A 2.267-Gb/s, 93.7-pJ/bit Non-Binary LDPC Decoder With Logarithmic Quantization and Dual-Decoding Algorithm Scheme for Storage Applications

Yuta Toriyama, *Member, IEEE*, and Dejan Marković, *Member, IEEE*

***Abstract*— Non-binary low-density parity-check (NB-LDPC) codes are a promising class of error-correcting codes that provide excellent coding gain beyond that of their binary counterparts. However, their decoding complexity has thus far limited practicality. We present an NB-LDPC decoder with information throughput of 2.267 Gb/s and power consumption of 212.4 mW, yielding an energy efficiency of 93.7 pJ/bit, implemented in 40-nm CMOS technology. The employed code is long and high-rate without degree-2 variable nodes, resulting in a low error floor and making the code appropriate for storage applications. A logarithmic quantization scheme is proposed to enable aggressive wordlength reduction to alleviate hardware complexity. In addition, a dual-decoding algorithm scheme is employed to alleviate the computational complexity of decoding, realized in an efficient architecture with high parallelism and avoidance of idle clock cycles.**

***Index Terms*— Application-specific integrated circuits (ASICs), digital integrated circuits, energy efficiency, low-density parity-check (LDPC) codes, min–max (MM) decoding algorithm, non-binary, quantization, very large scale integration (VLSI).**

I. Introduction

NON-BINARY low-density parity-check (NB-LDPC) codes defined over a Galois field of order q (GF(q)), where q is a power of 2, are known to have excellent error-correcting capabilities beyond that of their binary counterparts [1]. Unfortunately, NB-LDPC codes are currently limited from practical application by the immense complexity of decoding. Soft decoding algorithms, such as the min–max (MM) decoding algorithm [2], have been introduced as a means of alleviating the decoding complexity while incurring only a slight degradation in the coding gain relative to similar algorithms. The simplifications in the required operations leading to the MM algorithm, as well as other similar variants, have yielded several Application-specific integrated circuit (ASIC) solutions [3]–[6], which have made significant progress in creating practical very large scale integration (VLSI) designs. However, decoders based on soft decoding algorithms remain quite costly and slow relative to binary solutions. On the other hand, hard-decision-based decoding algorithms, such as the iterative hard reliability-based majority-logic decoding (IHRB-MLGD) [7], drastically simplify the required operations in the algorithm and lead to much simpler hardware [8], [9]. However, this simplistic approach yields an unacceptable degradation in the coding gain, nullifying the benefits attracting us to NB-LDPC codes in the first place.

The coding gain of NB-LDPC codes over binary LDPC codes at the cost of increased complexity in the decoder hardware suggests the use of low-field order codes in storage applications [10]–[13]. On one hand, the slight increase in the GF(q) order over which the code is defined can yield significant coding gain benefits that are otherwise very difficult to obtain. As physical device scaling trends decelerate, stronger error correction codes are passionately sought after in order to allow for improved memory densities [10], [11]. In fact, the improvement in error-correcting capabilities is particularly large for storage applications for two important reasons. First, the frame error rate (FER) requirements in storage applications are extremely low (especially relative to other applications such as wireless or wireline communications), since even a single fault signifies the irrecoverable loss of data, and thus device failure. Therefore, while the exact FER may not be observable due to impractically long simulation times, the sharper waterfall curve and lower error floor of NB-LDPC codes will make a much more profound impact on these extremely low-FER regimes. Second, code design optimization techniques that are unique to NB-LDPC codes can tailor these codes specifically for use in storage channels, further improving their performance over currently used binary LDPC codes with similar code parameters [12], [13]. The edge weights of NB-LDPC codes are a powerful, additional degree of freedom in code design that can target problematic

Manuscript received October 17, 2017; revised January 3, 2018, February 26, 2018, and April 25, 2018; accepted April 25, 2018. Date of publication May 22, 2018; date of current version July 20, 2018. This paper was approved by Associate Editor Marian Verhelst. *(Corresponding author: Yuta Toriyama.)*

Y. Toriyama was with the Department of Electrical Engineering, University of California at Los Angeles, Los Angeles, CA 90095 USA. He is now with Qualcomm Technologies, Inc., San Diego, CA 92121 USA (e-mail: ytoriyama@ucla.edu).

D. Marković is with the Department of Electrical Engineering, University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: dejan@ucla.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSSC.2018.2832851

structures in the Tanner graph specific to a particular storage channel (e.g., magnetic and NAND flash) and can be employed with minimal changes in the hardware.

On the other hand, the increased cost in the decoder can potentially be tolerated as long as the increase is moderate, since the coding gain benefits of NB-LDPC may be difficult to otherwise replicate by using binary LDPC codes only, which are already well studied and highly optimized. In applying NB-LDPC codes to storage systems and truly obtaining the coding gain benefits offered by NB-LDPC codes, currently existing solutions for NB-LDPC code decoder architectures, however, are insufficient for three main reasons. First, the GF(q) orders over which the codes are defined for available solutions are too high. GF(32) and above is often chosen with the extended min-sum (EMS) algorithm [14], where only a subset of size $n_m < q$ log-likelihood ratios (LLRs) is kept in each (sorted) message [3]–[6]. While this technique may offer large *relative* savings [for example, $n_m = 16$ for a GF(64) code in [4] or $n_m = 32$ for a GF(256) code in [3]], a non-binary code defined over lower Galois field orders are still much more manageable while providing significant coding gain over binary LDPC codes. Unfortunately, the aforementioned technique does not apply to codes at lower GF(q), because there is no appropriate choice of n_m for a small q that simultaneously maintains good coding gain and saves in hardware complexity, requiring us to pursue unique implementation techniques appropriate for lower GF(q).

Second, NB-LDPC decoder solutions provided thus far only apply to very short codes, which are not appropriate for storage systems. In fact, codes in storage systems are required to be very long (typically 1 kB and above) in order to ensure a sharp waterfall region, low error floor, and lack of undetectable errors, which gives rise to a new set of challenges in implementation, such as the large intermediate message storage requirement. A fully parallel architecture, as employed in [3] and [4], enables high throughput and avoids the intermediate storage altogether but is only possible by sacrificing coding gain due to extremely short codes. Thus, an architectural design enabling high throughput and moderate cost for long code lengths is required.

Third, the column weight of the code, also known as the variable node degree (d_v), cannot be 2. While it is generally accepted that NB-LDPC codes can have lower d_v relative to binary codes, the degenerate case of $d_v = 2$, for example in [4], often gives rise to an observable error floor. Since $d_v = 2$ does simplify the hardware, this parameter choice is potentially more appropriate in other applications, such as wireless communications, that do not require extremely low-FER levels, but is certainly inapplicable to storage systems which, again, requires an ensured low error floor.

In this paper, we present a high-throughput and energy-efficient NB-LDPC decoder chip in 40-nm CMOS suited for storage applications. Several algorithm-level and architecture-level techniques that are intimately tied to each other are proposed and exercised in the chip implementation. The employed code for the proof-of-concept chip is GF(8), code length 9396 bits, and column weight 3, satisfying the requirements for a very low error floor and offering coding gain benefits over binary LDPC codes of similar length and rate. The techniques presented are general and can apply directly to similar GF(8) codes, which would enable highly optimized codes of similar parameters as well as codes of different lengths and rates without a drastic change in architecture. Chip testing results show an information throughput range of 0.544–2.267 Gb/s with a power consumption range of 12.35–212.4 mW, yielding energy efficiencies of 22.7–93.7 pJ/bit by varying the supply voltage and operating clock frequency.

