decoders, a digit-online architecture should have an advantage over bit-parallel decoders because it has a fixed wiring cost with respect to precision. This would result in smaller permutation circuits in a digit-online decoder.

Since digit-online decoders have lower local and global routing demands than bit-parallel decoders, they may be well suited to FPGAs, which have limited interconnect resources. Many FPGAs also have different sized RAMs distributed across the chip, which would be well-suited for replacing some of the registers. Future work should determine whether digit-online decoding is appropriate for FPGAs. Possible drawbacks are the higher clock frequencies needed and the required density of sequential cells.

Finally, our work with frame-interlacing has assumed two frames of equal precision. Alternate frame-interlaced architectures are possible, including using low-precision operands for the first half of decoding, and then allowing the operands to grow into a precision for the last half of the decoding iterations. This would increase the controller complexity (currently 4% of the area for the 10GBASE-T design), but would not require changes to the rest of the decoder. This variable-precision architecture may allow for good error performance with lower total pipeline length than if both frames were high-precision [55]. While the current architecture can ignore uninitialized data in the pipeline, this new architecture would need to be more careful about resetting values as the operands grew. This sort of decoding may also be incompatible with early termination.

8.3 Publications Arising from Thesis

An 11-page paper containing an explanation of the digit-online LDPC decoding architecture and post-layout results from the 10GBASE-T LDPC ASIC entitled ”Deeply Pipelined Digit-Serial LDPC Decoding” [46] has been accepted for publication in Transactions on Circuits and Systems I – Regular Papers. Its DOI is 10.1109/TCSI.2012.2206461.

A 6-page paper entitled ”Effects of Varying Message Precision in Digit-Online LDPC Decoders” [56] detailing the precision scaling of digit-online LDPC decoders was accepted to the 2012 IEEE Workshop on Signal Processing Systems.

Bibliography

[1] C. E. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal*, vol. 27, pp. 379–423,623–656, Jul. and Oct. 1948.

[2] R. Gallager, “Low-density parity-check codes,” *IEEE Trans. on Information Theory*, vol. 8, no. 1, pp. 21–28, January 1962.

[3] X. Huang, “Near perfect decoding of LDPC codes,” in *Proceedings of the 2005 International Symposium on Information Theory (ISIT 2005)*, September 2005, pp. 302–306.

[4] R. M. Tanner, “A recursive approach to low complexity codes,” *IEEE Trans. on Inf. Theory*, vol. 27, pp. 533–547, September 1981.

[5] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, “Improved low-density parity-check codes using irregular graphs,” *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 585–598, Feb. 2001.

[6] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, “Design of capacity-approaching irregular low-density parity-check codes,” *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, February 2001.

[7] S.-Y. Chung, J. G. David Forney, T. J. Richardson, and R. Urbanke, “On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit,” *IEEE Comm. Letters*, vol. 5, no. 2, pp. 58–60, February 2001.

[8] S. L. Howard, C. Schlegel, and V. C. Gaudet, “Degree-matched check node decoding for regular and irregular LDPCs,” *IEEE Trans. Circuits and Systems II: Express Briefs*, vol. 53, no. 10, October 2006.

[9] C. Hartman and L. Rudolph, “An optimum symbol-by-symbol decoding rule for linear codes,” *IEEE Trans. Inf. Theory*, vol. 22, no. 5, pp. 514–517, Sep. 1976.

[10] G. Battail, M. Decouvelaere, and P. Godlewski, “Replication decoding,” *IEEE Trans. Inf. Theory*, vol. 25, no. 3, pp. 332–345, May 1979.

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

[12] A. J. Blanksby and C. J. 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, March 2002.

[13] T. Zhang, Z. Wang, and K. K. Parhi, "On finite-precision implementation of low density parity check codes decoder," in *The 2001 International Symposium on Circuits and Systems 2001 (ISCAS 2001)*, May 2001, vol. 4, pp. 202–205.

[14] C. Zhang, Z. Wang, J. Sha, L. Li, and J. Li, "Flexible LDPC decoder design for multigigabit-per-second applications," *Circuits and Systems I: Regular Papers, IEEE Transactions On*, vol. 57, pp. 116–124, January 2010.

[15] Y. Sun and J. R. Cavallaro, "A low-power 1-Gbps reconfigurable ldpc decoder design for multiple 4g wireless standards," in *2008 IEEE International SOC Conference*, September 2008, pp. 367–370.

[16] A. Anastasopoulos, "A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," in *IEEE Global Telecommun. Conf. 2001*, vol. 2, pp. 1021–1025.

[17] J. Chen and M. P. Fossorier, "Density evolution for BP-based decoding algorithms of LDPC codes and their quantized versions," in *IEEE Global Telecommun. Conf. 2002*, Nov. 2002, vol. 2, pp. 1378–1382.

[18] Z. Zhang, V. Anatharam, M. J. Wainwright, and B. Nikolic, "An efficient 10GBASE-T Ethernet LDPC decoder design with low error floors," in *IEEE J. Solid-State Circuits*, Apr. 2010, vol. 45, pp. 843–855.