II. Decoding of NB-LDPC Codes

An NB-LDPC code is a linear block code defined by its $M \times N$ parity check matrix H whose entries $h_{m,n}$, where $0 \le m \le (M-1)$ and $0 \le n \le (N-1)$, are field elements $a \in \mathrm{GF}(q)$. A valid codeword $\underline{x} = (x_0, x_1, \ldots, x_{N-1}), x_n \in \mathrm{GF}(q)$ is a vector in the nullspace of H. Practical decoding algorithms are based on the Tanner graph of this code, constructed by representing the rows of H with M check nodes, the columns of H with N variable nodes, and the connecting check node m and variable node n with an edge, weighted by $h_{m,n}$, if $h_{m,n} \in \{\mathrm{GF}(q) \setminus 0\}$. Let I_m denote the set of variable nodes that are adjacent to check node m and J_n denote the set of check nodes adjacent to variable node n. Regular NB-LDPC codes have constant check and variable node degrees, denoted by $d_c = |I_m|$ and $d_v = |J_n|$, respectively. Let $L_n(a)$ be the *a priori* channel information of variable node n corresponding to $a \in \mathrm{GF}(q)$, defined as

$$L_n(a) = \ln\left[\frac{\Pr(x_n = \hat{a})|\text{channel}}{\Pr(x_n = a)|\text{channel}}\right] \quad (1)$$

where $\hat{a}$ is defined to be the most likely field element for variable node n, i.e., $\Pr(x_n = a)$ is maximum when $a = \hat{a}$.

In iterative decoding algorithms of NB-LDPC codes, messages are passed back and forth between adjacent variable nodes and check nodes. Let $Q_{m,n}$ be the message from variable node n to check node m and $R_{m,n}$ be the message from check node m to variable node n. Let $Q_n(a)$ be the *a posteriori* information of variable node n for the field element a and y_n be the hard decision of variable node n.

A. Min–Max Decoding Algorithm

The MM algorithm [2] is a soft decoding algorithm with good coding gain and is defined as follows. The superscript k indicates the current iteration, and K is the maximum number of iterations allowed.

1) Initialization: The iteration index k is initialized to 0, and the *a posteriori* information Q_n as well as the messages from the variable nodes $Q_{m,n}$ are initialized to be equal to the *a priori* information.

2) Termination Check: A hard decision $\underline{y} = (y_0, y_1, \ldots, y_{N-1}), y \in \mathrm{GF}(q)^N$ is made and the syndrome $\underline{s} = (s_0, s_1, \ldots, s_{M-1}), s \in \mathrm{GF}(q)^M$ is computed

$$y_n = \underset{a \in \mathrm{GF}(q)}{\arg\min}\, Q_n(a); \quad \underline{s} = \underline{y} \times H^T. \quad (2)$$

If either $\underline{s} = 0$ or $k = K$, then $\underline{y}$ is given as the output. Otherwise, k is incremented by 1.

3) Check Node Processing: The messages from check nodes to variable nodes are updated with the forward–backward computations.

Forward Metrics ($i = \{0, 1, \ldots, d_c - 2\}$):

$$F_0(a) = Q_{m,n_0}\left(h_{m,n_0}^{-1} \times a\right) \tag{3}$$
$$F_i(a) = \min_{a' + h_{m,n_i} \times a'' = a}\left(\max\left(F_{i-1}(a'), Q_{m,n_i}(a'')\right)\right). \tag{4}$$

Backward Metrics ($i = \{d_c - 1, d_c - 2, \ldots, 1\}$):

$$B_{d_c-1}(a) = Q_{m,n_{d_c-1}}\left(h_{m,n_{d_c-1}}^{-1} \times a\right) \tag{5}$$
$$B_i(a) = \min_{a' + h_{m,n_i} \times a'' = a}\left(\max\left(B_{i+1}(a'), Q_{m,n_i}(a'')\right)\right). \tag{6}$$

Output Messages ($i = \{0, 1, \ldots, d_c - 1\}$):

$$R_{m,n_0}(a) = B_1(a) \tag{7}$$
$$R_{m,n_i}(a) = \min_{a' + a'' = -h_{m,n_i} \times a}(\max(F_{i-1}(a'), B_{i+1}(a''))) \tag{8}$$
$$R_{m,n_{d_c-1}}(a) = F_{d_c-2}(a). \tag{9}$$

4) Variable Node Processing: The messages from variable nodes to check nodes are updated

$$Q'^{(k)}_{m,n}(a) = L_n(a) + \sum_{m' \in J_n \setminus \{m\}} R^{(k)}_{m',n}(a) \tag{10}$$
$$Q^{(k)}_{m,n}(a) = Q'^{(k)}_{m,n}(a) - \min_{a' \in \mathrm{GF}(q)} Q'^{(k)}_{m,n}(a'). \tag{11}$$

In addition, the *a posteriori* information is updated:

$$Q_n(a) = L_n(a) + \sum_{m \in J_n} R^{(k)}_{m,n}(a). \tag{12}$$

5) Iteration: Go to step 2).

B. IHRB-MLGD Algorithm

The IHRB-MLGD decoding algorithm [7] is a hard decoding algorithm and is defined as follows. We note that in our definition, the reliability measures are defined so that a smaller value indicates higher likelihood.

1) Initialization: The iteration index k is initialized to 0, and the *a posteriori* information Q_n is initialized to be equal to either 0 if the symbol is the most likely one, or some (positive) value γ otherwise. The messages from the variable nodes $Q_{m,n}$ are initialized to be the most likely symbol itself.

2) Termination Check: This step is the same as the MM algorithm.

3) Check Node Processing: The messages from check nodes to variable nodes are updated

$$R^{(k)}_{m,n} = h_{m,n}^{-1} \times \bigoplus_{n' \in I_m \setminus \{n\}} \left(h_{m,n'} \times Q^{(k-1)}_{m,n'}\right) \tag{13}$$

where the $\oplus$ operator indicates summation in GF(q).

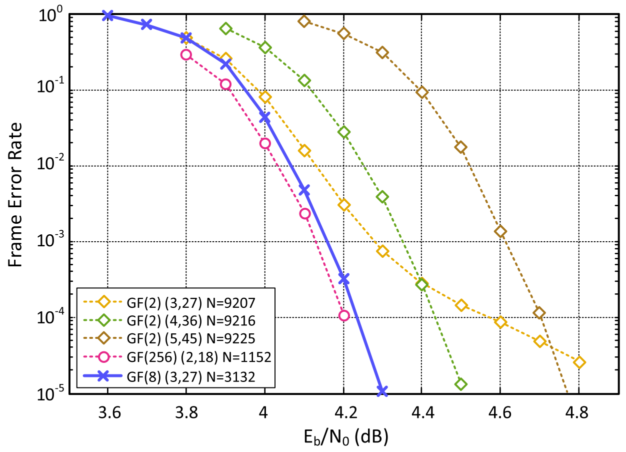


Fig. 1. FER Comparison between codes of various GF(q) and d_v. The length in bits is kept to be similar across codes. The binary codes are decoded with the min-sum algorithm, while the non-binary codes are decoded with the MM algorithm. The maximum number of iterations is set to 20. The GF(8) code has been simulated using our proposed techniques outlined in Sections III and IV.

4) Variable Node Processing: The *a posteriori* information is updated, and the messages from variable nodes to check nodes are correspondingly derived

$$\tilde{Q}^{(k)}_n(a) = Q^{(k-1)}_n(a) + \sum_{m' \in J_n} \begin{cases} 0, & \text{if } a = \arg\min_{a' \in \mathrm{GF}(q)} R_{m',n}(a') \\ \delta, & \text{otherwise} \end{cases} \tag{14}$$
$$Q^{(k)}_n(a) = \tilde{Q}^{(k)}_n(a) - \min_{a' \in \mathrm{GF}(q)} \tilde{Q}^{(k)}_n(a'), \tag{15}$$
$$Q^{(k)}_{m,n}(a) = \arg\min_{a \in \mathrm{GF}(q)} Q^{(k)}_n(a) \tag{16}$$

where δ is some positive number (usually 1).

5) Iteration: Go to step 2).

C. Choice of Code and GF Order

Our choice of GF(8) comes from a balancing act between coding gain improvements and hardware feasibility. We compare FER simulation results for binary codes, a GF(8), $d_v = 3$ code, and a GF(256), $d_v = 2$ code (Fig. 1). All codes have high-code rate and long length in bits (required for storage applications), but vary in GF order as well as column weight. No optimization was performed on the code construction besides guaranteeing a girth of 6. As expected, the non-binary codes outperform the binary codes by a significant margin (of note, the $d_v = 4$ code has the same column weight, code rate, and similar code length as codes compatible for the architecture presented in [11]). However, due to the choice of d_v, the GF(8) and GF(256) codes have similar observed performance, making the GF(8) code an appealing choice, knowing the complexity of implementing a decoder for very high-GF orders. Certainly, this simulation result is not definitive enough to claim that our design point at GF(8) is optimal. For example, various code optimization techniques (which are beyond the scope of this paper) will be able to improve the coding gain of each of these codes to some extent. Unfortunately, the GF(256), $d_v = 2$ code in [3] has a code length of <9% of our target of 8 kB, disallowing the application of their hardware solution to this particular code (similar for the GF(64) code in [4]). Thus, by proposing a hardware design of the GF(8) code, we create the opportunity for improved coding gain over binary codes in the application space of NB-LDPC codes for storage, which was previously not attainable.

Because our GF order is moderately low, the truncation of messages does not make sense; reducing the size of each message vector from 8 to $n_m < 8$ incurs a heavy penalty in the coding gain while introducing hardware overhead costs (sorters). It is noted that, therefore, our proposed architectural solution does not scale as well with the GF order and a different approach will be necessary to employ the ideas presented in this paper to a decoder of large GF(q).

III. Logarithmic Quantization Scheme

To achieve a cost-effective hardware implementation without paying a severe penalty in the coding gain, we propose a logarithmic quantization scheme which maintains a large-dynamic range even for a short wordlength [15]. Non-uniform quantization schemes have been proposed in the past for binary LDPC code decoders [16], but our objective here is to aggressively reduce the hardware cost, rather than to try to improve the coding gain.

A. Description of the Proposed Scheme

In a traditional, uniform quantization scheme, b bits are interpreted as an unsigned binary number (the LLRs in NB-LDPC decoding are non-negative), multiplied by some constant power of two to adjust for the location of the decimal point. In contrast, in our proposed logarithmic quantization scheme, all zeros represent the number 0, and the other numbers are successive powers of two. More specifically, the number X (interpreted as an unsigned integer) with b bits represents the number Y, related by:

$$Y = \begin{cases} 0 & X = 0 \\ 2^{\left(X-1-f\right)} & X \neq 0 \end{cases} \tag{17}$$

where f is a scaling factor which allows the control of the smallest and largest representable numbers. Some examples and comparison of uniform and logarithmic quantization schemes are shown in Table I. The logarithmic quantization scheme enhances the dynamic range dramatically while maintaining the ability to represent small numbers, at the cost of increasing the maximum rounding error for numbers of large magnitude.

B. Effects on Performance and Hardware

To observe the effect of the logarithmic scheme on the coding gain, we run simulations with the BPSK modulation over an additive white Gaussian noise (AWGN) channel to observe the FER (Fig. 2). The largest representable number is matched to ensure the absence of an early error floor [15]. The code employed is a quasi-cyclic GF(8) (327) code of length 3132 symbols, or 8352 information bits, which possesses the high-code rate and long-code length

TABLE I
Uniform and Logarithmic Quantization Scheme Examples

bits	ubdf	u5d2	u4d(-1)	lbdf	l3d1	l3d0
00. . .000	0	0	0	0	0	0
00. . .001	2^{-f}	0.25	2	2^{-f}	0.5	1
00. . .010	$2 \cdot 2^{-f}$	0.5	4	2^{1-f}	1	2
00. . .011	$3 \cdot 2^{-f}$	0.75	6	2^{2-f}	2	4
. . .	. . .	. . .	. . .	. . .	. . .	. . .
11. . .110	$(2^b - 2) \cdot 2^{-f}$	7.5	28	2^{2^b-3-f}	16	32
11. . .111	$(2^b - 1) \cdot 2^{-f}$	7.75	30	2^{2^b-2-f}	32	64

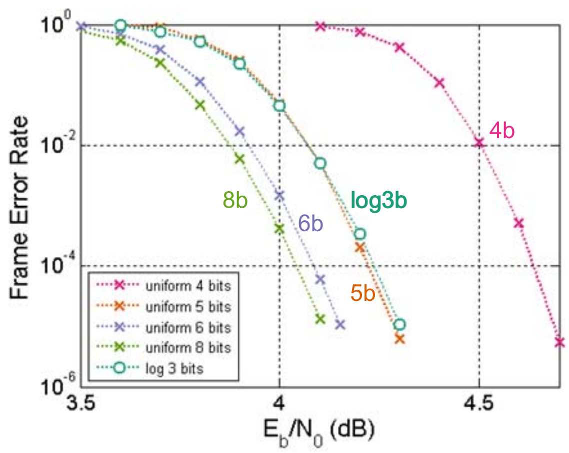


Fig. 2. FER comparison between uniform and logarithmic quantization schemes.

characteristics of LDPC codes employed in storage systems. We see that a logarithmic quantization scheme with three bits is enough to closely follow the performance of that of a 5-bit uniform quantization scheme. The 5-bit uniform scheme is in fact a common design choice in ASIC implementations as a tradeoff between performance and cost [3]–[6], making the coding gain of our proposed quantization scheme comparable with prior art. While there is a slight loss relative to an ideal case of uniform quantization schemes with more bits, this aggressive scaling down of wordlength for cost savings is impractical below 5 bits with the uniform quantization scheme, as is evidenced by the performance loss of the 4-bit uniform quantization scheme.

Such an aggressive reduction in the wordlength engenders great benefits in the hardware cost. First, NB-LDPC decoders for very long codes are necessarily partially serialized, since it is impractical to have a fully parallel architecture with as many variable and check node computation units as there are nodes in the Tanner graph. This implies the requirement for intermediate storage of the messages passed between these nodes. However, as the code becomes longer, the storage requirement also increases because the number of edges increases as well. This highly segmented storage often becomes a large portion of the total hardware cost of the decoder. When the wordlength is reduced, this storage requirement is directly alleviated and, thus, the overall cost is heavily relaxed. For example, the 3-bit logarithmic quantization scheme would save 40% in storage relative to a 5-bit uniform quantization scheme, with similar coding gain.

TABLE II
FPGA PROTOTYPE SYNTHESIS RESULTS

Bitwidth	Logic Util.	Combinational ALUTs	Message Memory Req.	Max Clock Freq.	Synthesis CPU Time (minutes)
4 uniform	44%	137,691 (32%)	590,976	83.1 MHz	142
5 uniform	53%	164,996 (39%)	738,720	72.5 MHz	316
6 uniform	61%	188,202 (44%)	886,464	70.7 MHz	482
3 logarithmic	41%	142,822 (34%)	443,232	90.9 MHz	111

The computational nodes are also affected by the quantization scheme. A non-uniform quantization scheme in general tends to complicate operations (for example, with floating-point computation). This is true for us as well in the variable nodes, since the addition of numbers in the logarithmic quantization scheme is not straightforward. Furthermore, we have found that applying the logarithmic quantization scheme to the channel input information greatly degrades the coding gain (not shown). Therefore, we choose to implement conversion blocks between uniform and logarithmic quantization schemes at the input and output of variable node computation units to enable the addition operations required within variable nodes. Fortunately, due to the simplicity of our proposed scheme, the conversion logic is quite simple, since the numbers that are representable in our logarithmic quantization scheme are one hot (or zero) in the uniform domain. Furthermore, the computations in the MM check nodes are comprised solely of comparison operations (minimums and maximums). Therefore, since the proposed logarithmic quantization scheme is *monotonic*, the check node unit implementation directly benefits from the reduced bitwidth without any overhead in complexity. In addition, the routing overhead corresponding to the edges in the Tanner graph is directly alleviated by the reduced bitwidth, improving the area utilization. All in all, we can expect a reduction in complexity by employing the logarithmic quantization scheme.