[19] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "A bit-serial approximate min-sum LDPC decoder and FPGA implementation," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2006)*, 2006, pp. 149–152.

[20] M. M. Mansour and N. R. Shanbhag, "Low-power VLSI decoder architectures for LDPC codes," in *Proceedings of the 2002 International Symposium on Low Power Electronics and Design (ISLPED 2002)*, 2002, pp. 284–289.

[21] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "Multi-Gbit/sec low density parity check decoders with reduced interconnect complexity," *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2005)*, vol. 5, pp. 5194–5197, 2005.

[22] Y.-L. Ueng, C.-J. Yang, K.-C. Wang, and C.-J. Chen, "A multimode shuffled iterative decoder architecture for high-rate RS-LDPC codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 10, pp. 2790–2803, Oct. 2010.

[23] H. Zhong and T. Zhang, "Block-LDPC: a practical LDPC coding system design approach," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 4, pp. 766–775, Apr. 2005.

[24] Y. Chen and K. K. Parhi, "Overlapped message passing for quasi-cyclic low-density parity check codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 6, pp. 1106–1113, Jun. 2004.

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

[26] T. Brack, M. Alles, F. Kienle, and N. Wehn, "A synthesizable IP core for WiMAX 802.16e LDPC code decoding," in *IEEE 17th Int. Symp. on Personal, Indoor and Mobile Radio Commun.*, Sep. 2006, pp. 1–5.

[27] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "Power reduction techniques for LDPC decoders," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, pp. 1835–1845, Aug. 2008.

[28] T. Brandon, R. Hang, G. Block, V. C. Gaudet, B. Cockburn, S. Howard, C. Giasson, K. Boyle, P. Goud, S. S. Zeinoddin, A. Rapley, S. Bates, D. Elliott, and C. Schlegel, "A scalable LDPC decoder ASIC architecture with bit-serial message exchange," *Integration VLSI J.*, vol. 41, pp. 385–39, May 2008.

[29] S. S. Tehrani, A. Naderi, , G.-A. Kamendje, S. Hemati, S. Mannor, and W. J. Gross, "Majority-based tracking forecast memories for stochastic LDPC decoding," *IEEE Trans. Signal Process*, vol. 58, no. 9, pp. 4883–4896, Sep. 2010.

[30] A. Naderi, S. Mannor, M. Sawan, and W. J. Gross, "Delayed stochastic decoding of LDPC codes," *IEEE Trans. Signal Process*, vol. PP, no. 99, Aug. 2011.

[31] R. M. Owens, "Techniques to reduce the inherent limitations of fully digit on-line arithmetic," *IEEE Transactions on Computers*, vol. C-32, no. 4, pp. 406–411, April 1983.

[32] N. Onizawa, T. Ikeda, T. Hanyu, and V. C. Gaudet, "3.2-Gb/s 1024-b rate-1/2 LDPC decoder chip using a flooding-type update-schedule algorithm," in *50th Midwest Symposium on Circuits and Systems 2007 (MWSCAS 2007)*, August 2007, pp. 217–220.

[33] T. Brack, M. Alles, T. Lehnigk-Emden, F. Kienle, and N. Wehn, "Low complexity LDPC code decoders for next generation standards," in *Design, Automation and Test in Europe Conference and Exhibition 2007 (DATE 2007)*, April 2007, pp. 1–6.

[34] C.-C. Lin, K.-L. Lin, H.-C. Chang, and C.-Y. Lee, "A 3.33Gb/s (1200,720) low-density parity check decoder," in *Proceedings of the 31st European Solid-State Circuits Conference 2005 (ESSCIRC 2005)*, September 2005, pp. 211–214.

[35] X.-Y. Shih, C.-Z. Zhan, and A.-Y. Wu, "A 7.39mm2 76mW (1944,972) LDPC decoder chip for IEEE 802.11n applications," in *IEEE Asian Solid-State Circuits Conference 2008 (A-SSCC 2008)*, November 2008, pp. 301–304.

[36] X.-Y. Shih, C.-Z. Zhan, C.-H. Lin, and A.-Y. Wu, "An 8.29 mm2 52mW multi-mode LDPC decoder design for mobile WiMAX system in 0.13 um CMOS process," *Solid-State Circuits, IEEE Journal of*, vol. 43, pp. 672–683, March 2008.

[37] K. K. Gunnam, G. S. Choi, and M. B. Yeary, "A parallel VLSI architecture for layered decoding for array LDPC codes," in *20th International Conference of VLSI Design 2007. Held Jointly with 6th International Conference on Embedded Systems*, January 2007, pp. 1063–1068.

[38] J. Sha, Z. Wang, M. Gao, and L. Li, "Multi-Gb/s LDPC code design and implementation," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions On*, vol. 17, pp. 262–268, February 2009.

[39] G. Baccarani, M. R. Wordeman, and R. H. Dennard, "Generalized scaling theory and its application to a 1/4 micrometer MOSFET design," *IEEE Transactions on Electronic Devices*, vol. ED-31, pp. 452–462, April 1984.