To validate this insight, we prototype the MM decoding algorithm on an Altera Stratix-IV field-programmable gate array (FPGA) with both the uniform and logarithmic quantization schemes. The synthesis results are summarized in Table II. It can be observed that the 3-bit logarithmic quantization scheme is beneficial relative to a 5-bit (and even 4-bit) uniform quantization scheme in terms of hardware resource utilization, but importantly, as mentioned earlier, without degrading the coding gain. In addition, the maximum achievable clock frequency is greater due to lower routing complexity. Finally, although not a direct measure of hardware cost, the time it takes for the synthesis software to finish is also reduced for the logarithmic scheme, which is another indicator of the achieved simplicity in hardware.

IV. PROPOSED DUAL-DECODING SCHEME

A. Motivation and Description of Scheme

It is difficult to yield an efficient hardware implementation solely based on soft decoding algorithms, such as the MM algorithm, because the severe complexity cannot be diminished too much before a non-negligible coding gain penalty arises. At the extreme, crude decoding methods, such as hard-decision-based algorithms, aimed at simplification generally yields extremely poor coding gain and potentially very high error floors. For example, the performance of hard-decision-based algorithms may be improved, such as with iterative soft reliability based (ISRB)-MLGD [7], but still falls far behind the performance of soft decoding algorithms (it is also noted that these algorithms perform quite poorly on codes with low to moderate column weights). We aim to achieve the best of both worlds: good coding gain and low error floor of the soft decoding algorithm and the computational efficiency of the hard decoding algorithm.

We propose the following dual-decoding strategy. First, a coarse decoding algorithm will attempt to decode the codeword based on the given input LLRs. If this coarse decoder fails and an error is detected, then we invoke the MM decoding algorithm on the same input LLRs. Note that this is different from the commonly employed technique (especially in NAND flash devices) of successively increasing the precision of input LLRs [10], where the *same* decoder (treated basically as a black box) initially attempts to decode on coarse channel information, and then, later on finer channel information. In our scheme, *different* decoders attack the same set of input LLRs in an attempt to improve the hardware characteristics of the decoder itself. For example, if at a particular SNR a hard decoder can achieve an FER of 10^{-2} and a soft decoder an FER of 10^{-8}, then although the hard decoder has terrible coding gain, the effort put into all the computations of the soft decoder was only necessary about 1% of the time. This strategy has the potential to yield great benefits in the hardware implementation, because most of the time the decoder will exhibit properties (power consumption and throughput) of the hard decoder, while the coding gain performance can approach that of the soft decoder, at the cost of a slight increase in worst case latency.

The IHRB-MLGD algorithm is a sensible initial candidate for our crude decoder. However, while it is quite simple and cheap to implement, the performance degradation relative to soft decoding algorithms is quite large. A larger variable node degree helps the performance of IHRB-MLGD [9], but this does not suit our needs. The final average computational complexity will be heavily influenced by the FER of the coarse decoding. However, in order for our proposed scheme to be effective, the FER of the coarse decoding must be sufficiently low at a reasonably low signal-to-noise ratio. Thus, to close this "gap," we propose several modifications to the IHRB-MLGD algorithm in order to improve the performance while maintaining the low computational complexity.

B. Augmented IHRB-MLGD Algorithm

There are two fairly simple modifications we can make that will improve the performance of the IHRB-MLGD algorithm. The proposed modified algorithm, which we call the augmented IHRB (A-IHRB)-MLGD, will better suit our needs as an initial crude decoding method. The first change to make is

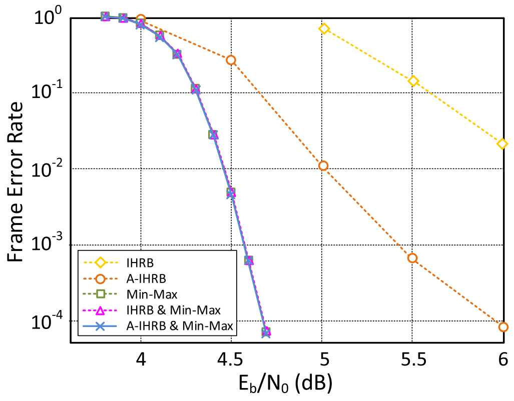


Fig. 3. FER comparison between IHRB-MLGD, A-IHRB, log-MM, IHRB&log-MM, and proposed A-IHRB&log-MM algorithms.

to utilize the soft input information from the channel, instead of initializing the channel information to be a simple vector consisting of 0 or γ as in Section II-B. This is traditionally avoided in implementations of IHRB-MLGD [9] from the perspective of the increased storage requirements. However, in our case, this is not an issue as the soft channel information is required by the MM algorithm anyways. The other change we propose is to differentiate each outgoing message of a variable node by excluding the incoming message from the target check node. Traditionally, this is avoided because it increases the storage requirement at the variable node linearly with the variable node degree d_v. We can curb this increase by maintaining a moderate d_v, since we are not seeking the final coding gain out of the coarse decoder itself. In addition, these two modifications make the variable node processing in the A-IHRB similar to that of the MM algorithm, which enables an efficient implementation in hardware (as will be discussed in Section V).

FER simulation results with the original IHRB-MLGD, our proposed A-IHRB, the MM (with log quantization), a scheme with the original IHRB-MLGD as the coarse decoder and log-MM as the soft decoder, and our proposed dual algorithm scheme with the A-IHRB as the coarse decoder and (log) MM as the soft decoder are all plotted in Fig. 3. The original IHRB-MLGD exhibits a large coding gain degradation, due to its simplistic nature as well as our relatively low d_v. Our two simple modifications bring the performance much closer to that of the MM algorithm, while still only being a hard-decision-based algorithm [dealing only with GF(q) symbols in the check nodes]. The dual-decoding scheme with the original IHRB-MLGD algorithm as the coarse decoder has the same performance as the MM algorithm, but the performance of the IHRB-MLGD is so poor that we would not reap any benefits in the hardware performance until the SNR is extremely high. Thus, the performance improvement of the A-IHRB is worth the small overhead price over the original IHRB-MLGD, because the initial coarse decoder performance will affect how often the soft decoder is invoked, which is in turn directly tied to the performance of the hardware implementation (throughput and power). Of course this algorithm would not suffice as a decoding algorithm by itself, but the shallower slope of the A-IHRB is not of vital importance to us since the residual errors will be corrected when the MM algorithm is invoked. It is also observed that the dual-algorithm scheme has essentially the same coding gain as that of the soft decoding algorithm. In addition, no undetectable errors were observed while simulating the A-IHRB within the range of simulated E_b/N_0 (plotted in Fig. 3) and corresponding FER (at least 100 frame errors were observed for each simulated point), due to the very long code length (the avoidance of undetectable errors in storage applications is, in general, one of the reasons for the long code length requirement). This is not to say, of course, that the absence of undetectable errors can be guaranteed; their likelihood depends not only on the algorithm but also on factors such as the parity-check matrix design and the channel model (higher confidence of the risk of undetectable errors can be obtained by techniques popular in industry, such as accelerated simulations with FPGA farms, which should be conducted after heavy code optimization and more accurate channel modeling, and is beyond the scope of this paper). Thus, the simulation results confirm the acceptability of using a coarse decoder upfront to reduce the invocation frequency of complex soft decoding (as a side note, simulation times were indeed reduced for the proposed dual-decoding approach).

V. Decoder Architecture

A. A-IHRB Decoder Scheduling

In message passing decoding schedules, variable and check node computations are conducted in separate phases due to the data dependencies. Translating this in hardware to operating different cores in separate clock cycles leads to inefficiencies, due to cores being idle while they wait for their inputs to be computed. In our dual-algorithm scheme, most frames are processed only by the A-IHRB algorithm, whose check node processing consists of simple bitwise XOR logic. Therefore, to avoid idle cycles, we move the A-IHRB check node operations into the same clock cycle as the variable node processing.

To enable this, we define a check node "state," $S_m^{(k)}$ as follows:

$$S_m^{(k)} = \bigoplus_{n' \in I_m} \left(h_{m,n'} \times Q_{m,n'}^{(k-1)}\right). \tag{18}$$

$S_m^{(k)}$ can be calculated in a serial manner, as $Q_{m,n'}^{(k-1)}$ is accumulated serially. In addition, once $S_m^{(k)}$ is calculated for check node m, the check-to-variable message $R_{m,n}^{(k)}$ for any adjacent variable node n [13] can be extracted by subtracting out the appropriate variable-to-check message from the previous iteration

$$R_{m,n}^{(k)} = h_{m,n}^{-1} \times \left(S_m^{(k)} \oplus \left(h_{m,n} \times Q_{m,n}^{(k-1)}\right)\right). \tag{19}$$

This insertion of combinational logic in the A-IHRB path (indicated by *CN* in Fig. 5) enables concurrent processing of the A-IHRB variable node and check node computations in the same clock cycle.

The scheduling of the A-IHRB check and variable node computations is shown in Fig. 4. The dark squares indicate non-zero elements of the parity-check matrix, and the circles

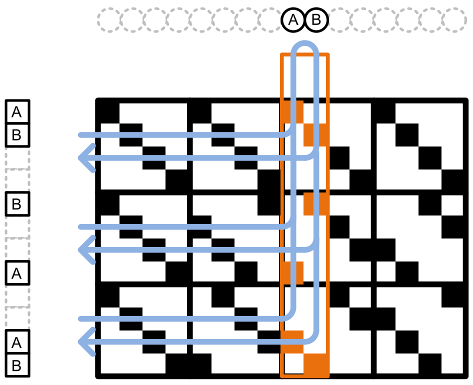


Fig. 4. Scheduling of A-IHRB variable and check node computations over the quasi-cyclic parity-check matrix structure.

and squares at the top and left are the variable and check nodes of the Tanner graph, respectively. In this example, there are two variable node computation units (A and B) working in parallel. First, the check-to-variable message is extracted from the state for each of the check nodes connected to the current variable node under consideration. These messages are then accumulated to conduct the variable node computation. Finally, the output of the variable node as well as a newly computed state for each of the rows is stored in memory for later use. The variable node computation units move over to the next set of variable nodes and the sequence of computations is repeated. There is no data collision within this sequence of moving data to and from memory, since due to the quasi-cyclic structure of the parity-check matrix, concurrently computed variable nodes will not share any check nodes as long as the parallelism factor divides the circulant matrix size evenly. However, we cannot insert pipelining to this datapath in a straightforward manner, because the state calculations must be finished before the next set of variable nodes start their execution when moving over to an adjacent circulant matrix. This unfortunately yields a relatively slow maximum operating frequency, as we will observe in Section VI. However, we are able to maintain a high efficiency by avoiding idle cores that would otherwise arise from the flooding decoding schedule.

B. Variable and Check Node Computation Units

The A-IHRB algorithm reduces check node computations to bitwise XOR logic. Therefore, the variable node operations are the computational bottleneck. Furthermore, the dual-algorithm scheme requires the implementation of two separate algorithms, which implies an area overhead cost. We have designed the A-IHRB algorithm such that many operations can be shared between it and the MM algorithm, which we must take advantage of in our implementation to lower the cost relative to implementing a variable node computation unit for each algorithm individually.

The variable node computation unit architecture is shown in Fig. 5. As described earlier, in the A-IHRB, the input messages from check nodes are extracted from the previous state and the previous outgoing message to that check node (top left), and as the current output message is computed, the current state is updated as well (middle right). The core

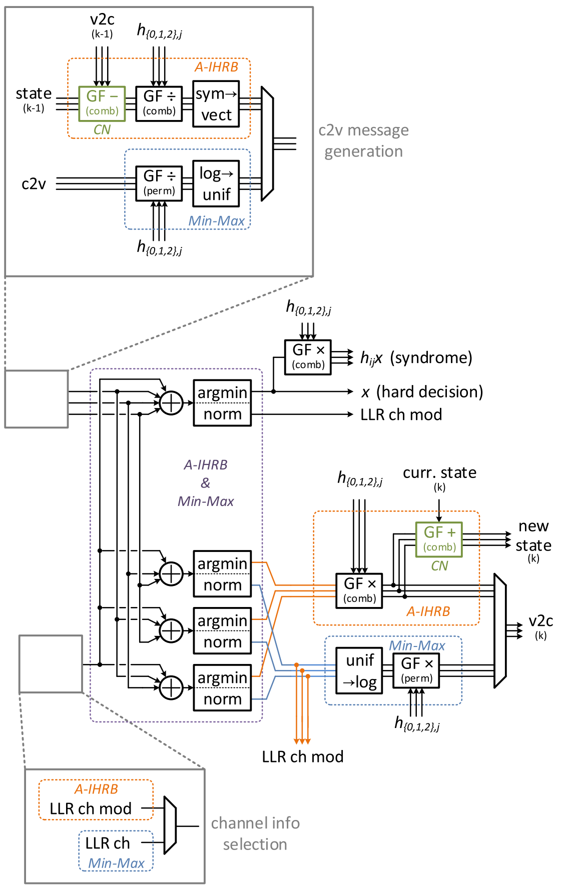


Fig. 5. Variable node computation unit architecture.

computations of the variable node in the A-IHRB consist of addition and argmin operations. On the other hand, the core computations of the variable node in MM consist of addition and normalization operations. Since normalization consists of finding the minimum, these operations between the two algorithms can be shared, as indicated in the center of Fig. 5, to reduce overhead for the most expensive portion of variable node processing. The "c2v message generation" and "channel info selection" blocks select the proper inputs to this shared computation portion, based on which algorithm is currently chosen. Messages in the A-IHRB domain are GF(8) symbols, which must be converted (by the sym→vect block) into a vector format, where each element is δ except for a single zero corresponding to the input GF(8) symbol, which supplies the core additions with the correct information. Messages in the MM domain are logarithmically quantized, which must be converted (by the log→unif block) into a uniformly quantized number in order to add them together, along with the channel input LLRs. Some of the adders are replaced by custom one-hot addition (OHA) logic (Fig. 6) to optimize the area and critical path. This is made possible due to the fact that the δ from (14) in an A-IHRB mode as well as the integer that has been converted into the linear domain from log domain in the MM mode are both one-hot numbers, for which a full-adder (FA) logic is redundant. The output is immediately converted back into the log domain (with rounding, by the unif→log block), keeping the uniformly quantized domain to a minimum in order to reduce the storage and routing

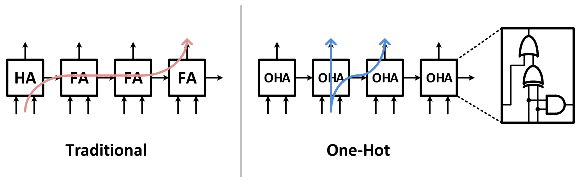


Fig. 6. Custom OHA logic versus a traditional ripple-carry adder with FA and half-adder (HA) cells and their respective critical paths.