[40] M. J. Irwin and R. M. Owens, "Digit-pipelined arithmetic as illustrated by the Paste-Up system: A tutorial," *Computer*, vol. 20, no. 4, pp. 61–73, April 1987.

[41] M. Ercegovac, "Online arithmetic: An overview," in *Proceedings of SPIE V.495: Real Time Signal Processing VII*, 1984, pp. 86–93.

[42] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Transactions on Electronic Computers*, vol. EC-10, no. 3, pp. 389–400, September 1961.

[43] M. D. Ercegovac and T. Lang, *Digital Arithmetic*, Morgan Kaufmann, 2004.

[44] M. A. Thornton, "Signed binary addition circuitry with inherent even parity outputs," *IEEE Transactions on Computers*, vol. 46, pp. 811–816, July 1997.

[45] Y. Harata, Y. Nakamura, H. Nagase, M. Takigawa, and N. Takagi, "A high-speed multiplier using a redundant binary adder tree," *IEEE Journal of Solid-State Circuits*, vol. SC-22, pp. 28–34, February 1987.

[46] P. A. Marshall, V. C. Gaudet, and D. G. Elliott, "Deeply pipelined digit-serial LDPC decoding," *IEEE Trans. Circuits Syst. I, Reg. Papers*, Accepted for publication 2012, 11 pages, DOI: 10.1109/TCSI.2012.2206461.

[47] M. D. Ercegovac and T. Lang, "On-the-fly conversion of redundant into conventional representations," *IEEE Transactions on Computers*, vol. C-36, no. 7, pp. 895–897, July 1987.

[48] D. E. Knuth, *The Art of Computer Programming, Vol. 3*, chapter 5.3.4, pp. 225–229, Addison-Wesley, second edition, 1998.

[49] J. M. Gamble, "Sorting networks," `http://pages.ripco.net/~jgamble/nw.html`.

[50] T.-C. Kuo and A. N. Willson, "A flexible decoder IC for WiMAX QC-LDPC codes," in *Proc. 2008 IEEE Custom Integrated Circuits Conf.*, Sep. 2008, pp. 527–530.

[51] C.-H. Liu, S.-W. Yen, C.-L. Chen, H.-C. Chang, C.-Y. Lee, Y.-S. Hsu, and S.-J. Jou, "An LDPC decoder chip based on self-routing network for IEEE 802.16e applications," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, pp. 684–694, Mar. 2008.

[52] X. Peng, Z. Chen, X. Zhao, D. Zhou, and S. Goto, "A 115mW 1Gbps QC-LDPC decoder ASIC for WiMAX in 65nm CMOS," in *2011 IEEE Asian Solid State Circuits Conf.*, Nov. 2011, pp. 317–320.

[53] Y.-L. Wang, Y.-L. Ueng, C.-L. Peng, and C.-J. Yang, "A low-complexity LDPC decoder architecture for WiMAX applications," in *2011 Int. Symp. on VLSI Design, Automation and Test*, Apr. 2011, pp. 1–4.

[54] X.-Y. Shih, C.-Z. Zhan, C.-H. Lin, and A.-Y. Wu, "An 8.29 mm^2 52 mW multi-mode LDPC decoder design for mobile WiMAX system in 0.13 m CMOS process," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, pp. 672–683, Mar. 2008.

[55] S. Li, "Power characterization of a Gbit/s FPGA convolutional LDPC decoder," M.S. thesis, University of Waterloo, Aug. 2012.

[56] P. A. Marshall, V. C. Gaudet, and D. G. Elliott, "Effects of varying message precision in digit-online LDPC decoders," in *IEEE Workshop on Signal Processing Systems 2012*, Accepted for publication 2012, 6 pages.

Appendix A

Addition Circuits

This appendix provides the full logic for the different addition circuits used in the digit-online LDPC decoders. The logic for performing the actual addition operations is explored in Section 3.3.1. The explanations in this appendix focus on the additional logic required to break the carry chains between successive operations. In any given stage of an adder, the aim is to set the carry-out to the next stage to zero when the MSD is present in said stage. The input *R* to each of the adders is an input vector marking which digits are the MSDs of a value. For any input *x* and time *i*, x_i is the MSD of *X* if and only if $r_i = 1$.