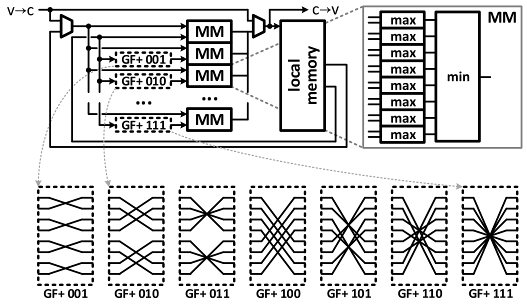


Fig. 7. Check node computation unit for the MM algorithm.

overhead requirements, which are proportional to the bitwidth. The decoding algorithm also requires multiplications in GF(8) when messages are passed between the check and variable nodes. In the MM path, LLR vectors being multiplied by some GF element is equivalent to a permutation, which is implemented by a series of multiplexors. In the A-IHRB path, the GF(8) multiplication (or division) on a symbol is a simple combinational logic (consisting primarily of XOR gates).

The check node computation unit for the MM algorithm is implemented in a fairly straightforward manner (Fig. 7). The forward-backward computations of (3)–(9) are a series of minimum-of-maximum operations, implemented as eight pairwise maximums followed by an 8-input minimum. One of the input vectors are added (permuted, as shown) by every GF(8) element, to generate each of the 8 output messages. The intermediate forward and backward results are stored in the local memory. The left mux allows for the appropriate combination of inputs to feed into the MM blocks: Q and F_{i-1} to generate F_i [(4)], Q and B_{i+1} for B_i [(6)], and F_{i-1} and B_{i+1} for R [(8)]. The right mux allows for the initialization of F_0 and R_{d_c-1} in the local memory.

C. Storage

In the partially parallel decoder architecture, storage of intermediate messages is a large cost in the overall area. The largest memories are those related to the MM algorithm whose messages length q vectors of b-bit LLRs, rather than the A-IHRB algorithm whose messages are GF(q) elements represented by $log(q)$ bits. To store $Q_{m,n}$ [(11)], a total of $B_{\text{Qmn}} = q \times b_{\text{log}} \times N \times d_v$ bits are necessary. Since the messages are stored in the log-quantized domain ($b_{\text{log}} = 3$), $B_{\text{Qmn}} = 225\,504$. B_{Rmn}, the storage requirement for $R_{m,n}$ [(7)–(9)], is the same size. While a separate memory is not required for the messages between nodes for the A-IHRB since the memory can be reused, explicit storage for the *a posteriori* information of the A-IHRB [equivalent to Q_n of IHRB-MLGD in (15)] is necessary, because the δ values are accumulated over iterations [(14)]. The requirement for storage of these values is $B_{\text{Qn}} = q \times b_{\text{lin}} \times N = 150\,336$, where $b_{\text{lin}} = 6$. Finally, the storage of channel information L_n [(1)] is also $B_{Ln} = q \times b_{\text{lin}} \times N$.

Not only are a large number of bits required but also the intermediate message storage must be highly segmented in this architecture. In our particular design, we must have $d_v \times P$ memory instances (or access ports), where P is the parallelism factor (the number of variable node instances), to fetch d_v messages from check nodes into each of the P parallel variable nodes, and to have the P parallel instances write d_v messages back. The high segmentation is required for the bandwidth and access granularity, but comes at a cost in area. Because each memory block becomes rather small [for example, $B_{\text{Qmn}}/(d_v \times P) = 1296$ bits], an SRAM storage does not scale well. Thus, we choose to implement the storage with register-based memories instead of SRAMs to avoid the overhead cost of address decoders, sense amplifiers, and so on. In combination with the clock gating (automated by synthesis tools), this yields a low-power design.

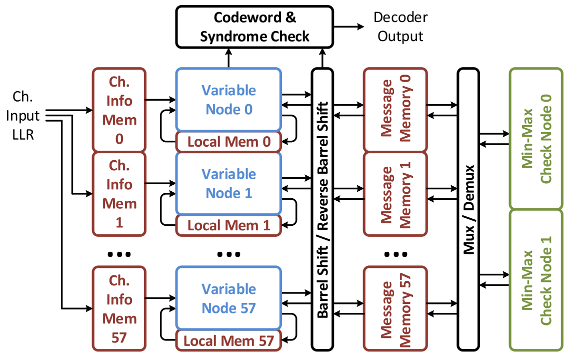


Fig. 8. Overall architecture of decoder.

D. System Architecture of Decoder

The overall architecture of the decoder is shown in Fig. 8. A total of $P = 58$ variable node computation units (which include the A-IHRB check node computations) are placed in parallel to enable a high throughput, which is made possible due to the quasi-cyclic nature of the parity-check matrix. Because the memories consume a significant portion of the area, a higher parallelism would have enabled higher throughput and better area efficiency. This is because, for example, doubling the number of variable node computation units doubles the overall throughput as well as the variable node area, but only doubles the *segmentation* of the memories, which would not cause the area to grow nearly as much because the memory is already highly segmented. The maximum possible parallelism factor without data hazards is the size of the circulant matrix, which we would have adopted in our design had we not been tightly constrained in available silicon area. Data between the variable node unit and message memories are passed through barrel shifters (again because of the quasi-cyclic code). The MM check node units are activated only when the MM algorithm is invoked. For the purposes of this proof-of-concept architecture, only two such units are instantiated (again, due to our area constraint), but if the worst-case latency must be improved, this number can be increased.

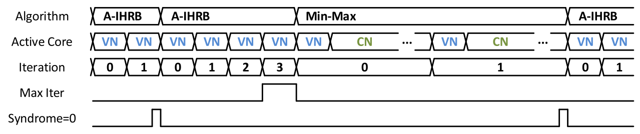


Fig. 9. Phases of decoder operation, assuming four maximum iterations in A-IHRB.

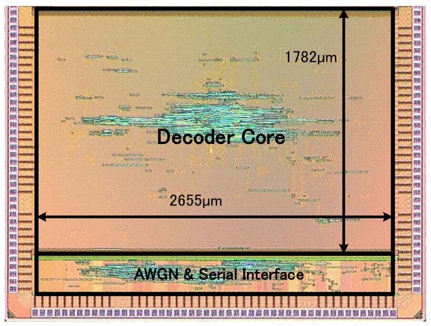


Fig. 10. Chip micrograph.

The message memory is shared in use between the A-IHRB and MM modes to store data that is generated by one node and processed by another. Local memory at each variable node unit exists to store data only used by this particular variable node, such as the outgoing message in the A-IHRB mode.

The phases of operation are shown in Fig. 9. The A-IHRB will initially attempt to decode the input and will move on to the next codeword if successful (syndrome = 0). If the maximum number of iterations is reached, the MM algorithm will be invoked. When either the input is decoded properly or the maximum number of iterations is reached, the next frame will start, again with the A-IHRB algorithm. Since each block of 58 variable nodes only takes one clock cycle to process and there are 3132 variable nodes in total, the entire variable node phase completes in 54 clock cycles. In the A-IHRB mode, this also includes the check node computations and, thus, each *iteration* of the algorithm completes in 54 clock cycles.

VI. Chip Implementation Results

A proof-of-concept chip is taped out in a 40-nm low power (LP) process. The chip micrograph is shown in Fig. 10. The decoder core measures 1782 μm$\times$ 2655μm for an area of 4.73 mm^2 (it is noted that a simple scaling of this design to higher GF orders would probably be prohibitively expensive in area). The low threshold voltage (LVT) transistors and standard cell library are used. The chip contains a serial-parallel interface in the test circuitry and communicates via this interface to an FPGA board for testing. An on-chip AWGN (and corresponding LLR) generator is utilized to run FER simulations with the all-zero codeword as input. The measured FER as well as the average number of iterations are plotted in Figs. 11 and 12. The "raw" FER of the A-IHRB is also measured and plotted. The maximum number of iterations for the MM algorithm can be tuned to tradeoff the FER and the maximum latency. No error floor is observed down to an FER of $\sim 10^{-8}$ (each frame is 9396 coded bits long). The average number of iterations for the MM algorithm tends to converge to a single line, because error events are rare.