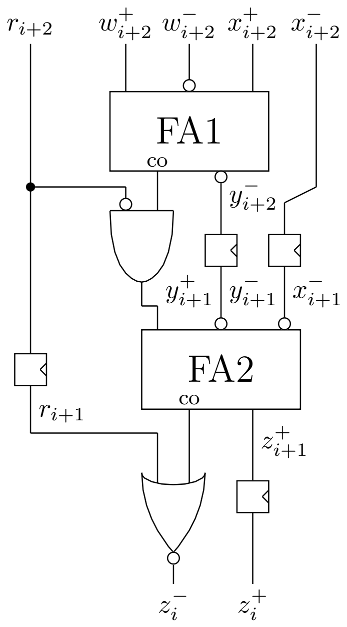


Figure A.1: *Circuit for digit-online addition of two signed-binary numbers W and X with a signed-binary result Z.*

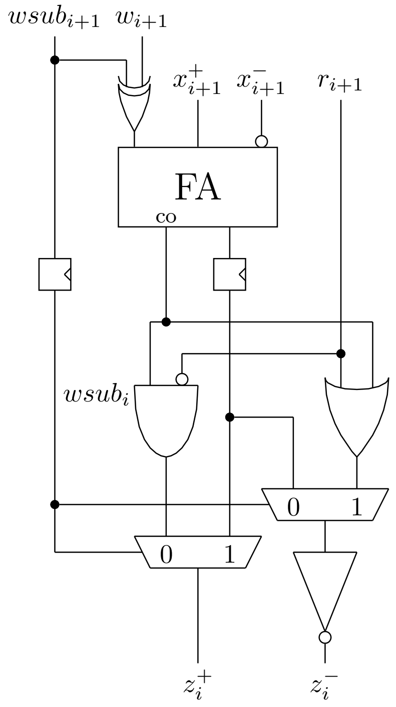


Figure A.2: *Circuit for digit-online addition of a signed-binary number X and a conventional binary value W with a signed-binary result Z. Input wsub determines whether W is subtracted or added and must remain constant for the entire operation.*

Fig. A.1 is used to add two signed-binary values, and has the most straightforward logic for breaking carry chains. When r_{i+2} is high (i.e., the MSDs of W and X are present in the first stage), we wish to avoid a carry-in to FA2. This is easily accomplished by ANDing y_{i+1} with $\overline{r_{i+2}}$. When r_{i+1} is high, the carry-out from FA2 must be blocked. Since this carry-out must be inverted to give the correct z_i^-, r_{i+1} can be NORed with the carry out to yield z_i^-.

Fig. A.2 is used to add or subtract a conventional unsigned value from a signed-binary value, with a signed-binary result. To understand the logic to prevent carry-outs from the adder, we must look at the two seperate cases for *wsub*. During addition ($wsub_i = 0$), the carry-out of the full adder is used for z_i^+. In this case the logic for the carry-out from FA1 in Fig. 5.7 can be used. When $wsub_i = 1$, the carry-out of the full adder is inverted to give z_i^-. Thus the logic for the carry-out from FA2 in Fig. 5.7 can be used.

If *wsub* in Fig. A.4 is fixed to 1, the circuit can be simplified to yield Fig. A.3 which is used in the check nodes to subtract the offset (which is broadcast in unsigned-binary). The addition logic is the same as the circuit in Fig. 3.6(b).

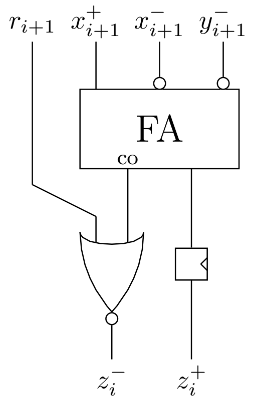


Figure A.3: *Circuit for subtracting the offset X in conventional unsigned-binary format from a signed-binary number Y.*

Fig. A.4 shows the circuit for additing or subtracting a conventional binary number from the sum of two signed-binary numbers. The second stage of addition (FA3) is identical to Fig. A.2, and thus carry-out from that stage is handled in an identical manner. For the first stage of addition (FA1 and FA2), the situation is slightly more complex.

If $wsub_{i+2} = 1$, FA1 has two negative inputs and one positive input. From Fig. 3.6 we see that this means the carry out is the negative half-digit and the sum output is the positive half-digit. This gives FA2 two positive half-digits, which means that the carry-out is the positive half-digit.

When $wsub_{i+2} = 0$, the situation is reversed. FA1 has two positive inputs which makes the carry output the positive half-digit and the sum the negative half-digit. This gives FA2 two negative half-digit inputs and makes its carry-out the negative half-digit.

In either case, one of the two half-digits between stages is positive and the other is negative. Since the positive half-digits have an active-high convention and the negative half-digits have an active-low convention, one of the two carry outs from FA1 and FA2 must be 1 and the other must be zero. The AND and OR gates on the inputs of FA3 ensure that this is the case.

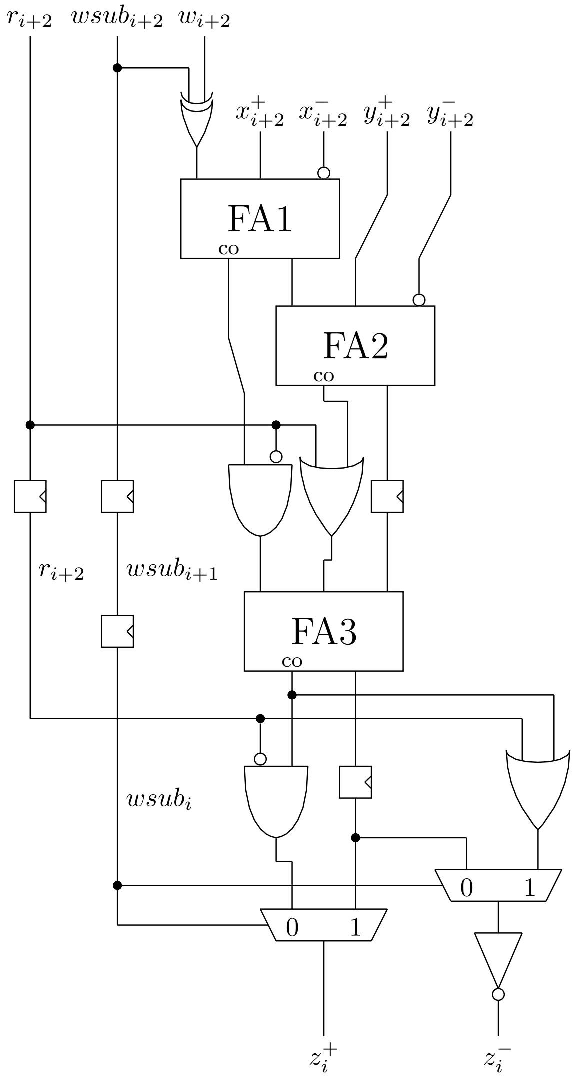


Figure A.4: *Circuit for digit-online addition of two signed-binary numbers X and Y and a conventional binary value W with a signed-binary result Z. Input wsub determines whether W is subtracted or added and must remain constant for the entire operation.*

Appendix B

Variable Node Equations

This appendix details the addition trees used in the variable node designs used to produce Table 4.1 and in all ASIC designs. Each set of parentheses represents a single addition operation with the type of addition (3:2, 4:2 or 5:2) implicit from the operands. Node inputs λ_i and outputs L_i are two-bit-per-digit signed-binary values while the channel input c is a one-bit-per-digit value. The individual bits of a signed-binary digit d are denoted as d^+ and d^-. The logic implementation of the addition circuits are given in Appendix A.

Variable node equations for $d_v = 2$:

$$\begin{aligned} \lambda_0 &= (L_1 + c) \\ \lambda_1 &= (L_0 + c) \end{aligned} \tag{B.1}$$

Variable node equations for $d_v = 3$:

$$\begin{aligned} \lambda_0 &= (L_1 + L_2 + c) \\ \lambda_1 &= (L_0 + L_2 + c) \\ \lambda_2 &= (L_0 + L_1 + c) \end{aligned} \tag{B.2}$$

Variable node equations for $d_v = 4$:

$$\begin{aligned} \lambda_0 &= (L_1 + (L_2 + L_3) + c) \\ \lambda_1 &= (L_0 + (L_2 + L_3) + c) \\ \lambda_2 &= ((L_0 + L_1) + L_3 + c) \\ \lambda_3 &= ((L_0 + L_1) + L_2 + c) \end{aligned} \tag{B.3}$$

Variable node equations for $d_v = 5$:

$$\begin{aligned}
\lambda_0 &= ((L_1 + L_2) + (L_3 + L_4) + c) \\
\lambda_1 &= ((L_0 + L_2) + (L_3 + L_4) + c) \\
\lambda_2 &= ((L_0 + L_1) + (L_3 + L_4) + c) \\
\lambda_3 &= ((L_0 + L_1) + (L_2 + L_3) + c) \\
\lambda_4 &= ((L_0 + L_1) + (L_2 + L_4) + c)
\end{aligned} \tag{B.4}$$

Variable node equations for $d_v = 6$:

$$\begin{aligned}
\lambda_0 &= ((L_1 + L_2 + L_5^+) + (L_3 + L_4 + L_5^-) + c) \\
\lambda_1 &= ((L_0 + L_2 + L_5^+) + (L_3 + L_4 + L_5^-) + c) \\
\lambda_2 &= ((L_0 + L_1 + L_5^+) + (L_3 + L_4 + L_5^-) + c) \\
\lambda_3 &= ((L_0 + L_1 + L_2^+) + (L_2^- + L_4 + L_5) + c) \\
\lambda_3 &= ((L_0 + L_1 + L_2^+) + (L_2^- + L_3 + L_5) + c) \\
\lambda_3 &= ((L_0 + L_1 + L_2^+) + (L_2^- + L_3 + L_4) + c)
\end{aligned} \tag{B.5}$$

Appendix C

Synthesis and Place-and-Route Methodologies

This appendix explains our synthesis and place-and-route methodology for experiments that compared digit-online and bit-parallel architectures. To ensure a fair comparison, the synthesis and place-and-route scripts were kept as similar as possible across all decoder designs.

Synthesis was done with identical synthesis parameters, optimization levels, and compilation options. Automatic register retiming was used for all designs. Synthesis scripts differed only in the range of clock frequencies targeted, as the achievable clock frequencies vary greatly with architecture and precision. For each decoder, the design with the highest throughput/area was selected for inclusion in the thesis. All logic synthesis was performed in Synopsys Design Compiler D-2010.03-SP5.