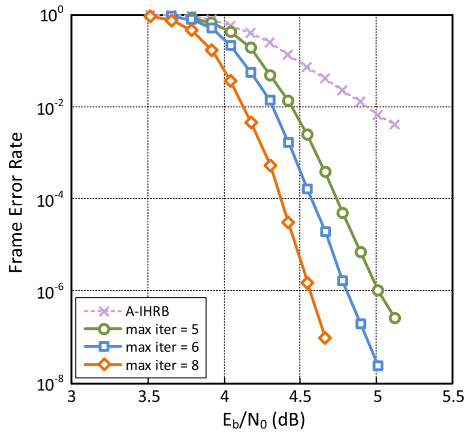


Fig. 11. FER measurement results.

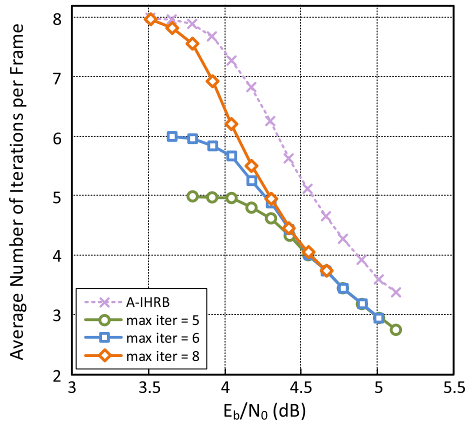


Fig. 12. Average number of iterations per frame measurement results.

The hardware performance, such as the average throughput and energy efficiency, depends mostly on the FER of the A-IHRB and is relatively unaffected by changing the maximum number of iterations of MM. The operating principle that we have discussed makes this clear; the complex MM algorithm is only invoked at the detectable FER of the A-IHRB. Based on Fig. 11, we choose to make chip measurements at $E_b/N_0 = 5$ dB, with six maximum MM iterations, which achieves an A-IHRB FER of 10^{-2} and an overall FER of 10^{-8}. This decision is somewhat arbitrary; our parameter choices are driven more by the demonstrability of our technique in correspondence with the FER curve and not so much the absolute performance itself. For example, the hardware can exhibit similar characteristics even as the maximum number of iterations of MM is increased and the FER becomes unobservable (we try to observe at least 100 frame errors for each simulation point), but the fact that the FER is too low to be simulated is not related to the average throughput or energy efficiency. With these parameters, at a nominal supply voltage of 1.2 V, we achieve 2.551-Gb/s coded throughput, which is equivalent to a 2.267-Gb/s information throughput. The clock frequency is 125 MHz, and the core power consumption is 212.4 mW. This operating point gives us an energy efficiency of 93.7 pJ/bit. To improve the energy efficiency of energy per decoded bit, we can scale the supply voltage and operating clock frequency (Fig. 13). The chip is functional at a minimum supply voltage of 0.65 V and a clock

TABLE III
COMPARISON WITH PRIOR ART

	TMAG 2007 [11]*	TCAS 2012 [5]†	TVLSI 2013 [6]*	TVLSI 2014 [3]*	JSSC 2015 [4]		This work	
Code length (information bits)	8,192	682	3,630	704	480		8,352	
Column weight	4	4	4	2	2		3	
Code rate	0.89	0.55	0.87	0.8	0.5		0.89	
Galois field q	binary	GF(32)	GF(32)	GF(256)	GF(64)		GF(8)	
Decoding algorithm	Min-Sum	Sel.-Input Min-Max	Relaxed Min-Max	RTBCP	Truncated EMS		A-IHRB & Min-Max	
Quantization (bits)	4+1 (sign)	5-7	5	5	5		3 (log)	
Technology	65 nm	90 nm	180 nm	28 nm	65 nm		40 nm	
Core area (mm^2)‡	2.32 (0.88)	10.33 (2.04)	-	1.289 (2.63)	7.04 (2.67)		4.73	
Utilization	-	-	-	75.7%	87%		89.5%	
Supply voltage (V)	0.9	-	-	-	0.675	1.0	0.6	1.2
Clock frequency (MHz)	300	260	200	520	400	700	30	120
Coded throughput (Mb/s)	2,100	47.69	66	546	698	1,221	612	2,551
Information throughput (Mb/s)	**1867**	**26.2**	**57.2**	**436.8**	**349**	**611**	**544**	**2267**
Power (mW)	**-**	**479.8**	**-**	**976**	**729**	**3704**	**12.35**	**212.4**
Area efficiency (Mb/s/mm^2)‡	**804.7 (2125.0)**	**2.54 (12.84)**	**-**	**338.87 (166.05)**	**49.57 (130.91)**	**86.79 (229.18)**	**115.1**	**479.4**
Energy efficiency (pJ/b)	**-**	**18313**	**-**	**2234**	**2089**	**6062**	**22.7**	**93.7**

* Synthesis estimates.
† P&R estimates.
‡ Numbers in parentheses are normalized to 40nm (assumes no supply scaling)

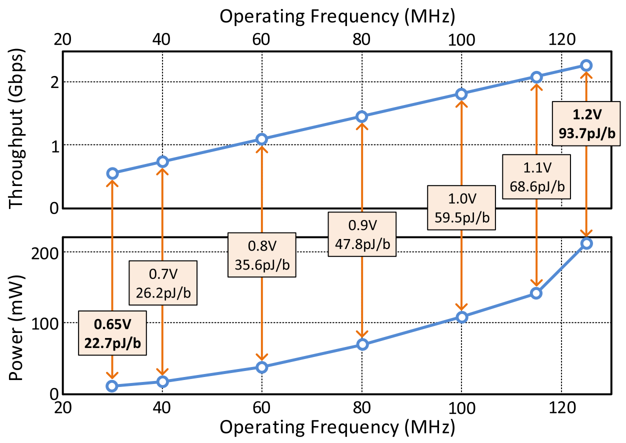


Fig. 13. Throughput versus operating frequency, and power versus operating frequency (points labeled with supply voltage and energy efficiency).

frequency of 30 MHz, where the throughput is 544.2 Mb/s and the power consumption is 12.35 mW, resulting in a minimum energy efficiency of 22.7 pJ/bit. These measurement results are fairly consistent across multiple chips, and the results of Fig. 13 are based on the chip with worst case measurements (some shmoo plots are shown in Fig. 14).

A comparison of our ASIC results with prior art is shown in Table III. In somewhat older works (such as [5] and [6]), code parameters tended not to be sacrificed for cheaper hardware. Therefore, the column weight is high ($d_v = 4$) but there exists a multiple-orders-of-magnitude gap in the information throughput and the energy efficiency of these

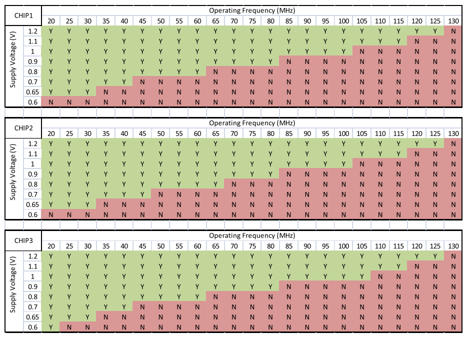


Fig. 14. Shmoo plots for several measured chips.

works and binary LDPC decoders (see [11]). In more recent works, such as [3] and [4], decoders for higher order GF(q) codes are achieved with moderate throughputs. However, the power consumption and correspondingly the energy efficiency are both still quite poor. Furthermore, as discussed earlier, code parameters, such as the code length and column weight, are sacrificed to achieve the impressive hardware performance, which is a poor tradeoff against simply choosing a binary LDPC decoder. In addition, these solutions simply do not provide a viable implementation of decoders for codes whose parameters have been chosen for application in storage systems. The hardware cost (such as area) associated with our design as well as other NB-LDPC designs relative to binary decoders unfortunately leads us to believe that NB-LDPC codes are inadequate for other applications, such as wireless communications, where binary LDPC codes would suffice, since the required error rates are not very low. However, as motivated in Section I, the high demand for stronger codes with sharp waterfall curves and low error floors previously unachievable by binary LDPC codes, applied at extremely low error rate regions provides a substantial push toward the adoption of NB-LDPC codes as a solution, enabling the increase of storage densities and potentially lowering the overall cost of storage, even if the cost of implementing the error correction code increases. Our code boasts a high code length, moderate column weight, and high code rate, and outperforms the binary LDPC with similar characteristics. The hardware is able to achieve very high information throughputs while consuming very little power, yielding a highly energy efficient design. This paper is the first in NB-LDPC decoders to the best of our knowledge to demonstrate greater than 2-Gb/s information throughput and better than 100-pJ/bit energy efficiency.