For post-layout results, a layout for each decoder was generated at initial densities of 80%, 85%, and 90%. The placement and routing scripts were identical for each decoder, with only the netlist and timing goals being decoder-specific. In most cases the achievable timing decreased at higher placement densities. Therefore, the design of the three with the highest throughput/area was chosen to present in this thesis. Timing-driven placement with pre-place and post-place optimizations was used. Clock tree specifications were generated automatically from the design with identical parameters. Two stages of optimization were performed after clock tree synthesis. Timing-driven routing and two stages of post-route optimization were used. These levels of optimization were chosen because in both cases the second stage still improved timing significantly, while the third stage made very little difference. All optimization settings (and any other unspecified settings) were the same for all designs. All placement, routing, and power estimation was performed with Cadence

Encounter EDI 9.1. Mentor ModelSim 6.6c was used to generate net activity information for power estimation.

Appendix D

Place-and-Route with Cadence Encounter

This appendix provides a general tutorial of achieving good placement and routing results in Cadence Encounter. This tutorial is aimed at designers that already have a basic knowledge of placement and routing of a design, and focuses on the key steps and parameters to optimize designs. Since command syntax changes frequently with new Encounter versions, information will be presented in as general a way as possible. The specific commands presented in the appendix are for EDI 9.1 and are typeset in **bold text**.

The design flow presented assumes a fully digital design in a small-nm design process with at least 4 metal layers suitable for routing. Many of the parameters depend on the type of design being fabricated. An effort will be made to explain the tradeoffs involved in each decision. Place-and-Route is a much more iterative process than synthesis. It is often necessary to evaluate results, tune parameters, and then re-run placement and routing from the beginning.

The rest of this appendix explains the following general steps to do a full place-and-route:

1. Design Setup
2. Placement and Post-Place Optimization
3. Clock Tree Synthesis (CTS) and Post-CTS Optimization
4. Routing and Post-Routing Optimization
5. Verification
6. Post-Layout Power Measurements

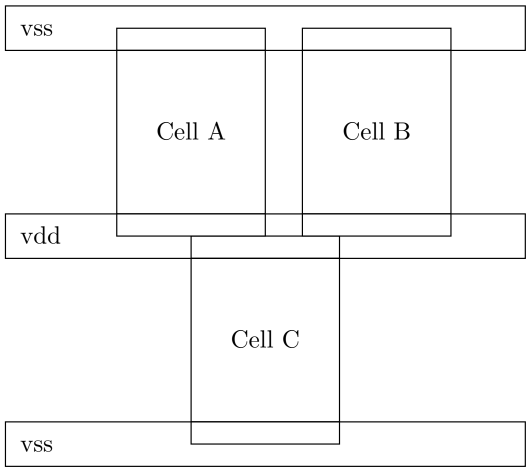


Figure D.1: *Back-to-back placement of standard cell rows.*

D.1 Design Setup

Floor Planning

Modern cell libraries for small-nm processes are designed for back-to-back placement, as shown in Fig. D.1. This removes the requirement to leave space between rows of standard cells. The placement tool will automatically flip or rotate instances as necessary to match the power rails correctly. Ensure that Encounter is set to back-to-back rows with zero spacing in order to maximize logic density. Metal strips will be routed over the cell power pins in order to ensure that all cells receive power even when there are gaps between them.

The ratio of standard cell area to core area (density) is a parameter that must be chosen carefully. While higher densities make more efficient use of silicon area, they also increase routing congestion. This can lead to unroutable designs, or designs with unnecessarily poor clock frequencies due to wires that require a non-direct path. A good rule-of-thumb is to start with 80% density and make adjustments based on the final design's routability and wiring congestion. Density tends to increase through the process of place-and-route due to the addition of the clock tree and up-sizing of gates during optimization steps. Therefore, the density of the final design is usually higher than the initial density chosen for floor planning. If the final density is lower than the initial density, this implies that the clock period can be decreased.

Designs with initial densities above 85% may be difficult to close timing on even if routing completes. They may require more adjustment of optimization parameters and more design iterations than usual.

Power Planning and Special Route

In most designs, the power planning stage consists of defining global nets, adding power stripes and rings, and then running special route (SROUTE). The size of power rings and stripes vary greatly with fabrication technology and design size and are beyond the scope of this tutorial. Before adding power rings or stripes, the global power and ground nets must be set up as usual.

Adjacent metal layers are routed in opposite directions with odd metal layers routed east-west, and even layers routed north-south. This allows metal 1 stripes to be placed along the power rails of standard cells, and metal 2 wires to route to standard cell pins. Metal 1 is generally unsuitable for routing as it is densely used in standard cells. Occasionally Encounter will use Metal 1 for short jogs to a cell pin.

Routing congestion is higher in lower metal layers because they provide the shortest path between standard cell pins and Encounter will prefer to use them. For this reason it is important to leave as much open space in low metal layers as possible.

Power rings should be routed in metal 2 on the left and right and metal 1 on top and bottom. This allows for the shortest connection between standard cell power pins and power rings without interfering with routing.

Placing power stripes in Metal 2 will block placement of standard cells beneath them and provide routing difficulties. However, every layer of vias between the power rings and stripes will increase the resistance and decrease the effectiveness of the stripes. For these reasons, there should be power stripes in Metal 4. If required, additional power stripes can be added to higher metal layers and connected to lower layers in a mesh.

Once power rings and stripes are in place, SROUTE is invoked to connect power nets and insert power buses for standard cells. The default parameters are usually fine, but be sure to specify which nets you wish SROUTE to connect.

Optimization Settings

Optimization settings in Encounter are set using the **setOptMode** command. The best optimization settings will be different at various stages and should be set before each stage of optimization. Remember to set optimization settings before commands that include an optimization step, such as doing placement with pre-place or in-place optimizations.

The rest of this section provides a basic introduction to the most important parameters to the **setOptMode** command. This list is not exhaustive, and designers should refer to its manual page by typing ”man setOptMode” from the Encounter command-line.

setOptMode -reset returns all optimization settings to their default.

setOptMode -effort [*low*|*medium*|*high*]
Sets how hard Encounter tries to optimize your design. High effort is recommended except during prototyping runs.

setOptMode -setupTargetSlack <*value*>
setOptMode -setupHoldSlack <*value*>
Unless there is a great deal of slack in the synthesized design, the clock period will need to be increased before physical placement in order to close timing. Additionally, the achievable clock period will change at the different stages of placement and routing. For most designs, a single timing constraint (sdc) file can be used throughout the entire place-and-route and the settings for extra setup time slack can be adjusted as necessary throughout the process. Likewise, use hold slack to suppress hold-time violations that disappear after routing. Encounter sets default extra setup slack values for certain early design stages, but they are generally not sufficient for high-performance designs and should be overridden. Once placement and routing is run once, the designer should have a good sense of how much setup slack is needed at early stages so that post-route timing is still met.

setOptMode -maxDensity <*value*>
Encounter has a maximum density setting that it will not exceed at any point during optimization. The closer the design density comes to the maximum density, the less effective optimization will be. If Encounter cannot remain below the maximum density, it may stop the current optimization and/or leave instances in invalid locations (such as overlapping other instances). For designs with an initial density of above 80%, the maximum density should be increased from its default of 0.95 to 0.99.

setOptMode -criticalRange <*value*>
Critical Range is a parameter on $[0,1]$ that determines which paths should be optimized. A critical range of 0 will optimize *only* the critical path, and a critical of 1.0 will make all paths eligible for optimization. If timing is not met during optimization, better results can

be achieved by increasing the critical range from its default of 0.4 to within the range of 0.5-0.7. Generally, using a smaller value in early optimization steps and a larger one in later steps will result in the best compromise between running time and clock frequency. Smaller values provide a smaller worst negative slack, while larger values provide a smaller total negative slack. Therefore, increasing the critical range too much (beyond 0.7) is not advisable as it will greatly increase run time and can cause poor results if optimizing already short paths makes it difficult to optimize longer paths.

setOptMode -reclaimArea $[true|false]$
This setting determines whether Encounter attempts to down-size gates during optimization to decrease the density of the design. At lower placement densities, disabling area reclamation during placement will make post-placement optimization run faster. However, at higher densities this will cause the density limit to be reached during post-placement optimization.

setOptMode -usefulSkew $[true|false]$
If this setting is true, Encounter attempts to exploit clock skew to improve performance. In my experience this setting is of dubious use. It is meaningless before clock tree insertion, and clock skew changes so much during routing that it isn't helpful for post-CTS optimization. It may provide a slight boost to post-route optimization. Enabling usefulSkew *greatly* increases CPU time for optimization.

D.2 Placement

Placement options are set with **setPlaceMode** and placement is performed with **placeDesign**.

Timing changes considerably once the clock tree is introduced, so it is not beneficial to run more than one post-placement optimization step. Remember that density will increase during clock tree synthesis, so the density after post-placement optimization must be sufficiently less than the maximum density. The exact margin depends on the size of the clock tree, which generally increases with the number of sequential elements and the target operating frequency.

setPlaceMode -timingDriven $[true|false]$
placeDesign (-prePlaceOpt) (-inPlaceOpt)
A timing-driven placement with pre-place and in-place optimizations is required to achieve good results with high-performance designs. This can take considerably longer than without

these optimizations enabled, but the resulting timing will be much better and subsequent optimization stages will take less time.

setPlaceMode -congEffort $[low|medium|high]$
setPlaceMode -doCongOpt $[true|false]$
For designs with routing congestion, enabling congestion-aware placement may be necessary. This is achieved by setting doCongOpt true and setting the desired congEffort. Medium congestion effort is recommended as a starting point.

setPlaceMode -modulePadding $<module>$ $<factor>$
It is also possible to allow extra space in certain modules or around certain cells if the congestion is localized. To do this, specify which $<module>$ you would like Encounter to view as $<factor>$ times larger for the purpose of placement. The $<module>$ specification supports wildcards.

setPlaceMode -maxRouteLayer $<value>$
When routing a design with no IO pads, setting the maxRouteLayer to below the top layer will prevent Encounter from using the top metal layers for pins. This is desirable since top metal layers are usually thicker with larger design rules and therefore unsuitable for use as a routing layer.