VII. Conclusion

In this paper, we presented a 2.267-Gb/s information throughput 212.4-mW NB-LDPC decoder employing a code suited for storage applications. An aggressive reduction in the bitwidth of LLR messages within the decoding algorithm is made possible due to our proposed logarithmic quantization scheme, greatly decreasing the message storage requirement as well as some of the node computation core costs. The effectiveness of this strategy was validated through FPGA prototyping. In addition, a dual-algorithm scheme was proposed to relax the average computational complexity of the decoder while maintaining the superior coding gain made available by the NB-LDPC soft decoding. To this end, algorithmic reformulations were proposed to achieve a moderate coding gain based on hard-decision decoding. The architectural design allowed for the high throughput while simultaneously alleviating the overhead for implementing two separate algorithms. The application of our proposed techniques to our proof-of-concept chip allowed a realization of unprecedented throughput and energy efficiency in NB-LDPC code decoders, bringing such codes closer to actual deployment in a practical application.

Acknowledgment

The authors would like to thank A. Hareedy and Prof. L. Dolecek of the Electrical and Computer Engineering Department, University of California at Los Angeles, as well as Dr. K. Harada and Dr. H. Uchikawa of Toshiba Memory Corporation for helpful technical discussions.

References

[1] M. C. Davey and D. MacKay, "Low-density parity check codes over GF(q)," *IEEE Commun. Lett.*, vol. 2, no. 6, pp. 165–167, Jun. 1998.

[2] V. Savin, "Min-max decoding for non binary LDPC codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2008, pp. 960–964.

[3] J. Lin and Z. Yan, "An efficient fully parallel decoder architecture for nonbinary LDPC codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 12, pp. 2649–2660, Dec. 2014.

[4] Y. S. Park, Y. Tao, and Z. Zhang, "A fully parallel nonbinary LDPC decoder with fine-grained dynamic clock gating," *IEEE J. Solid-State Circuits*, vol. 50, no. 2, pp. 464–475, Feb. 2015.

[5] Y.-L. Ueng, C.-Y. Leong, C.-J. Yang, C.-C. Cheng, K.-H. Liao, and S.-W. Chen, "An efficient layered decoding architecture for nonbinary QC-LDPC codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 2, pp. 385–398, Feb. 2012.

[6] F. Cai and X. Zhang, "Relaxed min-max decoder architectures for nonbinary low-density parity-check codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 11, pp. 2010–2023, Nov. 2013.

[7] C.-Y. Chen, Q. Huang, C.-C. Chao, and S. Lin, "Two low-complexity reliability-based message-passing algorithms for decoding non-binary LDPC codes," *IEEE Trans. Commun.*, vol. 58, no. 11, pp. 3140–3147, Nov. 2010.

[8] C. Xiong and Z. Yan, "Low-complexity layered iterative hard-reliability-based majority-logic decoder for non-binary quasi-cyclic LDPC codes," in *Proc. IEEE Symp. Circuits Sys. (ISCAS)*, May 2013, pp. 1348–1351.

[9] X. Zhang, F. Cai, and S. Lin, "Low-complexity reliability-based message-passing decoder architectures for non-binary LDPC codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 11, pp. 1938–1950, Nov. 2012.

[10] K. Zhao, W. Zhao, H. Sun, T. Zhang, X. Zhang, and N. Zheng, "LDPC-in-SSD: Making advanced error correction codes work effectively in solid state drives," in *Proc. USENIX Conf. File Storage Technol.*, Feb. 2013, pp. 244–256.

[11] H. Zhong, W. Xu, N. Xie, and T. Zhang, "Area-efficient min-sum decoder design for high-rate quasi-cyclic low-density parity-check codes in magnetic recording," *IEEE Trans. Magn.*, vol. 43, no. 12, pp. 4117–4122, Dec. 2007.

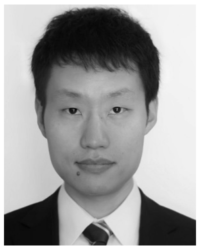
[12] A. Hareedy, B. Amiri, R. Galbraith, and L. Dolecek, "Non-binary LDPC codes for magnetic recording channels: Error floor analysis and optimized code design," *IEEE Trans. Commun.*, vol. 64, no. 8, pp. 3194–3207, Aug. 2016.

[13] A. Hareedy, C. Lanka, and L. Dolecek, "A general non-binary LDPC code optimization framework suitable for dense flash memory and magnetic storage," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 9, pp. 2402–2415, Sep. 2016.

[14] D. Declercq and M. Fossorier, "Decoding algorithms for nonbinary LDPC codes over GF(q)," *IEEE Trans. Commun.*, vol. 55, no. 4, pp. 633–643, Apr. 2007.

[15] Y. Toriyama, B. Amiri, L. Dolecek, and D. Marković, "Logarithmic quantization scheme for reduced hardware cost and improved error floor in non-binary LDPC decoders," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2014, pp. 3162–3167.

[16] X. Zhang and P. H. Siegel, "Quantized min-sum decoders with low error floor for LDPC codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Sep. 2012, pp. 2871–2875.



Yuta Toriyama (M'13) received the B.S. degree in electrical engineering and computer science from the University of California at Berkeley, Berkeley, CA, USA, in 2009, and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Los Angeles (UCLA), Los Angeles, CA, USA, in 2011 and 2016, respectively.

He has held internship positions at Broadcom, Irvine, CA, USA, in 2011, and Toshiba, Yokohama, Japan, in 2014. In 2017, he joined Qualcomm Technologies, Inc., San Diego, CA, USA, as an Image Signal Processing (ISP) System Architect.

Dr. Toriyama was a recipient of the UCLA Electrical Engineering Departmental Fellowship in 2009 and the Broadcom Fellowship in 2013.

Dejan Marković (M'04) received the Ph.D. degree from the University of California at Berkeley, Berkeley, CA, USA, in 2006.

He co-founded Flex Logix Technologies, Mountain View, CA, USA, a semiconductor intellectual property (IP) startup, in 2014. He is currently a Professor of electrical engineering with the University of California at Los Angeles, Los Angeles, CA, USA, where he is also a Co-Chair of neuroengineering with the Bioengineering Department. His current research interests include implantable neuromodulation systems, domain-specific architectures, embedded systems, energy harvesting, and design methodologies.

Dr. Marković received the 2007 David J. Sakrison Memorial Prize for the Ph.D. degree and the National Science Foundation (NSF) CAREER Award in 2009. In 2011, he was a co-recipient of the 2010 International Solid-State Circuits Conference (ISSCC) Jack Raper Award for Outstanding Technology Directions. He received the 2014 ISSCC Lewis Winner Award for Outstanding Paper.