D.3 Clock Tree Synthesis

A clock tree is required for good performance in all digital designs. However, very little designer effort is required to achieve this since Encounter does virtually everything automatically.

Multiple stages of post-CTS optimization can have mixed results. Post-CTS timing does not always correlate with post-route timing. The degree of post-CTS optimization that is appropriate must be evaluated on a per-design basis.

addCTSCellList {***list of inverter/buffer cells***}
Specify which inverters and buffers Encounter can choose from for the clock tree. Be sure to include clock buffers and clock inverters in the cell list (you may wish to use *only* these cells). They are cells that are optimized for clock delay, but do not always appear with the standard logic cells. They may even be in a separate library/LEF file.

clockDesign -genSpecOnly <*ClockTreeSpecFile*>
Generates a clock tree specification based on your designs timing constraints. In most cases Encounter generates a clock tree specification that does not need to be altered. However, you can edit the clock tree specification file if you wish to set different goals for parameters such as maximum skew or transition time.

clockDesign -specFile <*ClockTreeSpecFile*> **-outDir** <*ClockReportDirectory*>
Loads the clock tree specification file and synthesizes the clock tree. A clock report will be generated in <*ClockReportDirectory*> reporting the specifications of the clock tree (which may or may not have met the goals set in the clock tree specification).

D.4 Routing

Routing in Encounter is run in three distinct phases. Encounter reports the number of design rule errors (shorts, open nets, or physical design rule violations) at each iteration of each phase.

In the first phase, nets are routed without much regard for shorts or spacing rules. It is not uncommon to see tens to hundreds of thousands of design rule errors, especially in the early iterations. The second phase focuses more on fixing these errors, and by the final iteration most wires should be fixed in place. The final phase is a clean-up phase which is only intended to fix the last few (generally fewer than 1000) design rule errors. The final stage may not even run if there are too many design rule errors.

If there are design rule errors remaining after routing stage, it is generally not helpful to increase the number of routing iterations. It is much more effective to decrease the initial density and re-place if Encounter is unable to successfully route your design.

Whether or not timing-driven routing is worthwhile depends greatly on the design. It can greatly increase the run time of routing and in some cases makes very little difference to timing. It should be enabled only if the timing gets much worse between the start and end of routing. Timing-driven routing provides more benefit in denser or more highly congested designs.

Optimization has the least room to work post-route. This means that the negative slack must be very small before optimization begins (less than 100-150 ps is a good rule-of-thumb for 65 nm). Subsequent postRoute optimization steps can occasionally make timing worse, so save the design after each optimization.

setNanoRouteMode -routeTopRoutingLayer $<value>$
Sets the top metal layer used for routing. You may wish to exclude top metal layers intended for power routing.

setNanoRouteMode -routeWithTimingDriven $[true|false]$
Sets whether timing-driven routing is enabled.

setNanoRouteMode -routeTdrEffort $<value>$
An integer value from 0-10 specifying the amount of effort used to meet timing. This does not usually need to be set as the default is generally good.

routeDesign -globalDetail
Perform a global detail (full) route of the design with the previously specified parameters.

D.5 Verification

After routing, the routing must be double-checked using **verifyConnectivity** and design rules must be checked with **verifyGeometry**. Generally these commands do not need any arguments.

Running verification steps in Encounter is *not* a substitute for proper DRC and LVS. Due to limitations of the file formats used by Encounter and its verification tools, Encounter does not check all physical design rules and may not properly check that all nets are connected. If the design is to be fabricated, DRC and LVS must be run in another tool (usually Calibre or Diva, depending on the technology) before tape-out. However, these checks will save time by allowing the designer to catch many errors early.

If Encounter is consistently generating designs that pass **verifyGeometry** but have design rule errors that show in other tools, the only recourse is often to add or modify design rules in the technology LEF file. There is full documentation of the LEF file format in the Cadence documentation (cdsdoc or cdshelp).

D.6 Post-Layout Power Measurements

In order to get accurate power measurements, the designer must provide an accurate RC file (which should be loaded in your design configuration file) and an activity file. The activity file is generally a Value Change Dump (VCD) file which can be generated from an

HDL simulator such as ModelSim. Since the clock frequency of the activity file cannot be overridden, the simulation must be performed at the post-layout clock frequency for valid results.

read_activity_file -format VCD -vcd_scope $<DUThierarchy>$ $<VCDfile>$
Loads an activity file for accurate power estimation. The hierarchy to the device-under-test (your current design) must be set with $<DUThierarchy>$. This usually looks something like testbench_instance_name/DUT_instance_name.

report_power -outfile $<reportFile>$
report_power -net -outfile $<reportFile>$
report_power -leakage -outfile $<reportFile>$
Report total power, switching power, and leakage power, respecively.